Discutindo soluções do clássico problema da sujeira com agentes inteligentes, *Java* e *Jess*

ANDERSON DE REZENDE ROCHA[1]
ANTÔNIO MARIA PEREIRA DE RESENDE[1]
(ORIENTADOR)

[1] UFLA - Universidade Federal de Lavras
DCC - Departamento de Ciência da Computação
Campus Universitário, CEP 37.200-000, Lavras (MG)
{undersun,tonio}@comp.ufla.br

24 de junho de 2003

Resumo: Soluções do problema clássico da sujeira, *dirty catch problem* [Weiss, 1999], são apresentadas sob várias égides aos iniciados em inteligência artificial [Rich, 1993]. Apresenta-se neste trabalho uma forma a mais, díspar, instigante.

Ao unir o poder da linguagem *Java* [Sun©, 2002] e a facilidade de um pacote de gerenciamento de conhecimentos, *Jess* [SandiaLabs, 2001], tem-se em mãos o poder de simular, sem perda de generalidade, o interessante mundo dos agentes inteligentes [Sichman e Alvares, 1997] bem como a solução de seus problemas. Neste artigo o leitor terá a possibilidade de conhecer mais uma vertente de um mundo multifacetado: o mundo da *Inteligência Artificial*.

Palavras-chave: *Problema da sujeira, Agentes Inteligentes, Java, Jess.*

1 Introdução

A simulação do conhecimento humano vem sendo, desde a gênese da Inteligência Artificial, uma obsessão dos cientistas do ramo. Sabe-se hoje, devido a exaustivos estudos, que tal conhecimento é, em sua essência, fruto de ações neurais cerebrais. Estas ações proporcionam ao cérebro humano a capacidade de realizar inúmeras atividades com mais habilidade que os computadores. Figuram-se dentre tais atividades a realização de inferências, raciocínio lógico, reconhecimento de padrões, processamento de imagens...

Devido a estas atividades em que os *homo-sapiens* são melhores que os computadores, muitas áreas de estudo, principalmente na Inteligência Artificial, foram criadas com o intuito de suprir tais falhas dos sistemas computacionais. Uma destas áreas e, de particular interesse para os autores deste artigo, é a pesquisa em inferências lógicas com agentes inteligentes.

Como conseguir aprender os conceitos deste ramo de maneira inteligível? Uma das formas mais estimulantes é simulando uma situação real. Assim, procurou-se resolver um problema clássico na literatura de I.A. de uma maneira acessível ao público leigo.

2 História

Historicamente a I.A. têm apresentado muitos problemas interessantes de se pensar. Obviamente, muitos desses problemas seriam trivialmente solucionados pela mente humana. Entretanto, a pergunta é: como se comportaria um computador diante da apresentação a tais problemas? Os computadores seriam capazes de resolver tais problemas em tempo hábil e de maneira satisfatória?

Um dos problemas mais conhecidos e divulgados é o problema da coleta de sujeira, conhecido na literatura como *Dirty Catch Problem*. O problema consiste em se criar um ambiente aleatório, uma sala por exemplo, com vários pontos de sujeira localizados randomicamente na mesma. Em seguida, procura-se alocar nesta sala, vários agentes chamados coletores de sujeira capazes de deixar o ambiente limpo. Além disso, os ditos agentes devem ser capazes de se desviar dos obstáculos encontrados bem como sua maior prioridade deve ser recolher focos de sujeira ignorando quaisquer outros focos ignotos [Weiss, 1999]. Este problema, quando mapeado para ser resolvido por um computador, torna-se complexo. A principal limitação reside num dilema inerente da Inteligência Artificial:

> *Um sistema de IA precisa conter muito conhecimento para poder lidar com outras coisas além de problemas triviais, em "domínios de brinquedo". Mas, com o aumento da quantidade de conhecimento, fica mais difícil acessar as coisas apropriadas quando necessário, e, portanto, mais conhecimentos têm de ser incluídos para ajudar. Conseqüentemente, haverá mais conhecimentos a serem gerenciados e, portanto, mais conhecimentos precisam ser acrescentados e assim ininterruptamente* [Pleiad, 1992].

3 Definição de agentes

Sabe-se que, dado um determinado sistema, denomina-se *agente* cada uma de suas entidades ativas. O conjunto de agentes neste ambiente constitui uma *sociedade*. Tal ambiente é formado pelas entidades passivas. A um agente cabe "raciocinar" sobre o ambiente, sobre outros agentes bem como decidir racionalmente objetivos a seguir e ações a tomar [Ferreira, 2002].

De acordo com [Rocha, 2002] ativo é aquele que exerce a ação, que age. Claramente o termo agente está intimamente ligado a um conjunto de regras e bases de conhecimento além de um mecanismo de controle para sua ativação como pode ser acompanhado na figura 1.

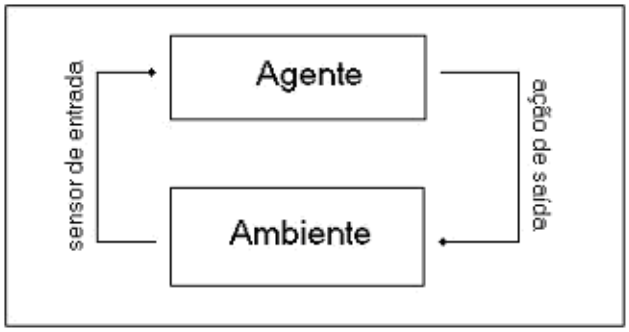

Figura 1: Estrutura do Robô

Deste modo pode-se inferir que um agente se assemelha a um processo cerebral. Conseqüentemente uma sociedade em busca da solução de um problema seria o cérebro com seus inúmeros processos também resolvendo um problema.

4 O estado da arte

Os autores conseguiram, neste projeto[1], simular uma resolução do problema utilizando a linguagem de programação *Java* juntamente com um pacote para gerenciamento de regras de conhecimento, *Jess*[2], também feito em *Java*, pelo SANDIA LABORATORIES [SANDIALABS, 2001], órgão do governo federal americano.

A linguagem de programação *Java* foi escolhida devido a vários aspectos relevantes tais como:

- **Portabilidade**. A linguagem é portável entre vários sistemas operacionais conhecidos dentre os quais Linux, Windows©, Solaris© entre outros. Há também a relativa facilidade em se estender as aplicações desenvolvidas para *desktop* para aplicativos *web*;
- **Documentação**. Possui uma documentação completa com vários exemplos e tutoriais muito explicativos;
- **Inteligibilidade**. Dentro dos paradigmas das linguagens de programação esta é classificada como de *alto nível*. Deste modo, tende a ser mais compreensível pela maioria dos pesquisadores atualmente.

Da mesma forma, escolheu-se a plataforma *Jess* devido a:

- **Simplicidade**. Por ser um ambiente se simulação acadêmico *Jess* tem uma base de conhecimentos simples e compacta. Isto facilita a representação de regras de conhecimento em domínios limitados de problemas;
- **Documentação**. Possui uma documentação completa com vários exemplos e tutoriais muito explicativos;
- **Suporte técnico**. Possui um corpo técnico sempre disposto a esclarecer as dúvidas pertinentes;
- **Código livre**. Todo o pacote de representação e gerenciamento de conhecimentos é de código livre. Isto habilita fortemente a possibilidade de trabalhar-se diretamente sobre os motores de inferência.

Após a escolha destas ferramentas, foram desenvolvidas 5 (cinco) simulações para o problema abordado. A primeira delas emprega primitivas de resolução em um agente tal qual o cérebro utilizaria com um único processo neural. Três simulam uma arquitetura centralizada de processos. Finalmente, a quinta simulação enfatiza a resolução de problemas utilizando uma arquitetura distribuída de processos.

Sem qualquer perda de generalidade, em todas as simulações o termo agente foi estendido para robô. Por robô entende-se a união de uma base de conhecimentos e seus atuadores, responsáveis por suas ações no ambiente [Brooks, 1986]. Tais ações podem ser: locomoção, o reconhecimento de objetos, a detecção de obstáculos...

Todas as bases de conhecimento foram escritas em *Jess* e os atuadores dos robôs escritos em *Java*.

O ambiente das simulações foi representado como uma matriz quadrada **NxN** onde **n** denota o número de linhas ou colunas. As simulações foram tratadas discretamente, ou melhor, por pontos inteiros. Uma mini-representação do ambiente encontra-se na figura 2.

Se um robô estiver totalmente no centro (em destaque) diz-se que ele ocupa a posição 1:1, ou seja, linha 1, coluna 1 da matriz de ambiente.

Para representação dos robôs foi adotado o modelo da funcionalidade emergente [Weiss, 1999] utilizando-se as concepções da arquitetura de subsunção, do inglês *Subsunction Architecture* [Sichman e Alvares, 1997]. Adicionalmente, foram abstraídos alguns princípios da modelagem segundo a eco-resolução [Brooks, 1986].

Nas simulações em que aparecem mais de um robô no ambiente é necessário dizer que cada robô tem consciência própria, i.e., tem sua própria base de conhecimentos e princípios de raciocínio. Importante salientar que os princípios de conhecimento são baseados em pares **(conhecimento, ação)** [Weiss, 1999]. Assim sendo, todas as decisões de um robô não afetam decisões dos companheiros na sua sociedade. Para que um robô saiba de

[1] Financiando pelo CNPq no período de 01/08/2001 a 31/07/2002.
[2] *Java Expert System Shell*

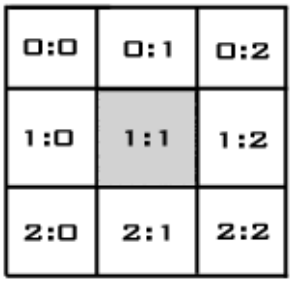

Figura 2: Representação da matriz de ambiente

alguma decisão por parte de um companheiro é necessário que ambos se comuniquem através de seus módulos de comunicação, quando implementados.

O uso de um pacote de gerenciamento de regras de conhecimento nestas simulações é justificável porque cada agente não conhece o ambiente em que se encontra. Pode-se dizer que o ambiente não é totalmente acessível o que torna este problema um problema de múltiplos estados [Russel e Norvig, 1995].

De forma geral, todas as simulações implementam as funções básicas de movimento e captura de sujeira. Assim, torna-se necessária em toda a simulação estabelecer regras de:

1. Locomoção;
2. Giro em torno de si mesmo;
3. Captura de sujeira;
4. De parada;
5. Inferências sobre melhores locais a visitar.

As primeiras quatro regras poderiam, obviamente, ser implementadas com igual funcionamento sem utilização de um gerenciador de conhecimentos. No entanto, o conjunto de regras apresentado por último (*Inferências*) são próprios para implementação utilizando-se bases de conhecimento. Isto facilita, por exemplo, a recursividade, *backtracking*, aprendizado entre outros.

Cada uma das simulações é apresentada em detalhes a seguir. Mais informações podem ser encontrados em [Rocha, 2002].

4.1 1ª simulação: Robô alone não deliberativo (RAND)

O nome *robô alone* foi dado devido ao fato de o robô atuar sozinho, do inglês *alone*, neste ambiente. O termo "não deliberativo" significa que não há raciocínio lógico na execução das tarefas apresentadas ao robô. Este robô é determinístico dado que sempre tem o mesmo caminho na procura da solução do problema.

O RAND foi desenvolvido utilizando as seguintes regras de conhecimento:

1. Regra *existe sujeira*, se a posição onde o robô estiver possuir sujeira ▷ Limpar.
2. Regra *não existe sujeira*, caso a posição onde o robô estiver não possuir sujeira ▷ seguir em frente.
3. Regra *andar*, dividida em 4 regras menores:
   - Seguir norte
   - Seguir sul

- Seguir leste
- Seguir oeste

4. Regra *virar*, divida em 4 regras menores:
    - Virar norte
    - Virar sul
    - Virar leste
    - Virar oeste
5. Regra *determinar final*, se o ambiente foi totalmente percorrido ▷ finalizar robô.

Tudo o que é decidido pela base de conhecimentos do robô é passado aos seus atuadores. Para que isto seja possível existe um tratador de eventos, responsável por interpretar os sinais produzidos pela base de conhecimentos e repassar aos atuadores para que estes, efetivamente, produzam a ação deliberada, figura 3.

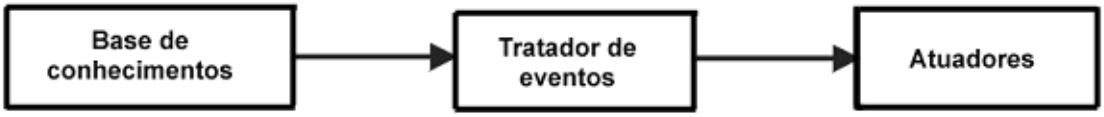

Figura 3: A estrutura interna de um robô.

Dado, por exemplo, que a regra *seguir sul* foi ativada o tratador de eventos captura esta ativação e aciona os atuadores do robô que são responsáveis pela concretização do movimento. Tudo o que acontece pode ser visualizado em uma interface gráfica bastante amigável. A configuração do ambiente, posições com sujeira e obstáculos, é realizada manualmente.

O determinismo do RAND pode ser verificado no algoritmo adotado para sua locomoção. Eis o algoritmo:

- Iniciar o robô na posição 0:0, i.e, linha 0, coluna 0.
- Se não houver obstáculos ▷ seguir em frente.
- Caso haja algum obstáculo ▷ vire 90º ▷ continue.
- Marque cada posição visitada.
- Caso haja sujeira na posição atual ▷ limpe-a.
- Caso esteja no centro do ambiente ▷ finalize.

O resultado da execução deste algoritmo encontra-se na figura 4.

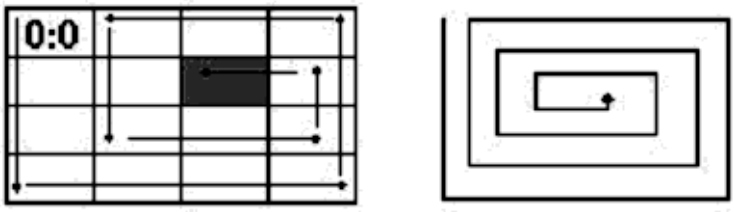

Figura 4: Algoritmo de locomoção de um robô.

Para que o RAND funcione sem conflitos as regras precisam seguir algumas prioridades de execução. Deste modo, as regras de movimentação *andar* e *virar* têm prioridade -2. A regra *determinar final* tem prioridade -1. A

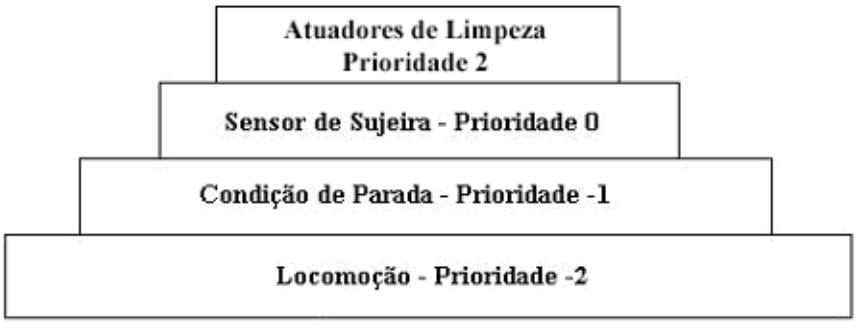

Figura 5: As prioridades de execução das regras.

regra *existe sujeira* tem prioridade 0. Nesta simulação, a regra relacionada aos atuadores de limpeza tem a maior prioridade de execução, como pode ser visto na figura 5.

Uma tela desta simulação pode ser encontrada na figura 6.

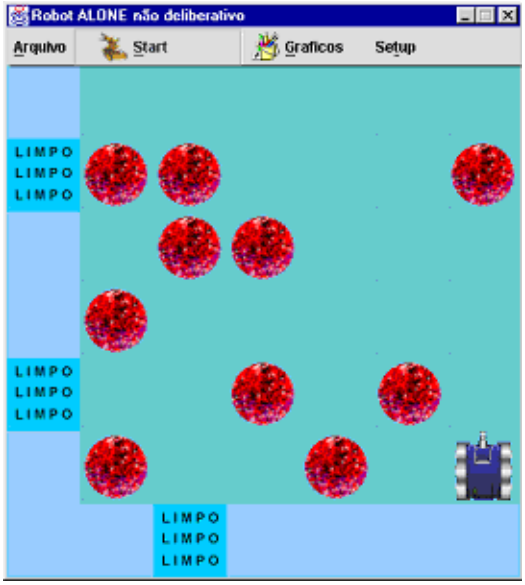

Figura 6: Tela de execução do RAND.

4.2 2ª simulação: Robôs não-conscientes (RNCs)

Quando se utiliza mais de um *agente robô* em seu ambiente vários problemas começam a aparecer. Comparativamente ao comportamento cerebral tal abordagem seria semelhante aos problemas que aparecem quando o cérebro opta por resolver um problema através de vários processos ao invés de resolver o mesmo apenas com um processo. O termo *robôs não-conscientes* denota que estes não realizam qualquer raciocínio lógico. A principal diferença dos RNCs para os RANDs é que aqueles realizam movimentos randômicos.

As regras foram modeladas da seguinte maneira:

1. Regra *sortear-movimentação*, sorteia uma posição a ser seguida.
2. Regra *obstáculo*, verifica se na posição ocupada pelo robô há sujeira.
3. Regra *andar*, dividida em 4 regras menores:

- Andar norte
- Andar sul
- Andar leste
- Andar oeste

4. Regra *algum limite*, verifica se há algum limite físico para o robô que o impeça de prosseguir.
5. Regra *determinar final*, verifica se algum limite foi atingido um certo número de vezes. Caso isso ocorra o robô pára.

Note-se que os RNCs só páram quando 'batem' um certo número de vezes, i.e, o mecanismo de parada é bastante rudimentar. Estes robôs apresentam vários problemas interessantes. Dentre esses problemas pode-se citar a infração à lei da física de que dois corpos não ocupam o mesmo lugar no espaço. Devido ao fato de não possuírem nenhum sensor para identificar a presença de outros robôs um robô pode bater com outro em uma determinada posição. Os RNCs apresentam apenas sensores de limites físicos, ou seja, limites do ambiente onde estão localizados. Assim, é possível, pela rudimentar condição de parada, que um RNC fique 'batendo cabeça' em algum limite, seguidas vezes, até tomar outro rumo de locomoção.

Nesta simulação existe uma espécie de ser onisciente que recebe todas as informações do ambiente e dos agentes nele presentes. Este ser comporta-se tal como um processo cerebral criado com o único objetivo de controlar os demais processos menores.

4.3 3ª simulação: Robôs semi-conscientes (RSCs)

Os *robôs* são dotados de semiconsciência na execução das tarefas. Comparativamente aos processos do cérebro seria quando este dotasse os seus processos de alguma consciência, ainda que, permitindo certos erros na resolução de um dado problema. O que torna os robôs ditos *semi-conscientes* é o fato destes não repetirem locais por onde já passaram, i.e, eles passam a ter uma espécie de memória de movimento. Se já fizeram certo percurso não o fazem novamente. Com esta medida os RSCs diminuem, em muito, o número de colisões com outros robôs. Entretanto, ainda estão propensos a cometê-las.

Colisões com outros robôs ainda acontecem devido à ausência de comunicação com seus pares no ambiente no sentido de todos saberem os locais já visitados. Caso cada robô fosse capaz de avisar aos companheiros que visitou um certo local obviamente seus companheiros não visitariam este local novamente. Suas bases de conhecimentos têm o seguinte formato:

1. Regra *configurar estado inicial*, prepara a 'consciência' do robô para o início da execução no sentido de zerar qualquer dado em sua memória, caso exista.
2. Regra *obstáculo*, verifica se na posição atual há sujeira, se houver ▷ limpar.
3. Regra *sortear* faz o sorteio da próxima posição a ser ocupada pelo robô. Esta regra torna o movimento dos RSCs randômico.
4. Regra *presente*, verifica se o robô já esteve presente num determinado local. Em caso afirmativo ▷ ativar regra *sortear*.
5. Regra *andar*, efetua o movimento de locomoção a partir de uma resposta da regra *sortear* conjugado com a regra *presente*.

Neste ponto, pode-se observar algo interessante:

> *A força de um sistema na busca de sujeira em todas as simulações está na quantidade de robôs e não na capacidade destes de elaborar planos, visto que, nenhuma simulação prevê tal capacidade. Deste modo, busca-se a resolução distribuída de problemas não a deliberação de um avançado sistema multiagentes.*

4.4 4ª simulação: Robôs conscientes (RCs)

O problema *dirty catch* é resolvido segundo o cérebro o resolveria utilizando uma arquitetura de vários processos em execução submetidos a um processo coordenador [Orwell, 1992]. Todos os processos submissos ao processo maior são capazes de se comunicar com este e, em contrapartida, este é capaz de se comunicar com todos os processos.

O termo robôs conscientes vem do fato destes conseguirem um certo dinamismo em sua atuação no ambiente. O algoritmo de locomoção de cada robô consiste em sortear uma entre quatro possibilidades de movimento. Logo, os RCs locomovem-se randomicamente. Devido ao seu sensor de sujeira as posições dotadas com focos desta têm maior probabilidade de escolha durante o movimento.

Ao realizar um movimento o robô avisa um agente central, que tem o controle de todas as operações, que o movimento foi realizado. Ao receber o aviso o agente centralizador avisa a todos os outros robôs do ambiente sobre tal ação de modo que estes não repitam uma ação já realizada. Assim sendo, uma mesma posição não será visitada mais de uma vez e, conseqüentemente, a verificação de sujeira em uma dada posição também ocorrerá apenas uma vez. A implementação da base de conhecimentos consiste nas seguintes regras:

1. Regra *obstáculo*, verifica se há sujeira na posição ocupada pelo robô.
2. Regra *sortear*, faz o sorteio da próxima posição do movimento.
3. Regra *andar*, dado o sorteio ▷ efetua a movimentação.
4. Regra *limpar verificações*, limpa as variáveis para novas verificações no ambiente.

Note-se que a inclusão de consciência nos robôs simplifica sua base de conhecimentos diminuindo o número de regras. Na verdade acontece uma expansão do poder das regras, i.e, estas poucas regras fazem o trabalho de muitas pequenas outras.

Os RCs páram sua execução quando não conseguem mais efetuar nenhum movimento válido, ou seja, estão em algum beco sem-saída ou já visitaram toda sua vizinhança. Observa-se, às vezes, que nem todas as regiões com sujeira foram limpas. Isto se deve ao fato de que o movimento destes robôs é randômico e que eles não retornam a locais já visitados. Desta forma é bem possível que um RC se feche em um canto sem mesmo ”imaginar” que existem muitos outros locais a serem investigados.

4.5 5ª simulação: Robôs conscientes distribuídos (RCDs)

Até este ponto, em todas as simulações, a comunicação entre os robôs, caso houvesse, era realizada via agente centralizador. Nesta simulação há uma mudança de paradigma. A figura do agente centralizador que tudo vê e tudo ouve, lembrando um pouco o *Big Brother* de George Orwell em sua obra 1984 [Orwell, 1992], deixa de existir. Cada robô passa a ser responsável por sua ação bem como por sua comunicação com os companheiros de jornada. Esta simulação trata da arquitetura distribuída do cérebro.

A estrutura da base de conhecimentos não mudou muito em relação aos RCs. Acompanhe:

1. Regra *configurar-estado-inicial*, prepara o robô para início de execução.
2. Regra *obstáculo*, verifica se a posição atual tem sujeira, em caso afirmativo ▷ limpar.
3. Regra *sortear* faz o sorteio da próxima posição a ser ocupada. Responsável pelo movimento randômico dos RCDs.
4. Regra *presente* verifica se o robô já esteve em certa posição.
5. Regra *andar* faz o movimento dado a resposta da regra *presente*. Se o robô já esteve em certa posição ▷ mudar a direção de movimento.

6. Regra *verificar possibilidades*, verifica uma espécie de memória das mensagens recebidas dos robôs companheiros de modo que este RCD não visite uma posição já visitada por outro.

A estrutura de comunicação e troca de mensagens é bastante simples. Utilizou-se a 'invocação remota de métodos', do inglês *Remote Invocation Method*, um recurso da linguagem JAVA que permite a comunicação entre módulos de programas escritos nesta linguagem em máquinas diferentes. Desta forma, é possível a execução de vários robôs em várias máquinas distintas em uma rede para resolver o problema proposto. A troca de mensagens é feita por TCP/IP [Sun©, 2002].

Não há sobreposição de robôs nesta simulação. Dado que o movimento dos RCDs é randômico e que estes não visitam locais já visitados por qualquer outro companheiro, ou por ele mesmo, nem sempre todos os focos de sujeira são limpos. Isto serve para demonstrar que o cérebro humano é uma caixinha de surpresas. Dado um problema tão simples quanto o *dirty catch* a busca de solução para o mesmo de forma artificial torna-o um problema não trivial.

5 Problemas encontrados durante as simulações

Obviamente, vários problemas aparecem quando há tentativas de se simular processos naturais com inteligência artificial. Neste trabalho pôde-se verificar alguns como:

RAND:

1. O robô não respeitava decisões da base de conhecimentos.
2. Saía dos limites do ambiente e não percorria este por inteiro.
3. Não havia um algoritmo eficiente de busca em matriz para a locomoção do robô.

O primeiro problema foi causado devido à velocidade em que as regras de conhecimentos são verificadas. Deste modo, havia momentos em que todo o processamento já havia sido feito e nenhuma modificação na tela havia sido demonstrada. Para que isto fosse resolvido implementou-se um tratador de eventos. Este tratador é responsável por receber uma inferência da base de conhecimentos e passá-la aos atuadores para que a tela seja adequadamente atualizada. Contudo, enquanto a tela não for atualizada o tratador não libera o processamento de uma nova inferência pela base de conhecimentos.

O segundo problema é um problema clássico de Computação Gráfica: *tratamento de colisão*. Desta forma, para haver o respeito aos limites físicos do ambiente e consequente determinação de uma colisão, cada robô passou a ser dotado de instintos "ação-reação" [Weiss, 1999]. Ao encontrar um limite o robô deve reagir a este limite de acordo com sua base de conhecimentos.

Quanto à inexistência de algoritmos eficientes implementou-se um algoritmo simples, embora não-eficiente.

RNCs:

1. Dificuldade em estabelecer uma condição de parada para os mesmos.
2. Dificuldade em fazer os robôs terem consciência própria e trabalhar em paralelo

O primeiro problema aparece devido ao fato de os RNCs não terem consciência nenhuma do ambiente em que estão inseridos. Como agem segundo o paradigma da ação-reação ou bateu-levou podem chegar a um obstáculo e ficar por ali *ad eternum*. Para contornar o problema estabeleceu-se que, depois de um certo número de tentativas repetidas de sorteio para mudança de movimentação e robô ativa uma regra de desligamento.

No segundo caso, como houve uma mudança de paradigma, a representação da individualidade de cada robô foi estabelecido em sua base de conhecimentos. Para isso, cada entidade robô passou a ter um processo próprio na máquina virtual java simulando o funcionamento de um processo do sistema oeracional.

RSCs:

1. Problemas com a representação eficiente de posições visitadas.

A solução foi criar-se uma espécie de memória para posições visitadas na própria base de conhecimentos do robô. Assim, quando um RSC passar por uma posição específica a sua base de conhecimentos, através dos atuadores do robô, recolhe informações sobre tal posição tratando de memorizá-las. Isto evita que uma posição visitada seja novamente inspecionada futuramente.

RCs:

Neste ponto, nenhum problema foi tolerável. Deste modo, qualquer problema detectado nas simulações anteriores foi solucionado. Eis alguns:

1. Superposição de robôs.
2. Módulos de comunicação ainda não funcionavam muito bem.

Para o primeiro problema passou-se a guardar, na memória de um agente centralizador, as informações sobre todas as posições visitadas por quaisquer robôs no ambiente. Conseqüentemente, se um RC tivesse a intenção de se movimentar para uma determinada posição, este solicitaria informações ao agente centralizador de modo a verificar a possibilidade do movimento.

Para o segundo problema, um melhoria da comunicação entre os robôs foi realizada desenvolvendo-se um módulo de comunicação em cada um destes. Deste modo todos passaram a ter capacidade de comunicar-se com o agente centralizador do ambiente.

RCDs:

Como todo o problema passou a ser simulado de forma distribuída verificou-se a necessidade de memorizar os endereços **IPs** [Sun©, 2002] dos robôs no ambiente. Cada robô passou a ter armazenado todos os endereços IPs de seus companheiros de jornada.

6 Trabalhos futuros

Algumas propostas de continuação consistem em:

- Implementar os agentes de forma que quando estiverem sendo executados em uma rede de computadores possam, sob a iminência de desligamento do terminal onde estes se encontram, tomar algumas decisões de sobrevivência. Algumas decisões poderiam ser: criação de uma cópia de segurança (*backup*) de suas bases de conhecimentos e clonagem de si mesmos em outra máquina com funcionamento estável na rede. Logo após, estes agentes devem inserir seus conhecimentos nos clones criados. Com isso, os agentes se tornariam independentes das máquinas onde estivessem sendo executados não tendo mais relações temporais com os terminais;
- Conectar o robô a uma plataforma de navegação autônoma para realização de testes práticos em ambiente real. Poderia criar-se um *Driver Agent*, i.e., um agente se comportaria como o motorista de um trator em um campo de pastagens, por exemplo;
- Agregar um módulo de processamento de imagens e visão computacional aos agentes para que os mesmos possam ter mais parâmetros para tomar decisões acertadas.

7 Conclusões

O estudo das técnicas de arquiteturas para Inteligência Artificial Distribuída [Ferber et all, 1991b] possibilita a ligação entre o estudo do funcionamento cerebral e a possibilidade de sua simulação utilizando-se a Inteligência Artificial. Mesmo que tais simulações sejam em 'domínios de brinquedo', como o apresentado pelos autores do artigo.

Embora a simulação de muitos problemas de I.A já seja possível, hoje em dia, é importante observar que ainda há muito que se descobrir na área. Falar de Inteligência Artificial já é muito difícil. Quando se resolve falar já se está restringindo o termo inteligência [Ferber et all, 1991a]. O que se sabe é que as máquinas não são capazes de pensar. Pensar como os humanos, diga-se de passagem. Entretanto, de certa forma, já são capazes de pensar como máquinas. Dado que várias máquinas tenham a capacidade de se comunicar e resolver problemas elas já não estão pensando de alguma forma? É muito difícil predizer até onde tudo poderá chegar, quem sabe nem estaremos aqui quando uma máquina realmente pensar tal como os seres humanos. Caso algo assim venha a se concretizar, algum dia, haverá outros problemas preocupantes: um possível embate ente criatura e criador, máquinas e humanos. O vencedor? Isto é uma verdadeira incógnita.

Referências

[Brooks, 1986] BROOKS, R. *A robust layered control system for a mobile robot*. In: IEEE Journal of Robotics and Automation, v. 2, n. 1, p. 14-23, march 1986.

[Ferber et all, 1991a] FERBER, J. e Gasser, L. *Intelligence artificielle distribuée*. In: XI International Workshop on Expert Systems & their Applications, Avignon, France, 1991, Cours n. 9.

[Ferber et all, 1991b] FERBER, Jacques. Jacopin, Eric. *The Framework of Eco-Problem Solving*. In: Demazeau, Yves; Muller, Jean-Pierre (Eds.). Decentralized AI 2, Amsterdam: North-Holland, 1991.

[Ferreira, 2002] FERREIRA, Aurélio Buarque de Holanda. *Dicionário da língua portuguesa*. Rio de Janeiro, Editora Nova Fronteira, 1993.

[Goleman, 1995] GOLEMAN, Daniel. *Inteligência Emocional: A teoria revolucionária que define o que é ser inteligente*. Rio de Janeiro, Editora Objetiva, 1995.

[Orwell, 1992] ORWELL, George. *1984*. Rio de Janeiro, Editora Record, 1992.

[Pleiad, 1992] PLEIAD. *Vers une taxinomie du vocabulaire pour les systèmes multi-agents*. In: Actes de la 1ère. Journée Systèmes Multi-Agents du PRC-GDR Intelligence Artificielle, Nancy, France, 1992.

[Rich, 1993] RICH, Elaine. Knight, Kevin. *Inteligência Artificial*. São Paulo, Makron Books, 1993.

[Rocha, 2002] ROCHA, Anderson de Rezende. *Opiniões, artigos e publicações*. 18 de novembro de 2002, http://www.comp.ufla.br/~undersun

[Russel e Norvig, 1995] RUSSEL, Stuart J. e Norvig, Peter. *Artificial Intelligence a modern aproach*. New Jersey, Prentice Hall, 1995.

[SandiaLabs, 2001] SANDIA LABORATORIES. *Jess, the Expert System Shell for the Java Platform*. 10 de agosto de 2001, http://herzberg.ca.sandia.gov/jess/

[Sichman e Alvares, 1997] SICHMAN, Jaime Simão. Alvares, Luis Otávio. *Introdução aos Sistemas Multiagentes*. Anais da Jornada de Atualização em Informática - JAI 97. São Paulo, 1997, editora da Sociedade Brasileira de Computação, SBC.

[Sun©, 2002] THE SUN MICROSYSTEMS. *The Java Tutorial*. 23 de dezembro de 2002, http://java.sun.com/docs/books/tutorial/

[Weiss, 1999] WEISS, Gerhard. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, 1999.