CENTRO UNIVERSITÁRIO FEEVALE

PAULA DENISE CARLOS RITTER

COORDENAÇÃO EM SISTEMAS MULTIAGENTES APLICADA A UMA EQUIPE PARA A ROBOCUP RESCUE

Novo Hamburgo, novembro de 2008.

PAULA DENISE CARLOS RITTER

COORDENAÇÃO EM SISTEMAS MULTIAGENTES APLICADA A UMA EQUIPE PARA ROBOCUP RESCUE

Centro Universitário Feevale
Instituto de Ciências Exatas e Tecnológicas
Curso de Ciência da Computação
Trabalho de Conclusão de Curso

Professor Orientador: Edvar Bergmann Araujo
Professor Co-Orientador: Dr. Paulo Roberto Ferreira Junior

Novo Hamburgo, novembro de 2008.

RESUMO

Os desastres são uma das mais significativas causas de perdas de vida nos países desenvolvidos e o trabalho das equipes de resgate em missões de busca e salvamento em situações de risco é um dos mais pertinentes desafios enfrentados pelo homem. A RoboCup Rescue é uma linha de investigação de âmbito social, apoiada há vários anos pela RoboCup, que visa possibilitar avanços no esforço humano motivando o desenvolvimento de agentes heterogêneos capazes de coordenar equipes de bombeiros, ambulâncias e policiais. Estes agentes atuam em ações contra os efeitos de uma calamidade com hostilidades crescentes, como por exemplo, um terremoto. Estas ações incluem apagar incêndios, salvar pessoas presas ou soterradas por escombros e liberar passagem por ruas bloqueadas, com o objetivo de minimizar danos humanos e materiais. Os métodos de coordenação de tarefas aplicados nas equipes da RoboCup Rescue são obras de adaptações e extensões de diversas pesquisas na área de Coordenação de Sistemas Multiagentes. Este trabalho apresenta um estudo sobre a aplicabilidade do algoritmo DSA (*Distributed Stochastic Algorithm*) na RoboCup Rescue. Este algoritmo foi mencionado na literatura como sendo inferior a novas abordagens quando aplicadas a problemas abstratos. Tais abordagens originaram algoritmos, denominados LA-DCOP e Swarm-GAP, que já foram experimentados na RoboCup Rescue e cujos resultados publicados não mostram diferenças significativas entre eles. Com isso, objetivou-se implementar um novo algoritmo baseado no DSA e comparar seus resultados com os resultados previamente obtidos pelos referidos algoritmos. Demonstra-se que o algoritmo DSA é tão eficiente para a RoboCup Rescue quanto seus sucessores mais sofisticados.

Palavras-chave: DSA, LA-DCOP, RoboCup Rescue, Sistemas Multiagentes, Swarm-GAP.

ABSTRACT

The disasters are one of the most significant causes of loss of life in developed countries and the work of rescue teams in search and rescue missions in situations of risk is one of the most relevant challenges faced by man. The RoboCup Rescue is a line of research into the social sector, supported for several years by RoboCup, which aims to enable advances in human effort challenging the development of heterogeneous agents capable of coordinating teams of firefighters, ambulances and police. These officers work in actions against the effects of a disaster with increasing hostilities, such as an earthquake. These actions include deleting fires, save people trapped or buried by rubble and release passing through streets blocked, tending minimize human and material damage. The methods of coordination of tasks applied in the RoboCup Rescue teams are works of adaptations and extensions of several searches in the area of Coordination of Multiagent Systems. This paper presents a study on the applicability of the DSA algorithm (*Distributed Stochastic Algorithm*) in RoboCup Rescue. This algorithm was mentioned in the literature as being inferior to new approaches when applied to abstract problems. Such approaches led algorithms, called LA-DCOP and Swarm-GAP, which have already been tested in RoboCup Rescue and whose published results show no significant differences between them. With that aimed to implement a new algorithm based on DSA and compare their results with those obtained in advance by these algorithms. Shows that the DSA algorithm is as efficient for the RoboCup Rescue as their more sophisticated successors.

Key words: DSA, LA - DCOP, RoboCup Rescue, Multiagent System, Swarm-GAP.

LISTA DE FIGURAS

Figura 1.1: Exemplo de interação de agente com o ambiente. 15
Figura 1.2: Visão da simulação no *AgentSheets* 18
Figura 1.3: Visão do agente na simulação 18
Figura 1.4: Visão do painel de configuração. 18
Figura 1.6: Interface principal do *NetLogo-3D* 20
Figura 1.7: Janela principal do SIMULA 21
Figura 1.8: Tela de definição dos agentes 21
Figura 1.9: Tela de definição das regras de comportamentos dos agentes. 22
Figura 1.10: Interface do *StarLogo* 23
Figura 2.1: Estrutura do simulador da RoboCup Rescue. 28
Figura 2.2: Tela de um visualizador 2D. 29
Figura 2.3: Tela de um visualizador 3D. 29
Figura 2.4: Rede de telecomunicação entre os agentes. 33
Figura 2.5: Hierarquia dos objetos no espaço desastre. 33
Figura 2.6: Todos os objetos ligados por sua propriedade topológica. 34
Figura 2.7. Visão do espaço desastre. 34
Figura 2.8: Objeto *Road* destacando dimensões de inicio e fim. 37
Figura 2.9: Objeto *building* e suas entradas 38
Figura 2.10: Visão do espaço desastre. 38
Figura 3.1: Implementação do algoritmo Swarm-GAP. 45
Figura 3.2: Implementação do algoritmo Swarm-GAP – *TryAllocate* 46
Figura 3.3: Implementação do algoritmo Swarm-GAP – *Adapt* 46
Figura 3.4: Implementação do algoritmo LA-DCOP. 49
Figura 3.5: Implementação do algoritmo LA-DCOP para *tokens* especiais 50
Figura 3.6: Esboço do DSA executado por todos agentes. 52

Figura 4.1: Mapa de Kobe com todos os agentes. 56
Figura 4.2: Gráfico da probabilidade *P* que levam o DSA a obter os melhores resultados. 57
Figura 4.3: Média do *Score* com barras de erro para o mapa Kobe 59

LISTA DE TABELAS

Tabela 2.1: Capacidade dos agentes da RoboCup Rescue 35
Tabela 2.2: Propriedades do objeto Humanóide..... 36
Tabela 2.3: Propriedades do objeto *Road*..... 36
Tabela 2.4: Propriedades do objeto *Building*..... 37
Tabela 4.1: Resultados de testes para identificar valores para a probabilidade *P*..... 57
Tabela 4.2: Resultado obtido nas simulações. 58

LISTA DE ABREVIATURAS E SIGLAS

CSPs	*Constraint Satisfaction Problems*
COPs	*Constraint Optimization Problems*
DSA	*Distributed Stochastic Algorithm*
E-GAP	*Generalized Assignment Problem*
IA	Inteligência Artificial
ID	Identificador
GIS	*Geographic Information System*
HP	*Health point*
LGCA	*Lattice Gas Cellular Automation*
LP	Linguagem de Programação
LA-DCOP	*Low-communication Aproximation DCOP*
SMA	Sistemas Multiagentes

SUMÁRIO

INTRODUÇÃO **10**

1 FUNDAMENTOS **13**
1.1 Liga da RoboCup Rescue 13
1.2 Agentes 15
1.3 Sistemas Multiagentes 16
1.4 Simulação Multiagente 17
1.5 Coordenação em SMA 24

2 SIMULADOR DA ROBOCUP RESCUE **27**
2.1 Estrutura 27
2.2 Inicialização 30
2.3 Agentes no ambiente da RoboCup Rescue 31
2.3.1 Humanóides 34
2.3.2 Road 36
2.3.3 Building 37
2.4 Dinâmica da Simulação 38

3 ALGORITMOS **41**
3.1 Swarm-GAP 41
3.2 LA-DCOP 46
3.3 DSA 50

4 EXPERIMENTOS E RESULTADOS **53**
4.1 Implementação do DSA para a Robocup Rescue 53
4.2 Definição dos experimentos 55
4.3 Resultados obtidos 56

CONCLUSÃO **61**

REFERÊNCIAS BIBLIOGRÁFICAS **62**

INTRODUÇÃO

O resgate em situações de desastre e risco é um dos mais relevantes problemas sociais da atualidade, pois as catástrofes são uma das mais significativas causas de perdas de vida nos países desenvolvidos. Segundo Kitano (1999), a Robótica e a Inteligência Artificial (IA) podem contribuir diretamente para a transformação deste quadro, aprimorando ferramentas usadas em casos de desastres. A idéia de utilizar simulações para aperfeiçoar a atuação de equipes em missões de busca e salvamento é apoiada há vários anos pela RoboCup[1].

A RoboCup Rescue[2] é uma linha de investigação proposta no âmbito da iniciativa internacional RoboCup. O desafio consiste no desenvolvimento de uma equipe de agentes heterogêneos capazes de limitar e minimizar os danos numa área urbana após um desastre de larga escala e salvar pessoas colocadas em situações de emergência. A relevância social é evidente.

No cenário da RoboCup Rescue o objetivo é coordenar equipes de bombeiros, ambulâncias e policiais em ações contra os efeitos de uma calamidade que ocorra em um espaço urbano, como por exemplo, um terremoto ou um incêndio. Estas ações incluem reduzir as conseqüências de um desastre, apagando incêndios, salvando pessoas soterradas por escombros e liberando passagem por ruas bloqueadas.

O simulador empregado pela RoboCup Rescue é baseado em um ambiente dinâmico onde existem situações como: propagação de incêndios, colapso de edifícios, movimento de civis, congestionamento de trânsito, bloqueios nas ruas e evolução do estado de saúde das vítimas. Este simulador é construído seguindo a metodologia dos Sistemas Multiagentes (SMA).

[1] http://www.robocup.org
[2] http://www.robocuprescue.org

Para desenvolver uma equipe de agentes de resgate, várias áreas de investigação devem ser integradas: a arquitetura dos agentes, suas capacidades básicas, seus mecanismos de decisão, protocolos de comunicação, coordenação de tarefas, integração do seu comportamento individual perante ordens superiores, etc. Os métodos de coordenação aplicados nas equipes da RoboCup Rescue são frutos de adaptações e extensões de diversas pesquisas na área de coordenação de SMA.

Para o sucesso da aplicação dos SMA no ambiente da RoboCup Rescue, diversos desafios devem ser destacados (FERREIRA JR. 2008):

- Dinamicidade do ambiente: novas tarefas podem ingressar no sistema a qualquer tempo e os requisitos delas podem mudar;
- Grande quantidade de agentes executando um grande número de tarefas;
- Visão local dos agentes: a visão pode ser incompleta sobre os objetivos do sistema; e
- Tomadas de decisões: precisam ser tomadas o mais rápido possível.

Outros trabalhos na área de SMA já aplicaram princípios de coordenação em equipes, como o campeonato mundial de futebol de robôs (RoboCup) onde ao invés de humanos jogando bola, são robôs reais ou simulados. Estudos nessa área permitem a avaliação de diversas técnicas referentes à coordenação em SMA e análise do comportamento do sistema numa partida de futebol de robôs simulados.

Utilizar a RoboCup Rescue como cenário para os SMA, além de colaborar na solução de um problema socialmente significativo, apresenta características diferentes da RoboCup tradicional, como a aplicação de agentes heterogêneos, o planejamento de longo alcance e a colaboração emergente entre equipes de agentes.

Neste trabalho propõe-se estudar o modelo E-GAP[3] para a alocação dinâmica de tarefas aos agentes em ambientes de larga escala, bem como estudar os algoritmos mais bem sucedidos na solução dos problemas modelados como E-GAPs. Serão analisados os algoritmos Swarm-GAP, LA-DCOP e DSA, os quais seguem linhas diferentes para lidar com o referido problema. Estes algoritmos são aqui aplicados na RoboCup Rescue utilizando time

[3] O GAP é proposto por Scerri et al. para capturar domínios dinâmicos e interdependência entre tarefas. O E-GAP é sugerido como uma forma de alocação de tarefas.

que os implementem para que os agentes determinem quais tarefas realizar em determinado momento.

O objetivo deste trabalho é avaliar o desempenho do algoritmo DSA para a tomada de decisão na RoboCup Rescue (nunca antes experimentado neste cenário) em comparação com os algoritmos LA-DCOP e Swarm-GAP (já experimentados). O DSA foi mencionado na literatura como sendo inferior ao LA-DCOP e este último foi mencionado como sendo superior ao Swarm-GAP quando aplicados a problemas abstratos (FERREIRA JR., BOFFO, BAZZAN, 2008). Experimentos publicados com a RoboCup Rescue não mostraram diferenças significativas entre LA-DCOP e Swarm-GAP. Com isso, objetivou-se implementar um novo algoritmo baseado no DSA e comparar seus resultados com os resultados previamente obtidos pelos referidos algoritmos. Demonstra-se neste trabalho que o algoritmo DSA é tão eficiente para a RoboCup Rescue quanto seus sucessores mais sofisticados.

Este trabalho está organizado em quatro capítulos. O primeiro capítulo apresenta a fundamentação teórica sobre a liga RoboCup Rescue, agentes inteligentes e os SMA. Seguindo neste contexto, os fundamentos de simulação multiagente são apresentados somados à caracterização e exemplificação de alguns ambientes projetados para este tipo de simulação. Algumas técnicas existentes de coordenação multiagente são destacadas no final do capítulo, enfatizando àquelas que se aplicam adequadamente à liga da RoboCup. O segundo capítulo aborda o sistema de simulação da RoboCup Rescue, destacando suas especificações, tais como estrutura, inicialização e o comportamento dos agentes dentro deste ambiente. As capacidades dos objetos e suas classes também são estudadas, incluindo a dinâmica da simulação e a forma como é calculado o desempenho dos agentes neste cenário. O terceiro capítulo apresenta em detalhes os algoritmos Swarm-GAP, LA-DCOP e DSA. O quarto capítulo descreve os experimentos realizados, incluindo sua configuração e os detalhes da aplicação de cada um dos algoritmos. Em seguida, são apontados os resultados dos experimentos realizados e uma análise de sua implicação no âmbito da simulação da Robocup Rescue. Por fim, são apresentadas as conclusões do trabalho.

1 FUNDAMENTOS

Este capítulo está organizado em cinco seções. A seção 1.1 apresenta um estudo sobre a liga da RoboCup Rescue e todas suas peculiaridades, incluindo os motivos que determinaram à criação da liga e os objetivos gerais. Em seqüência, a seção 1.2 aborda detalhadamente o conceito de agente inteligente, especificando suas características, como capacidades, comportamentos, organização e comunicação. Seguindo neste âmbito, é destacado na seção 1.3 o funcionamento dos SMA enfatizando a coordenação como ponto central. A seção 1.4 fundamenta os conceitos de simulação multiagente, apontando alguns ambientes projetados especificamente para modelagem baseada em agentes. Finalizando o capítulo, a seção 1.5 destaca as técnicas de coordenação multiagente existentes enfatizando àquelas que se enquadram no contexto do trabalho, que é a coordenação de uma equipe de agentes para a RoboCup Rescue.

1.1 Liga da RoboCup Rescue

Em 1995, precisamente no dia 17 de janeiro, um terremoto de intensidade 7.3 na escala Richter[4] devastou a cidade de Kobe, matando mais de 6.400 pessoas e minando a confiança dos japoneses nas medidas de precaução adotadas pelo país. Esta calamidade, que abalou a quinta maior cidade do Japão, foi responsável por desabrigar mais de um quinto da sua população. Cerca de 1,5 milhão de pessoas perderam suas casas devido aos abalos sísmicos e incêndios.

As adversidades do desastre danificaram inúmeras construções, deixando somente 20% em condições de uso. Os danos relacionados à infra-estrutura de Kobe ultrapassaram 100 bilhões de dólares e considerando o dano causado à propriedade privada, este número se eleva

[4] A escala de Richter quantifica a magnitude sísmica de um terremoto.

para 300 bilhões de dólares.

A descrição deste cenário, apresentada por Kitano (2000), é concluída pelo autor com a identificação de situações críticas que passaram a ocorrer com os primeiros abalos sísmicos provocados pelo terremoto:

- Com o colapso de um grande número de construções, as vias de circulação ficaram obstruídas e os sistemas de transporte público permaneceram interrompidos agravando excessivamente os problemas de congestionamento;

- Os serviços básicos de fornecimento de energia elétrica, água, gás, saneamento e coleta de lixo foram extremamente comprometidos;

- Com as infra-estruturas de comunicação danificadas, a comunicação correta e precisa de informações necessárias para a ação das equipes de socorro ficaram impedidas;

- Muitos focos de incêndio eram percebidos, porém existiam graves restrições quanto à disponibilidade de água para impedir que o fogo se espalhasse; e

- Os bloqueios nas vias de acesso (causados pelos escombros) dificultavam as equipes de resgate a localizar as vítimas soterradas. Apoio aéreo para captar informações sobre as vítimas e coordenar as equipes de apoio eram inviáveis devido ao barulho emitido pelos mesmos, que impediam que as equipes de terra ouvissem pedidos de ajuda das pessoas.

As conseqüências da catástrofe de Kobe, em parte pelo insucesso das operações de resgate, demonstram necessidade de um sistema que possa criar planos robustos e dinâmicos para auxiliar o esforço humano a lidar com situações deste âmbito com hostilidades crescentes (KITANO et al., 1999).

Para possibilitar avanços nesta área, foi proposta pela iniciativa internacional RoboCup, a *RoboCup Rescue Simulation League* (que segue sendo chamada apenas de RoboCup Rescue). Esta competição permite, através de um campo de teste comum e simulações, a comparação da qualidade dos planos gerados por diferentes abordagens para lidar com as adversidades de uma catástrofe.

O desafio da RoboCup Rescue consiste no desenvolvimento de uma equipe de agentes heterogêneos capazes de limitar e minimizar danos humanos e materiais numa área urbana após um desastre de larga escala.

1.2 Agentes

Um agente pode ser definido como uma entidade computacional que, situada em um ambiente, possui alguma percepção deste através de sensores, capacidade de raciocínio e ação, de forma autônoma e de maneira a desempenhar a função para a qual foi projetado.

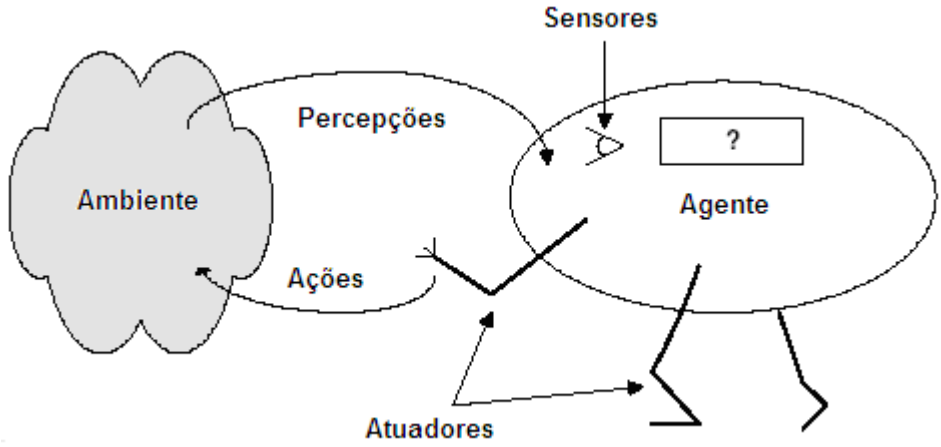

Figura 1.1: Exemplo de interação de agente com o ambiente.
Fonte: Adaptado de RUSSELL, 1995.

A figura 1.1 apresenta de forma genérica a arquitetura interna de um agente interagindo com seu ambiente através de sensores, tomadas de decisões e execução de ações por meio de seus atuadores. Esta interação ocorre através de possíveis ações que o agente pode tomar, capacitando-o a mudar o seu ambiente. Os agentes não podem tomar qualquer ação a qualquer momento. Para uma ação ser efetivada por um agente, ele precisa possuir alguma forma que o permita realizar tal ação. Todas as ações possuem pré-condições associadas (RUSSELL e NORVIG, 1995).

Segundo Wooldridge (2002) e Weiss (1999), as características envolvidas no conceito de agente, que definem suas capacidades são:

- **Racionalidade**: capacidade de agir com base na sua percepção e seu conhecimento, elevando a expectativa de seu desempenho em relação a um objetivo;
- **Reatividade:** capacidade de percepção de seu ambiente e de resposta, em tempo adequado, às mudanças percebidas;
- **Pró-atividade:** capacidade de manter objetivo em longo prazo e de tomar ações racionais para atingi-los; e

- **Habilidade social**: capacidade de comunicação **direta**, quando os agentes possuem conhecimento sobre os outros agentes, ou **indireta** quando a comunicação deve ocorrer por algum mecanismo compartilhado.

Na RoboCup Rescue, cada agente tem percepção limitada a sua volta. Agentes estão em contato pelo rádio, mas são limitados quanto ao número de mensagens que estes podem trocar. Portanto, visto que nenhum agente tem uma informação completa sobre o estado global, o domínio é, em geral, coletivamente parcialmente observável (NAIR et al. 2003). Isto significa que, mesmo se tendo o conhecimento de todos os agentes, ainda não há garantias de informação completa.

1.3 Sistemas Multiagentes

Segundo Kitano et al (1999), a RoboCup Rescue é um complexo de domínio multiagente que exige algumas questões importantes como: heterogeneidade, acesso à informação, planejamento de longo prazo e em tempo real.

Os SMA são sistemas compostos por vários agentes que apresentam um comportamento autônomo, mas que ao mesmo tempo interagem ou trabalham em equipe com os outros agentes a fim de desempenhar determinadas tarefas ou satisfazer um conjunto de objetivos (WOOLDRIDGE, 2000). Os SMA incluem pelo menos algumas das seguintes funcionalidades: coordenação, cooperação, competição e negociação.

Weiss (1999) classifica um SMA como **homogêneo**, para o caso em que os agentes do sistema tiverem as mesmas capacidades, ou **heterogêneo**, quando os agentes possuírem capacidades diferentes. A RoboCup Rescue se caracteriza por demandar um SMA heterogêneo uma vez que apresenta agentes com capacidades diferentes como exemplo bombeiros, ambulâncias e policiais.

A capacidade de comunicação entre os agentes permite a eles interagirem uns com os outros. Esta comunicação se faz com protocolos de interação social inspirado na sociedade de humanos, colônia de insetos sociais, entre outros. Com isso, o SMA passa a requerer cooperação, competição e coordenação.

Um SMA pode ser **cooperativo**, quando as ações tomadas pelos agentes são de forma a incrementar a utilidade global do sistema, concorrendo para a realização de um objetivo comum, ou **competitivo**, quando os agentes são projetados para contentar sua

satisfação pessoal, ou seja, possuem seus próprios objetivos. Na liga da RoboCup, pode-se definir um SMA cooperativo, pois todos agentes trabalham por um objetivo comum: minimizar danos de um desastre.

Nwana et al. (1996) afirmam que a cooperação entre agentes é fundamental, pois unicamente desta maneira pode-se ter uma combinação de esforços que permita aos agentes alcançarem objetivos comuns que não poderiam ser atingidos individualmente. Os autores completam, enfatizando a cooperação como principal razão para a existência de um SMA.

1.4 Simulação Multiagente

Segundo Drogoul (1992) a simulação multiagente consiste na reprodução artificial de um fenômeno natural através de um mundo artificial composto por agentes. A construção de modelos a partir de um agente permite que seguindo regras, se observe o comportamento de vários agentes interagindo entre si e com seu ambiente. Desta forma é possível observar de que forma que os padrões do sistema (como por exemplo, o fluxo de pedestres e bloqueios de trânsito), surgem independente dos comportamentos individuais. A partir daí, é possível adquirir um melhor entendimento sobre os comportamentos emergentes.

Dentre as várias técnicas de simulação destaca-se a modelagem baseada em agentes. Esse tipo de simulação é caracterizado pela existência de muitos agentes interagindo uns com os outros, com pouca ou nenhuma coordenação centralizada.

Existem vários ambientes de programação com abordagens distintas, projetados para simulação baseada em agentes. Dentre eles, destaca-se:

AgentSheets[5]**:** ambiente que possui características próprias e abordagem especialmente adequada para realização de atividades pedagógicas que utilizam simulação, pois sua LP é apropriada para a programação visual. Seu funcionamento baseia-se em agentes e sua atuação se desenvolve em tempo e espaço discretos.

A figura 1.2 mostra a visão de uma simulação no *AgentSheets* em execução com os agentes em movimento. A figura 1.3 exibe a imagem ampliada para que seja possível analisar o agente e seu comportamento durante a simulação.

[5] http://www.agentsheets.com

Figura 1.2: Visão da simulação no *AgentSheets*
Fonte: SANTANTANCHE A.; TEIXEIRA CAMILLO. C. A. 2000.

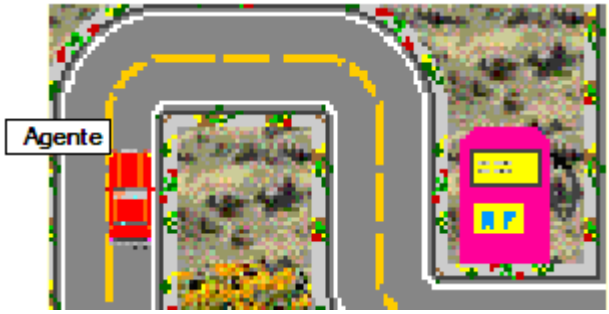

Figura 1.3: Visão do agente na simulação
Fonte: Adaptado de SANTANTANCHE A.; TEIXEIRA CAMILLO. C. A. 2000

A figura 1.4 identifica o painel de controle do comportamento do agente, no qual é possível configurar todas as suas ações.

Figura 1.4: Visão do painel de configuração.
Fonte: SANTANTANCHE A.; TEIXEIRA CAMILLO. C. A. 2000.

Klick & Play[6]**:** ambiente que faz uso de uma linguagem orientada a eventos. Apesar de ter sido desenvolvido para a criação de jogos, apresenta características para a modelagem multiagente. Esta plataforma foi muito utilizada no Brasil para fins educacionais, porém após

[6]http://www.clickteam.com

evoluções, atualizou nomes e versões, sendo a última especifica para jogos, chamada de *The Games Factory 2*. Esta linha de ferramentas usa uma forma de programação tabular, que apesar de não ser uma LP, pode ser usada para introduzir conceitos algorítmicos.

A Figura 1.5 mostra o editor de eventos da ferramenta, onde o projetista consegue visualizar todas as interações entre as entidades, o jogador e o jogo.

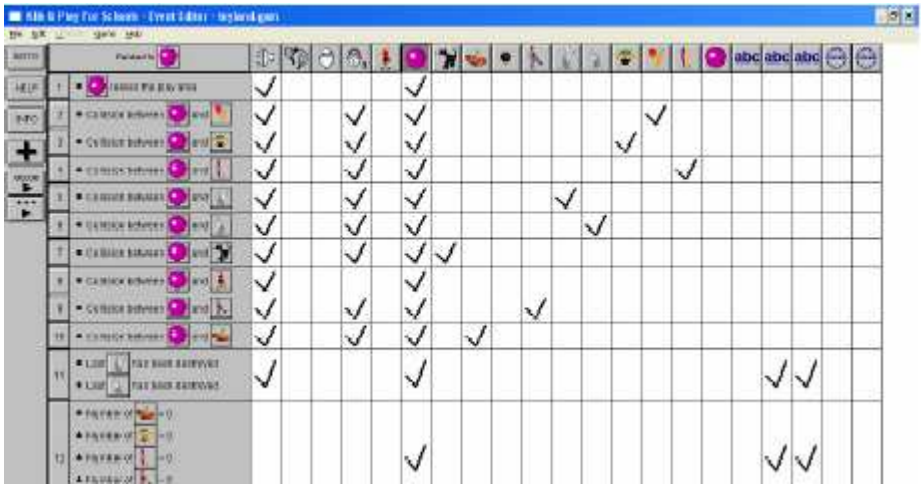

Figura 1.5: Editor de eventos do *Klick & Play* Fonte: CLICK TEAM - *Clik & Play* Official Site, 1996. Disponível em: < http://www.clickteam.com>

***NetLogo*[7]:** Ambiente que possui bibliotecas próprias, interfaces e recursos gráficos adequados para simular um SMA. Estas bibliotecas de exemplos podem ser usadas e modificadas. Sua LP é um dialeto *Logo* estendido para suportar agentes e os usuários desta ferramenta podem criar seus próprios modelos usando as facilidades e a documentação do *NetLogo*. Neste contexto, foi desenvolvido o *NetLogo-3D*, ferramenta computacional para simulação multiagente em três dimensões.

A figura 1.6 visualiza a interface principal do ambiente *NetLogo*, com uma instância de implementação em HPP[8]-3D.

[7] http://ccl.northwestern.edu/net
[8] HPP é um modelo da classe LGCA proposto para modelar partículas de gás

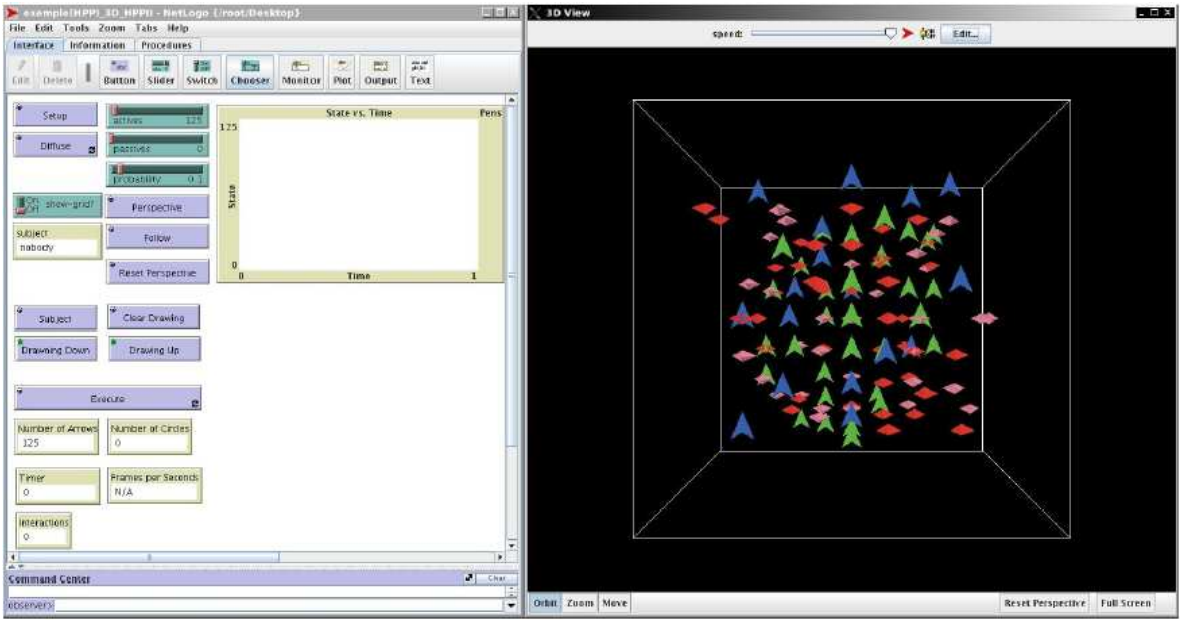

Figura 1.6: Interface principal do *NetLogo-3D*
Fonte: HPP - 3D em NetLogo 2007. Disponível em:
http://ccl.northwestern.edu/netlogo/models/community/Example-HPP-3D

SIMULA[9]**:** ambiente de desenvolvimento de aplicações baseadas em agentes reativos (Brooks, 1991). Foi desenvolvido na linguagem *Java*[10] e possui características próprias. SIMULA é uma plataforma ideal para o aprendizado de alunos em SMA, pois é considerado simples, não havendo a necessidade de conhecimentos profundos na área.

A figura 1.7 apresenta a interface do SIMULA, que possui quatro opções nas quais o usuário pode definir e visualizar a simulação da aplicação modelada. O menu **Arquivo** possui as funções padrão para a manipulação de arquivos. O menu **Configurações** possui as opções de definição da aplicação como: definição de agentes, de regras de comportamento, de critério de parada para a simulação, de variáveis, da distribuição dos agentes no ambiente e definição das dimensões do ambiente de execução da simulação. O menu **Executar** possui as opções de geração de código e de execução do processo de simulação. Menu **Ajuda** não está disponível.

[9] http://simula.sourceforge.net
[10] http://java.sun.com

Figura 1.7: Janela principal do SIMULA
Fonte: FROZZA, R (1999).

A figura 1.8 apresenta a tela onde é possível atribuir ao agente as suas definições como o nome do tipo do agente, o número de agentes deste tipo e sua área de percepção. A área de percepção é um valor limitado pelas dimensões do ambiente. Se o agente tiver uma área de percepção igual à zero, significa que não tem percepção alguma no ambiente. É possível editar uma figura e selecionar uma cor para o agente sendo assim possível identificar os diferentes tipos de agentes durante o processo de execução da simulação.

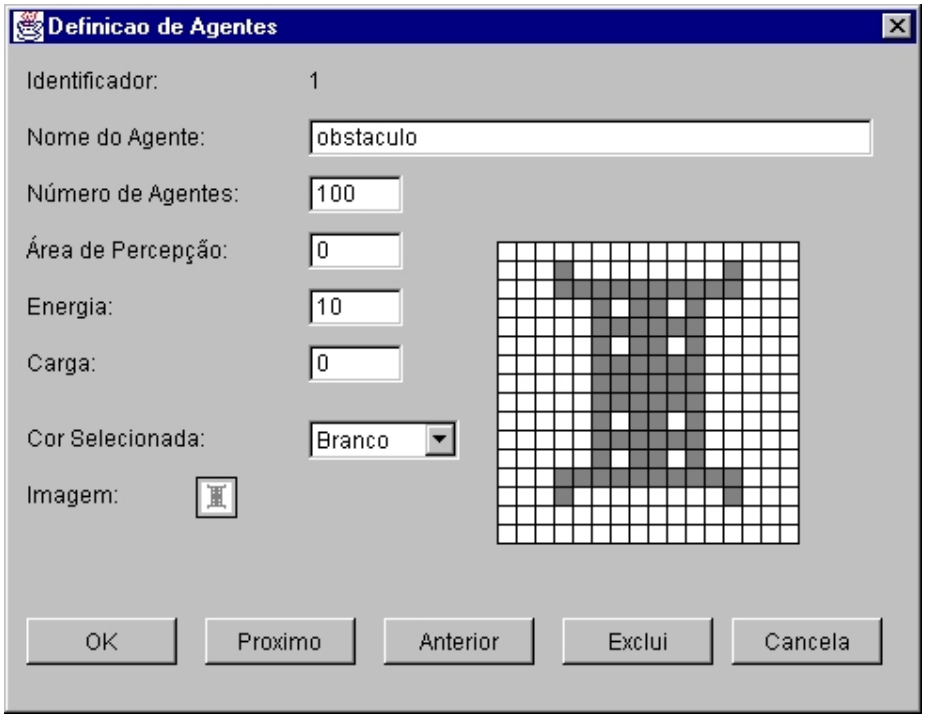

Figura 1.8: Tela de definição dos agentes
Fonte: FROZZA (1999).

A figura 1.9 mostra a definição dos comportamentos de cada tipo de agente. Esta definição de comportamentos é feita através de regras que determinarão as ações a serem

executadas pelos agentes e sob quais condições. A definição de comportamentos envolve a seleção do tipo de agente, a especificação da prioridade da regra e a criação da regra de comportamento.

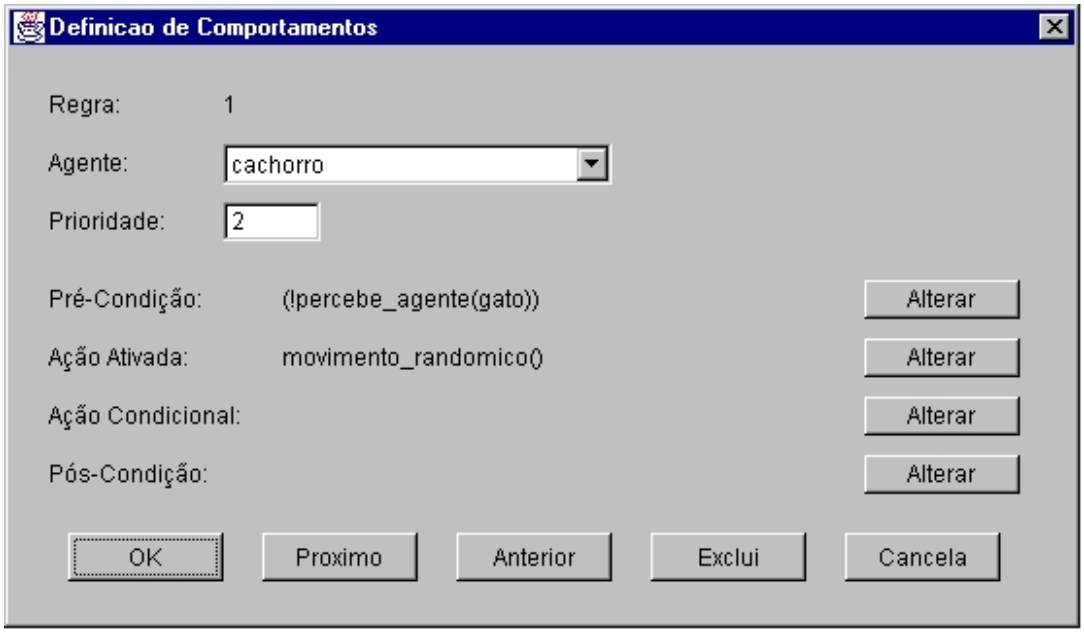

Figura 1.9: Tela de definição das regras de comportamentos dos agentes.
Fonte: FROZZA (1999).

StarLogo[11]**:** ferramenta de modelagem baseada em agentes e projetada para permitir a construção de seus próprios modelos de sistemas complexos e dinâmicos. Ela suporta um processo tangível de construção, análise e descrição de modelos que não exige habilidades avançadas de programação. Em seu ambiente, é possível impor regras simples para comportamentos individuais de agentes que se movem em um ambiente bidimensional. Exemplo: podem-se criar regras para um carro, descrevendo o quanto ele deve acelerar e a distância na qual ele deve encontrar obstáculos. Como o *StarLogo* faz uso de processos gráficos, ao assistir vários carros seguindo essas regras simultaneamente, é possível observar de que forma os padrões no sistema (como os congestionamentos de trânsito), surgem fora dos comportamentos individuais.

A figura 1.10 apresenta o ambiente *StarLogo* com um modelo de congestionamento de trânsito em ação.

[11] http://education.mit.edu/starlogo

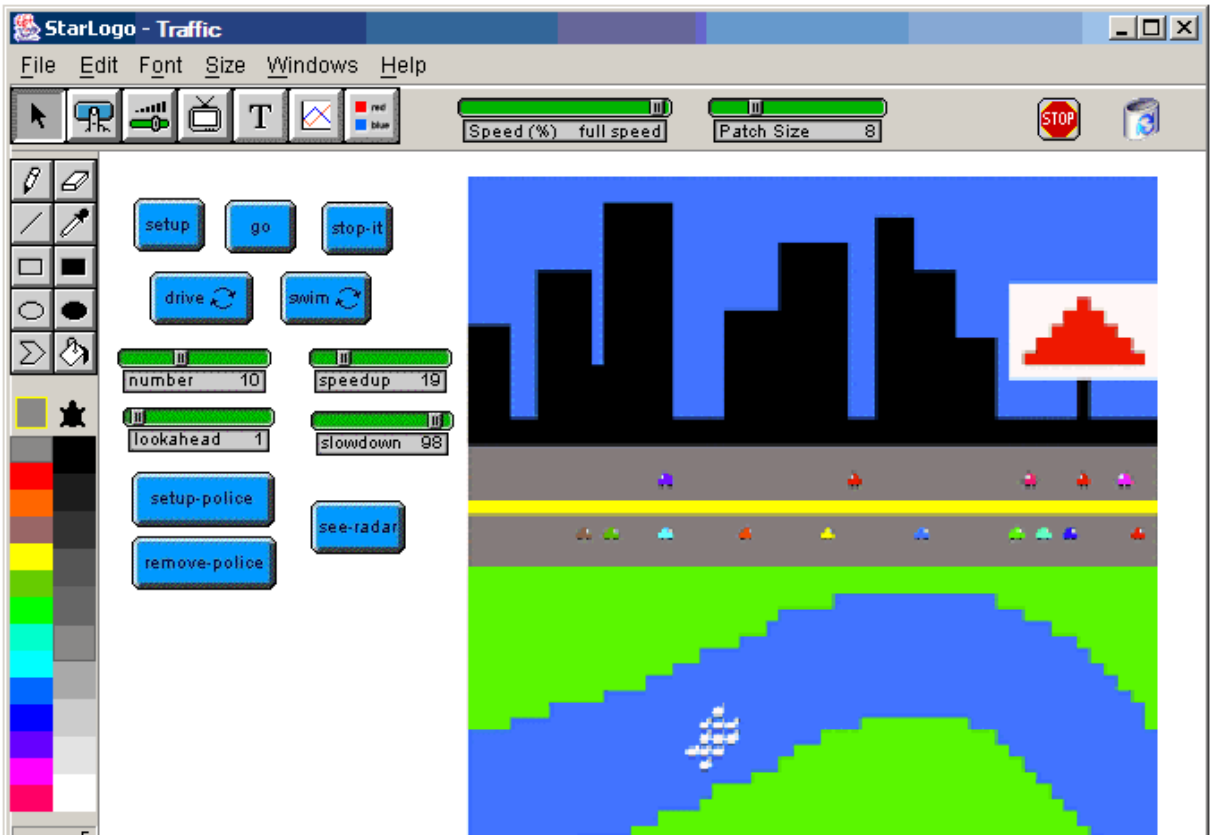

Figura 1.10: Interface do *StarLogo*
Fonte: STARLOGO. Disponível em: <http://education.mit.edu/starlogo>

Swarm[12]**:** ambiente baseado nas organizações de insetos. Apresenta uma plataforma com recursos que facilitam o desenvolvimento de aplicações que seguem o modelo baseado em agentes. Esta plataforma possui uma grande quantidade de recursos que podem proporcionar avanços no desenvolvimento de aplicações multiagentes.

Repast[13]**:** ambiente de código aberto que possui uma biblioteca baseada em agentes. Esta ferramenta busca dar suporte ao desenvolvimento de modelos de agentes flexíveis com ênfase em interações sociais. É possível construir simulações em várias plataformas como *Python*[14]e *Java*. O *Repast* possui vários mecanismos para processamento de topologias, incluindo um conjunto completo de ferramentas para armazenar e visualizar estados e ferramentas de integração automática com sistemas de informações de código aberto (MACAL, 2005).

Independente do processo idealizado por cada um dos ambientes, para definir uma sociedade de agentes é necessário dispor de elementos que descrevam essas sociedades, entre

[12] http://www.swarm.org
[13] http://www.python.org
[14] http://repast.sourceforge.net

eles: definição de agente, de comportamento, do ambiente e de variáveis. Além disso, os ambientes devem oferecer ferramentas conceituais como: gráficos, monitores, botões de acionamento, caixa de texto, controle de parâmetros e exportação de dados para área externa (para facilitar o acompanhamento do modelo e possibilitar a extração de dados comportamentais e análise).

Uma característica importante para a prática da aprendizagem cooperativa é a possibilidade do ambiente ser disponível para uso na *web*.

1.5 Coordenação em SMA

A coordenação é o ponto central no SMA, pois é o processo no qual os agentes se engajam para garantir que o conjunto deles que compõe o sistema irá atuar de uma maneira coerente (NWANA; LEE; JENNINGS, 1996). Quando os agentes trabalham para cumprir determinado objetivo, eles devem agir como uma unidade, coordenando suas ações, minimizando esforços redundantes, compartilhando recursos, dentre outros.

A coordenação em SMA pode ser definida como a forma harmoniosa e coerente de trabalhar em conjunto para atingir um objetivo comum. Uma coordenação efetiva é essencial para que os agentes alcancem seus objetivos em um SMA (EXCELENTE-TOLEDO; JENNINGS, 2005). Desta coordenação se espera o controle das várias formas de dependência que aparecem naturalmente quando os agentes possuem objetivos relacionados, quando habitam o mesmo ambiente ou quando compartilham recursos.

Jennings (1996) apresenta três razões principais que explicam a necessidade de coordenar ações em um SMA:

1. Existem dependências entre as tarefas dos agentes, as quais ocorrem quando a realização de uma tarefa de um determinado agente influencia na execução de uma tarefa executada por outro agente;

2. Agentes operam sobre restrições globais, as quais se tornam presentes quando os recursos estiverem escassos. A coordenação é então necessária para lidar com esta limitação, a qual não é respeitada quando os agentes apresentam um comportamento individual; e

3. Agentes individuais não possuem informação, recursos ou capacidade para resolver problemas, eles possuem competência para realizar tarefas diferentes ou possuem informação sensorial complementar devido à, por exemplo, localização distribuída no ambiente ou capacidade limitada de percepção.

Os métodos para a coordenação de SMA podem se dividir em dois grupos:

1. Aqueles que se preocupam com sistemas competitivos, compostos de agentes com interesse próprio (auto motivados) onde o esforço é adequar processos na resolução de conflitos, na argumentação e negociação; e

2. Aqueles que se aplicam aos sistemas cooperativos, que almejam alcançar melhor desempenho global, visando estratégias de coordenação na cooperação e considerando formas de resolver problemas comuns de maneira distribuída. Trabalham a fim de determinar times de agentes e realizar trabalho coletivo, o que pode ser comparado à noção de bem social presente em uma sociedade. Estes métodos se aplicam na RoboCup Rescue visando o trabalho coletivo.

Para Nwana et al. (1996), os mecanismos para coordenação multiagente podem ser classificados em quatro grupos:

1. **Coordenação por Estrutura Organizacional**: o uso de uma estrutura organizacional constitui um dos principais mecanismos que permitem a coordenação em SMA, pois os agentes devem possuir algum modelo que seja capaz de prever em certo nível o comportamento dos outros agentes. Os mecanismos de estrutura da organização são baseados em semelhança com organizações humanas, onde a coordenação passa a ser resultado do modelo de organização adotado, que atribui responsabilidades aos agentes. Através destas atribuições, a estrutura organizacional de um SMA estabelece linhas de controle, relações de autoridade e canais de comunicação, podendo incluir juntamente com os canais de comunicação, a natureza e quantidade de conhecimento que os agentes podem trocar.

2. **Coordenação por Criação de Contratos:** o uso de contratos é uma forma de coordenar gerando dinamicamente algumas relações nesta organização, auto-organizando o SMA (DURFEE; LESSER; CORKILL, 1989). Os agentes podem se gerenciar, partir um problema em subproblemas e contratar outros agentes para realizá-los e (ou) monitorar a qualidade da solução. Os agentes contratados podem, recursivamente, fazer novas contratações. O processo de contratação ocorre através de leilões que consideram a capacidade dos agentes para realizar as tarefas.

3. **Coordenação por Planejamento Multiagente:** os agentes se comunicam para criar um plano, de maneira centralizada ou distribuída, para evitar ações conflitantes ou inconsistentes. Esta forma de coordenar se baseia no planejamento conjunto entre os agentes para detalhar suas ações futuras e as interações necessárias para que o objetivo do sistema seja alcançado.

4. **Coordenação por Negociação:** mecanismos de coordenação baseados em negociação apresentam um processo explícito de comunicação, com protocolos definidos, onde os agentes trocam mensagens para atingir acordos sobre suas ações. Existem várias técnicas empregadas nessa negociação que são caracterizadas como: baseadas na teoria dos jogos; baseadas em planejamento; e inspiradas na interação entre humanos.

Com a correta coordenação, um SMA pode ser muito mais eficaz. Na RoboCup Rescue a técnica de coordenação empregada pelos agentes de resgate é o que, na prática, determina a eficiência do sistema.

A RoboCup Rescue permite que se implemente várias técnicas de coordenação pertencentes aos grupos mencionados acima, porém por ser um ambiente dinâmico e o tempo de deliberação entre os agentes ser restrito, as técnicas de coordenação tradicionais não se aplicam adequadamente. Isto trás novos desafios aos pesquisadores da área.

Existem três técnicas de coordenação multiagente empregáveis para o ambiente dinâmico da Robocup Rescue, que são:

1. **Swarm-GAP:** método proposto em base dos princípios de divisão trabalho presentes nas colônias de insetos sociais (FERREIRA JR.; BAZZAN, 2006). Esta técnica busca uma solução aproximada e distribuída do E-GAP e desenvolve uma proposta para a organização de SMA, onde a coordenação é associada à alocação e recolocação de tarefas aos agentes. Através da emergência de parte da estrutura organizacional do SMA, o mecanismo proposto pelos autores objetiva coordenar agentes para atuar em ambientes dinâmicos de larga escala. No Swarm-GAP os agentes decidem de forma autônoma e probabilística, em quais tarefas irão se engajar, levando em consideração o estímulo associado às tarefas e seu limite de resposta. Como não há negociação entre os agentes, o número de mensagens gerado é constante e igual ao número de agentes. Na seção 3.1 será detalhado mais especificamente este método.

2. **LA-DCOP:** o LA-DCOP (OKAMOTO, 2003; SCERRI et al., 2005) é um método para solução aproximada do E-GAP de forma distribuída e com baixo uso de comunicação. A técnica aborda a alocação distribuída de tarefas como um problema de decisão, utilizando um valor associado a cada tarefa como estimativa para a utilidade da decisão do agente de alocá-la ou não. Este método procura lidar com as necessidades de um grande número de agentes que devem operar em ambiente dinâmico. A auto-organização é a maneira que padrões de comportamento e estrutura emergem em um sistema, a partir da interação de suas partes mais simples. Na seção 3.2 será detalhado mais especificamente este método.

3. **DSA:** o DSA (WEIXIONG ZHANG et al.,2002) é um algoritmo distribuído probabilístico concebido para resolver problemas de satisfação e otimização de restrições de maneira distribuída (DSCPs e DCOPs). Estes problemas consistem na dificuldade de atribuição de valores a variáveis, respeitando restrições de valores entre elas. No DSA os agentes escolhem aleatoriamente um valor do domínio para atribuir a sua variável e uma série de iterações acontece e, em cada uma delas, um agente envia informações do seu estado atual para seus vizinhos. Se o agente mudar seu valor na etapa anterior, ele recebe a informação de seu novo estado e decide também probabilisticamente manter o seu valor atual ou mudar para um novo. O objetivo é o de mudar de valor eventualmente. Na seção 3.3 será abordado detalhadamente este algoritmo.

2 SIMULADOR DA ROBOCUP RESCUE

Este capítulo está organizado em quatro seções. A seção 2.1 apresenta o sistema de simulação da RoboCup Rescue, destacando a estrutura do ambiente. A seção 2.2 aborda todas as etapas e ciclos da inicialização do processo de simulação. Em seguida, a seção 2.3 exibe o comportamento dos agentes dentro do cenário da RoboCup Rescue, destacando as capacidades dos objetos e características das classes. Finalizando, a seção 2.4 exibe a dinâmica da simulação, informando como todo o processo da simulação acontece, indicando todas as etapas. Neste contexto, é apresentado o cálculo *Score* que é computado durante toda competição.

2.1 Estrutura

A RoboCup Rescue possui um simulador que deve ser visto como ambiente para SMA colaborativo, onde vários dos problemas que surgem podem ser solucionados com a correta coordenação.

A arquitetura do simulador baseia-se num ambiente dinâmico onde existem situações como: propagação de incêndios, colapso de edifícios, movimentação de civis, congestionamento de vias de acesso, falhas nos sistemas de distribuição e a evolução do estado de saúde das vítimas.

O sistema do simulador é em tempo real e distribuído, organizado através de um conjunto de vários módulos que são implementados de maneira diferente e comunicam-se através de um protocolo de rede (Figura 2.1). Cada módulo pode ser executado em diferentes computadores como um programa independente, de modo que a carga da simulação computacional pode ser distribuída para vários computadores. Os módulos do simulador são

os agentes, Simuladores Específicos (que serão chamados de subsimuladores daqui por diante), Sistema de Informação Geográfica (GIS), Visualizadores e o *Kernel*[15].

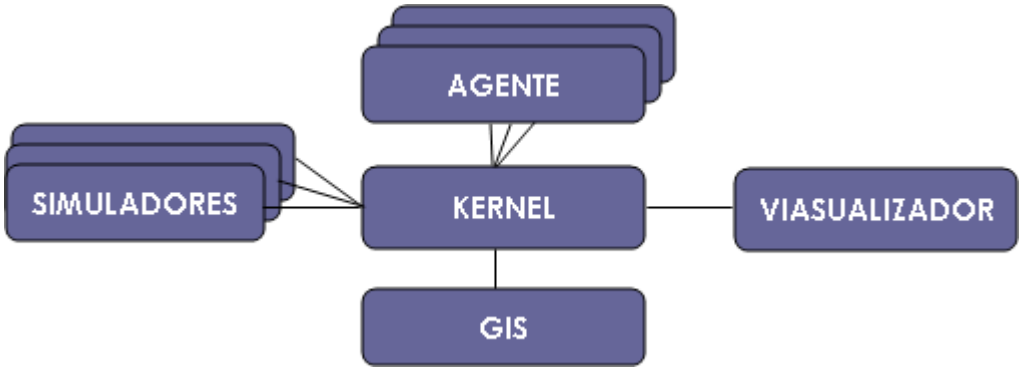

Figura 2.1: Estrutura do simulador da RoboCup Rescue.
Fonte: Adaptado de TAKAHASHI, 2000.

Cada módulo de agente controla um indivíduo autônomo. Indivíduos são entidades virtuais no mundo simulado e agem de forma independente, como, por exemplo, Civis, Equipes de Ambulância, Brigadas de Incêndio e Centros de Controle.

Os subsimuladores foram desenvolvidos para lidar com partes diferentes da simulação, como os diversos fenômenos da catástrofe, por exemplo, os terremotos, desmoronamento de edifícios, a propagação de incêndios, etc.

O módulo do GIS provê a localização geográfica dos objetos (construções, estradas, pontes, carros, pessoas, etc.) no mundo simulado e as condições iniciais do espaço desastre. Mesmo que esta informação se modifique com o progresso da simulação, ele atualiza seu modelo conforme resultados enviados pelo *kernel* e transmite ao visualizador.

O visualizador é responsável por exibir de forma que humanos possam compreender a informação do GIS que contém o estado da simulação. Atualmente existem visualizadores em 2D e 3D que exibem informações com variado nível de realismo. A figura 2.2 mostra a tela de um visualizador 2D exibindo um momento da simulação, com parte da cidade de Kobe como cenário. A figura 2.3 apresenta um momento de simulação semelhante, porém com efeitos de visualização 3D.

[15] No contexto deste trabalho, o *kernel* constitui o núcleo central da estrutura, o qual tem o controle completo sobre tudo que ocorre no sistema.

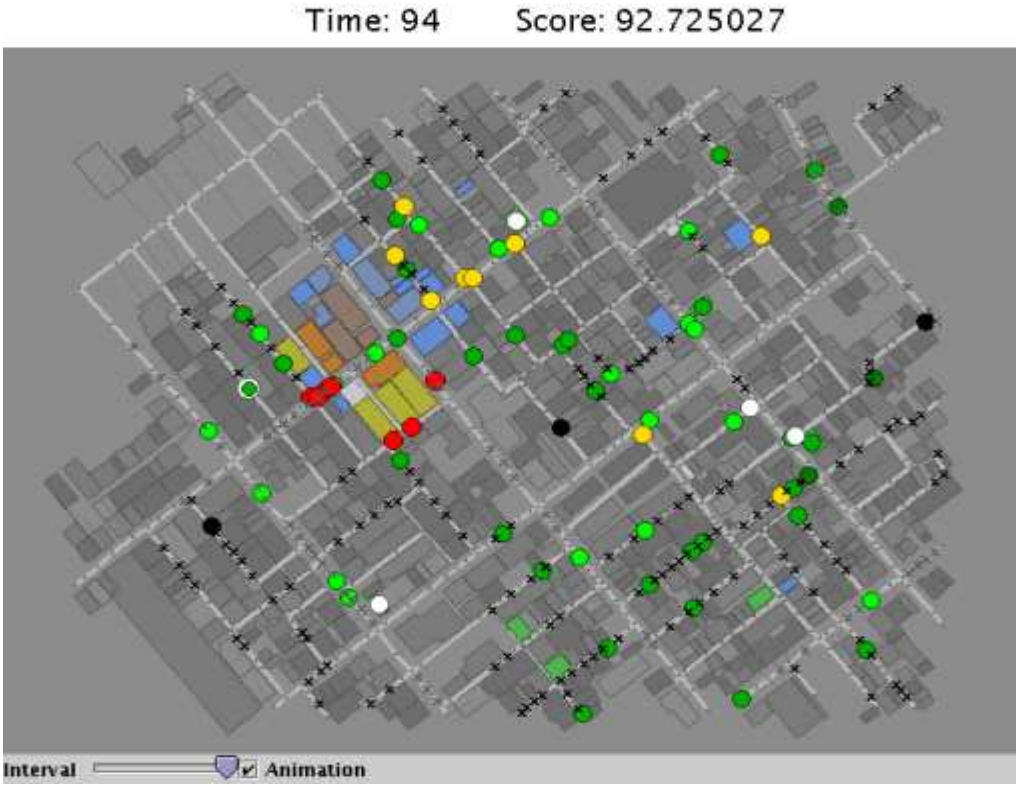

Figura 2.2: Tela de um visualizador 2D.
Fonte: TAKESHI MARIMOTO, 2002.

Figura 2.3: Tela de um visualizador 3D.
Fonte: ROBOCUP RESCUE - *Rescue Simulation Leagues*, 2006. Disponível em:
< http:// www.RoboCuprescue.org/simleagues.html >

O módulo do *kernel* controla todo o processo da simulação e gera a comunicação, ou seja, a troca informações com todos os outros módulos, sendo que nenhum módulo se comunica diretamente com outro módulo que não seja o *kernel*.

2.2 Inicialização

Antes do início da simulação, o *kernel* integra todos os módulos no simulador, enviando informação sensorial para cada um, possibilitando que o individuo possa perceber o mundo em um determinado momento. A maior parte desta informação é parcialmente visual, correspondendo ao que o agente pode enxergar dentro de seu raio de visão. Na integração e inicialização ocorrem os seguintes passos:

1. O *kernel* conecta-se ao GIS, o qual lhe fornece a condição inicial do espaço desastre.

2. Simuladores e o visualizador conectam no *kernel* o qual lhes envia a condição inicial.

3. Agentes se conectam com o *kernel* com seu tipo de agente. O *kernel* atribui cada agente para uma equipe de salvamento como, por exemplo, civis e envia a condição inicial de cada agente.

Quando todas as equipes de salvamento e civis no ambiente forem atribuídas aos agentes, o *kernel* finaliza a integração e a inicialização do simulador. Todos simuladores e os visualizadores têm de ser ligados ao *kernel* antes que toda atribuição de um agente seja encerrada.

A simulação está estruturada em ciclos, onde cada ciclo na simulação corresponde a um minuto. Por sua vez, estes ciclos são, em geral, compostos por seis etapas. A seguir procura-se explicar resumidamente cada ciclo.

No **primeiro ciclo** o subsimulador de colapsos simula a queda das construções e o subsimulador de incêndio começa a espalhar focos de incêndio.

No **segundo ciclo** o subsimulador de bloqueios simula os bloqueios das ruas baseado no resultado do subsimulador de colapso, e um subsimulador de *miscelâneas* começa simulando seres humanos que estão enterrados e feridos.

No **terceiro ciclo** os agentes começam a atuar e continua a execução das etapas seguintes (As etapas 1 e 2, descritas mais adiante, ainda não são executadas):

- **Etapa3**: O *kernel* envia comandos de ação dos agentes para todos os subsimuladores.
- **Etapa4**: Subsimuladores enviam estados atualizados do espaço desastre para o *kernel*.

- **Etapa5**: O *kernel* integra os estados recebidos dos subsimuladores, incrementa o passo da simulação e notifica os visualizadores.
- **Etapa6**: O *kernel* adianta a simulação do espaço desastre, ou seja, informa que segue para o próximo ciclo.

No **quarto ciclo** inicia a execução das etapas novamente:

- **Etapa1:** O *kernel* envia informações da visão individual de cada agente.
- **Etapa2**: Cada agente comanda uma ação para o *kernel* individualmente.
- **Etapa3**: O *kernel* envia comandos de ação dos agentes para todos os subsimuladores.
- **Etapa4**: Subsimuladores apresentam estados atualizados do espaço desastre para o *kernel*.
- **Etapa5**: O *kernel* integra os estados recebidos dos subsimuladores, incrementa o passo da simulação e notifica os visualizadores.
- **Etapa6**: O *kernel* adianta a simulação do espaço desastre, ou seja, informa que segue para o próximo ciclo.

A partir do quarto ciclo, todos os ciclos possuem as mesmas seis etapas identificadas acima.

2.3 Agentes no ambiente da RoboCup Rescue

O *kernel* modela o espaço desastre como uma coleção de objetos controlados pelos agentes, tais como edifícios, estradas e humanos. Cada objeto no simulador é constituído por um conjunto de parâmetros estáticos e dinâmicos. As classes de objetos no simulador, desconsiderando as classes base abstratas, têm propriedades como a sua posição e forma de identificá-la por um ID exclusivo. As classes de objetos são as seguintes:

- ***Building*** **(Construção):** uma construção pode ser desmoronada, obstruindo uma rua ou soterrando pessoas. É o único objeto que não é controlado por nenhum módulo de agente;

- ***Civilian*** **(Civis):** um grupo de civis que se encontra na área do desastre deve ser evacuado para um lugar seguro;

- ***Car*** **(Carro):** o carro é dirigido por um civil e possui as mesmas propriedades;

- ***FireBrigade*** **(Brigada de Incêndio) :** é responsável por apagar incêndios;

- ***FireStation*** **(Centro de bombeiros):** é o centro de controle para os bombeiros;

- ***AmbulanceTeam*** **(Equipe de ambulâncias):** responsável por socorrer vítimas soterradas e levar pessoas feridas a hospitais e centros de refugiados.

- ***AmbulanceCenter* (Centro de ambulâncias):** controla as ambulâncias, assim como o centro de bombeiros controla suas brigadas.

- ***PoliceForce* (Força policial):** sua função é liberar ruas bloqueadas.

- ***PoliceOffice* (Escritório policial):** o mesmo que o centro de bombeiros, para os policiais.

Um agente controlando um objeto *Civilian* é chamado de *Civilian Agent* (Agente Civil) e um agente controlando um objeto *AmbulanceTeam* é chamado de *Ambulance Team Agent* (Agente de equipe de Ambulâncias) e assim por diante.

Os objetos *AmbulanceTeam*, *FireBrigade*, e *PoliceForce* são agentes coletivamente chamados de *Platoon Agent* e a *AmbulanceCenter*, *FireStation* e *PoliceOffice* são também chamados *Center Agent*. O *Platoon* e o *CenterAgent* são chamados de *Rescue Agent* (Agente de Resgate). Os *Platoons* não podem se comunicar entre si quando são de classes diferentes. Ex.: ambulâncias não podem se comunicar com bombeiros.

Além de receber as informações sensoriais do *kernel*, os agentes estão em contato por rádio, podendo utilizar este mecanismo para notificar mudanças no ambiente, permitindo que os agentes mantenham modelos locais. Esta comunicação também deve ser feita pelo *kernel* (comandos para comunicar-se por rádio) e estão sujeitas a rígidas limitações.

A comunicação entre os agentes ocorre da seguinte forma: os times de agentes comunicam entre si e com os centros de comando. Conforme ilustrado na figura 2.4, os times de ambulância comunicam-se entre si e com o centro de ambulância; assim como, os times de brigada de incêndio comunicam-se entre si e com o centro de bombeiros; os times de força policial comunicam-se entre si e com seu centro, que no caso, o escritório policial. A comunicação também ocorre entre os centros.

Um agente tipo Centro de Comando pode enviar no máximo 2* N mensagens por ciclo de simulação (N é o número de agentes do mesmo tipo do centro de comando em questão). Para um agente de campo como bombeiros e policiais é permitido a troca de no máximo duas mensagens por ciclo. O tamanho da mensagem também é limitado a um número pré-determinado de *bytes*. A comunicação entre os agentes de campo de tipos diferentes é restrita, impossibilitando a comunicação direta, apenas indiretamente através dos centros de comando, constituindo uma hierarquia de comunicação.

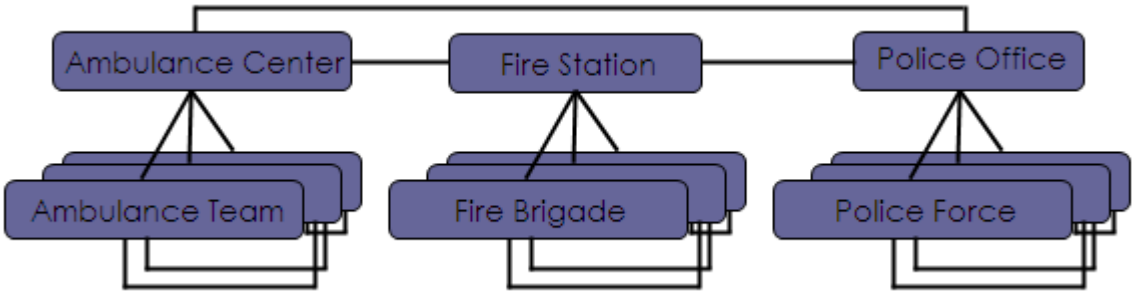

Figura 2.4: Rede de telecomunicação entre os agentes.
Fonte: Adaptado de TAKAHASHI, 2000.

A Figura 2.5 ilustra os objetos e sua organização hierárquica. Objetos imóveis estão localizados na posição designada pelas propriedades geográficas e todos os objetos estão ligados por suas propriedades topológicas (Figura 2.6).

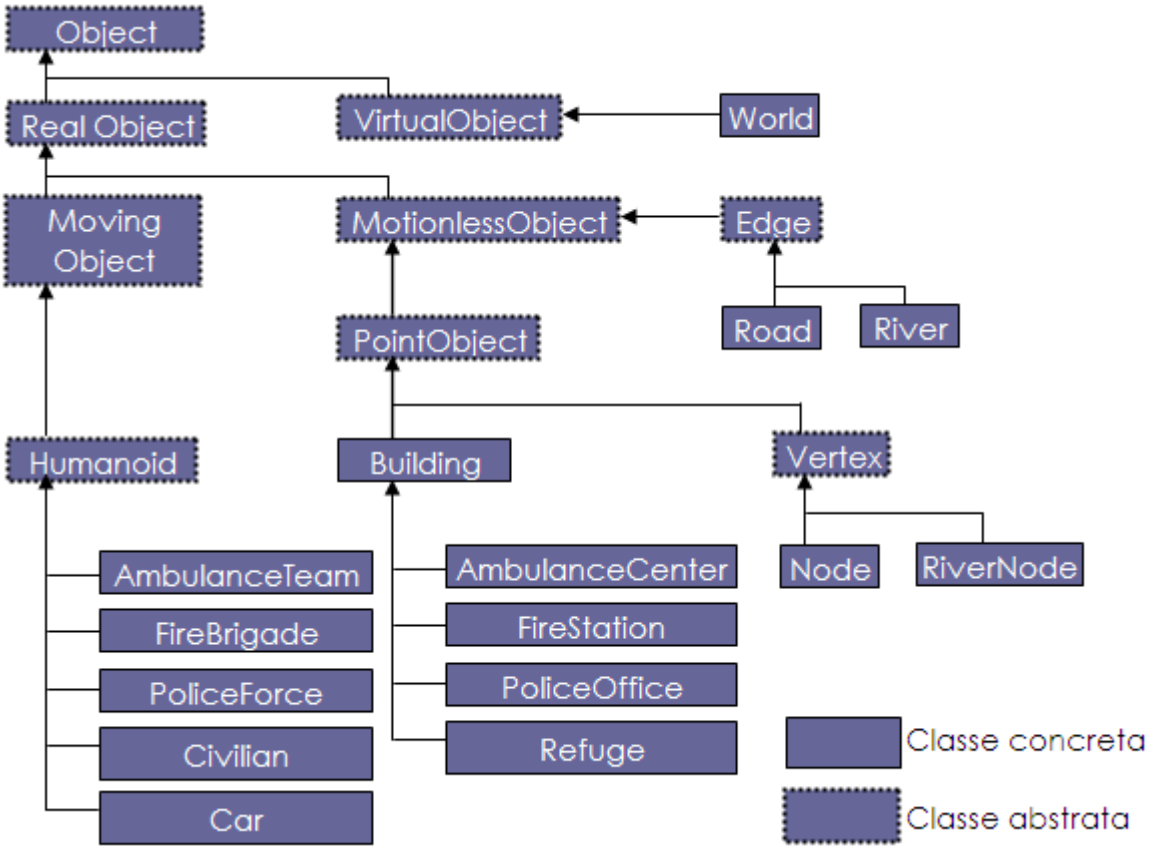

Figura 2.5: Hierarquia dos objetos no espaço desastre.
Fonte: Adaptado de TAKAHASHI, 2000.

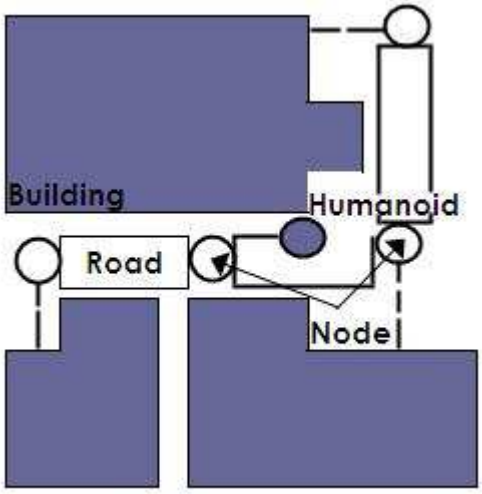

Figura 2.6: Todos os objetos ligados por sua propriedade topológica.
Fonte: Adaptado de TAKAHASHI, 2000.

Um objeto tipo *Node* modela a entrada de um prédio ou um final de uma rua. O objeto tipo *Humanoid* modela um civil ou um membro da equipe de salvamento, um objeto do tipo *Road* modela uma rua como uma área retangular (Figura 2.6).

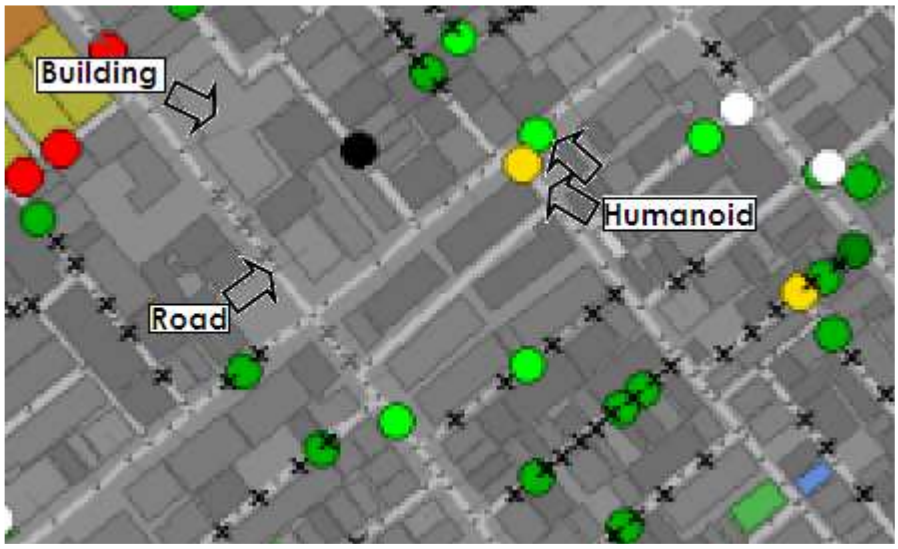

Figura 2.7. Visão do espaço desastre.
Fonte: Adaptado de TAKESHI MARIMOTO, 2002.

2.3.1 Humanóides

Na RoboCup Rescue os objetos como Civis, Equipe de Ambulâncias, Brigadas de Incêndio e Força Policial tomam decisões e realizam ações. Os agentes que os controlam precisam ser desenvolvidos e somente os Civis são programados como agentes controlados pelo *kernel*.

A ação de um objeto é processada repetindo as circunstâncias envolventes e decisão da ação em cada ciclo. Um agente reconhece o que está acontecendo na volta baseado na visão de informações recebidas do *kernel*. O agente decide uma ação e envia-a novamente para o *kernel*. Além disso, um agente comunica-se com outros agentes independentemente dos ciclos. Um agente tem diferentes capacidades de percepção e ação de acordo com seu tipo.

Tabela 2.1: Capacidade dos agentes da RoboCup Rescue

TIPO	CAPACIDADE
Civilian	Sentir, ouvir, dizer, mover.
Ambulance Team	Sentir, ouvir, dizer, comunicar-se por rádio, mover, salvar, carregar, descarregar.
Fire Brigade	Sentir, ouvir, dizer, comunicar-se por rádio, mover, extinguir.
Police Force	Sentir, ouvir, dizer, comunicar-se por rádio, mover, limpar.
Ambulance Center	Sentir, ouvir, dizer.
Fire Station	Sentir, ouvir, dizer.
Police Office	Sentir, ouvir, dizer.

Fonte: (Adaptado do Manual do Simulador da RoboCup Rescue, 2000).

Caracterização das capacidades:

- *Sense* (sentir) é caracterizada por informação sensorial;
- *Hear* (ouvir) se dá pela aquisição de informação por comunicação direta;
- *Move* (mover) é destacada como a forma de movimentação física do agente;
- *Say* (dizer) se dá pela troca de informação direta;
- *Tell* origina-se pela comunicação indireta, neste caso, via rádio.

Os comandos de ações contra o desastre são implementados como:

- *Extinguish* (extinguir);
- *Rescue* (salvar);
- *Load* (carregar);
- *Unload* (descarregar);
- *Clear* (limpar).

Os agentes, tais como Civis, Equipes de Ambulância, Brigadas de Incêndio e Centros de Ambulâncias são instâncias de humanóides. Os humanóides se movem pelas ruas e pelos prédios. Os Civis são os principais representantes desta categoria de humanóide, pois eles podem perder a vida, ficar soterrados e ser resgatados.

A tabela 2.2 apresenta as propriedades dos Humanóides. Pode-se analisar que o HP, o *Damage* e *Buriedness* são fundamentais para a pontuação da simulação, que será comentada no final do capítulo (seção 2.4).

Tabela 2.2: Propriedades do objeto Humanóide

PROPRIEDADE	COMENTÁRIO
Posição	Local que o objeto humanóide está. Quando o humanóide é carregado por uma ambulância, a posição passa a ser a ambulância.
Posição Extra	Distância da posição: distância do início da rua quando o humanóide está em uma, caso contrário, é zero.
HP (*Health Point*)	É a pontuação de saúde. O humanóide morre quando o HP se torna zero.
Damage (Danos)	É a pontuação dos danos. A pontuação dos danos diminui a HP em cada ciclo. Esta pontuação diminui à medida que o humanóide chega a um refúgio.
Buriedness (Enterrado)	O custo necessário para salvar o humanóide. Quando este for maior que zero, o humanóide não pode agir.

Fonte: (Adaptado do Manual do Simulador da RoboCup Rescue, 2000).

2.3.2 Road

A tabela 2.3 ilustra as principais definições do objeto *Road*, tratando especificamente das dimensões.

Tabela 2.3: Propriedades do objeto *Road*

PROPRIEDADE	COMENTÁRIO
Início e Fim	São os pontos que marcam o início e o fim da rua. Eles devem ser um *node* ou um *building.*
Comprimento, Largura	O comprimento do início até o fim da rua e sua largura.
Vias do Início ao Fim	A quantidade de vias de trafego para veículos trafegarem entre o inicio e fim da rua.
Bloqueios	Parte da largura da que está bloqueada, ou seja, na qual os carros não podem passar.
Custo de Conserto	O custo necessário para desobstruir o bloqueio.

Fonte: (Adaptado do Manual do Simulador da RoboCup Rescue, 2000).

A figura 2.8 mostra os pontos marcando o início e o fim da rua, bem como suas dimensões de largura e comprimento.

Figura 2.8: Objeto *Road* destacando dimensões de inicio e fim.
Fonte: Adaptado de TAKAHASHI, 2000.

2.3.3 Building

A tabela 2.4 apresenta as definições do objeto *Building*. Nesta tabela estão as características das construções enfatizando tanto área térrea como área construída. A propriedade *Fieryness* determina a taxa de quanto o prédio está queimando por um limite de tempo e a taxa do quanto ficou queimado neste tempo.

Tabela 2.4: Propriedades do objeto *Building*

PROPRIEDADE	COMENTÁRIO		
x,y	O x e y são as coordenadas do ponto que representa a localização do prédio.		
Entradas	Os nodos e ruas conectados		
Andares	Número de andares do prédio		
Área do terreno	Área do piso térreo		
Área total construída	Soma total da área de todos os andares		
Fieryness	Especifica o quanto o prédio está queimado		
	0	Não está queimado	
	1	Queimando	Taxa por tempo queimando
	2		0.00 ~ 0.33
	3		0.33 ~ 0.67
	5	Apagando	0.67 ~ 1.00
	6		Taxa do quanto ficou queimado
	7		0.0 ~ 0.2
			0.2 ~ 0.7
			0.7 ~ 1.0
Código *Building*	O código de um método da construção		
	Código	Método Construção	Taxa transmissão fogo
	0	de madeira	1.8
	1	estruturas de aço	1.8
	2	reforçada de concreto	1.0

Fonte: (Adaptado do Manual do Simulador da RoboCup Rescue, 2000).

A figura 2.9 destaca as entradas de um objeto *building* com os nodos conectados.

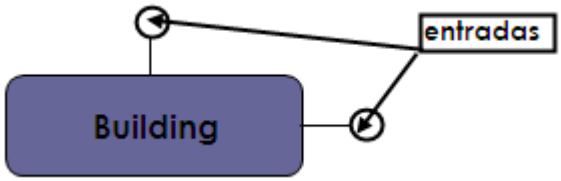

Figura 2.9: Objeto *building* e suas entradas
Fonte: Adaptado de TAKAHASHI, 2000.

2.4 Dinâmica da Simulação

No cenário da simulação da Robocup Rescue, de acordo como a dinâmica da simulação segue acontecendo, muitos pontos são destacados e modificados.

Conforme ilustrado na figura 2.10, os pontos verdes são humanóides Civis. Eles podem estar e andar tanto nas ruas quanto nos prédios. Durante a simulação os civis podem ficar soterrados e/ou feridos por escombros. O que determina o quanto o civil está soterrado é o *"buriednes"* (Tabela 2.2), que inicialmente é levado à zero, para que o civil possa ser removido. Enquanto as ambulâncias estão resgatando um civil, o *buriedness* dele diminui. Quando chega a zero, significa que não está mais soterrado, podendo então, a ambulância levá-lo para o refúgio.

Os pontos verdes vão ficando escuros à medida que o HP vai diminuindo (Tabela 2.2). Quando ele chegar à zero, significa que o civil está morto e seu ponto de verde, passa para preto. Se o civil chegar ao refúgio, seu HP pára de cair.

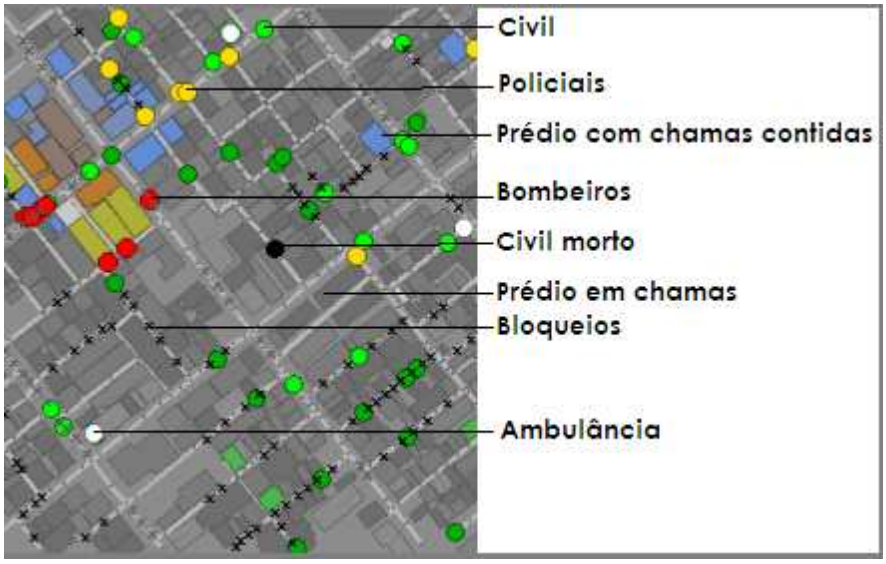

Figura 2.10: Visão do espaço desastre.
Fonte: Adaptado de TAKESHI MARIMOTO, 2002.

O refúgio protege os civis, pois é o único local que não pega fogo. É neste local que os bombeiros abastecem de água e os feridos são entregues pelas ambulâncias.

As ambulâncias são identificadas por pontos brancos. Elas andam aleatoriamente nas ruas e prédios a procura de civis feridos. Quando os encontram, carregam para um refúgio e seu ponto no simulador fica marcado com um circulo a volta, identificando que estão resgatando um civil. À medida que os civis vão sendo resgatado, o *Score* segue computando.

Durante a dinâmica, percebem-se os bombeiros pelos pontos vermelhos. Eles, assim como as ambulâncias, andam aleatoriamente nas ruas e prédios porém com intuito de apagar fogos dos prédios. De tempos em tempos, eles precisam abastecer a água, para isso seguem até os refúgios.

Os pontos amarelos no cenário são os policiais andando pelas ruas e prédios com o objetivo de desbloquear as ruas. De acordo com a intensidade do bloqueio, alguns ciclos são necessários. A cor da rua também varia neste caso, quanto mais intenso for o bloqueio da rua, mais escuro fica.

Durante a simulação, as ambulâncias, bombeiros e policiais correm o risco de ficar feridos se entrarem nos prédios que se encontram em chamas. Quanto mais escura estiver a cor do prédio, informa que mais intenso está o fogo. Quando ficar cinza escuro, significa que já foi totalmente destruído. Prédios azuis informam que tiveram as chamas contidas.

Abaixo, segue o calculo do *Score*, que ocorre durante a dinâmica da simulação e o significado de cada algarismo.

$$V = P + \frac{H}{H_{init}} * \sqrt{\frac{B}{B_{max}}}$$

onde:

- P = Número de civis vivos (HP > 0)
- H = Medida do estado de saúde de civis vivos, ou seja, vitalidade dos agentes (HP)
- H_{int} = Soma dos estados de saúde da população civil (HPs) no início da simulação

- *B* = Soma das Áreas de *Buildings* * fator
- *FIERYNESS* = (1, 4, 5) => fator = 0.66
- *FIERYNESS* = (2,6) => fator = 0.33
- *DEFAULT* => fator = 0
- B_{max} = Total de área construída (Soma das áreas).

3 ALGORITMOS

Este capítulo está organizado em três seções. A seção 3.1 apresenta o método Swarm-GAP, especificando sua origem, mecanismos, objetivos e equações de funcionamento. A seção 3.2 aborda o método LA-DCOP, bem como suas propostas, características, estruturas de alocação de tarefas e mecanismos de desempenho. Finalizando, a seção 3.3 exibe detalhadamente o algoritmo DSA, abordando sua metodologia e focalizando os problemas de satisfação e otimização de restrições de maneira distribuída.

3.1 Swarm-GAP

Com base nos princípios de divisão de trabalho presentes nas colônias de insetos sociais (THERAULAZ, 1998), Ferreira et al., (2005) propõem o algoritmo Swarm-GAP, capaz de lidar com o modelo E-GAP, que pode ser usado para formalizar alguns aspectos da atribuição dinâmica de tarefas aos agentes e proporcionar a coordenação de agentes aplicáveis na Robocup Rescue. Através da emergência de parte da estrutura organizacional de um SMA, o mecanismo simples de decisão estocástica proposto pelos autores objetiva coordenar agentes para atuar em ambientes dinâmicos de larga escala, com baixa comunicação e computação onde a coordenação é associada à alocação e re-alocação de tarefas a agentes.

Com base nos modelos teóricos e matemáticos que os descrevem, estes foram arquitetados com o objetivo de simular o funcionamento das colônias de insetos sociais e ajudar na compreensão de sua organização. Um dos modelos apresentados em Bonabeau (1999) pretende ser um modelo comum capaz de cobrir todos os aspectos que envolvem a divisão do trabalho nas colônias de insetos. Neste modelo, cada indivíduo tem um limiar de resposta a estímulos para realizar dada tarefa e cada indivíduo passa a executar esta tarefa quando o estímulo para executá-la ultrapassar seu limiar associado.

Sendo *s* a intensidade de um estímulo associado a uma atividade em particular, o qual *s* pode, por exemplo, simbolizar o número de encontros com outros indivíduos, ou qualquer outro fator quantitativo que possa ser sentido por um indivíduo. O limiar de resposta θ, expresso em unidades de intensidade de estímulo, é uma variável interna que determina a tendência de um indivíduo realizar a tarefa associada respondendo ao estímulo *s*.

A equação 1 apresenta uma função para a probabilidade de um indivíduo atender a resposta de um estímulo. Qualquer número inteiro *n* maior que 1 (um) pode ser usado com expoente de *s*, que determina o índice de crescimento de θ, porém Bonabeau (1999) utiliza em todos seus exemplos $n = 2$.

$$T_\theta(s) = \frac{s^2}{s^2 + \theta^2} \tag{1}$$

Se, desta maneira, o valor de θ for muito baixo, a probabilidade do indivíduo atender ao estímulo tende a 1 e se o valor de θ for muito alto, tal probabilidade irá tender a 0 e se $s=\theta$, a probabilidade é exatamente 1/2.

Para refletir o polimorfismo[16] e o polietismo[17] temporal dos insetos sociais, o limiar de resposta a um estímulo (valor de θ nas funções acima), pode variar para cada indivíduo. Indivíduos fisicamente diferentes ou de diferentes idades podem ter tendências diferentes de executar determinadas tarefas.

Qualquer indivíduo possui a probabilidade de deixar de executar uma tarefa no ato e pode retomar sua execução imediatamente se o estímulo que tiver para executar a tarefa que acabou de abandonar for superior ao seu limiar.

Conforme mostra o algoritmo 1, na figura 3.1, no Swarm-GAP os agentes decidem de forma autônoma e probabilística em quais tarefas irão se engajar considerando um estímulo associado às tarefas e seu limiar interno de resposta. Neste algoritmo os agentes se comunicam utilizando mensagens simples, reagindo a dois eventos: a percepção de tarefas e o recebimento de mensagens.

[16] Significa que classes diferentes podem ter comportamentos diferentes para a mesma operação.
[17] Divisão de trabalho reprodutivo.

Assim que um agente percebe uma tarefa, ele cria um objeto *taskAlloc* composto por estas tarefas (linhas 4 a 7). Seguindo esta linha, quando um agente recebe uma mensagem de outro agente (linha 9) ele também cria um objeto *taskAlloc*, mas desta vez com as tarefas que compõem a mensagem recebida. Logo que este procedimento é realizado, o agente passa a determinar qual destas tarefas quer alocar (linhas 11 a 23). Estas tarefas são priorizadas e o algoritmo principal aplica a função *TryAllocate*(), descrita no algoritmo 2, em cada tarefa. Com esta função, o agente decide qual tarefa alocar de acordo com sua tendência, restrito pela questão de que sua quantidade de recursos disponível tem de ser suficiente para alocá-la (linha 3). À medida que o agente determina alocar uma tarefa, este a atualiza para o estado *allocated* no objeto *taskAlloc* (linha 4). A quantidade de recursos que o agente possui é decrementada pela quantidade requerida pela tarefa alocada (linha 5). O algoritmo retorna *true* se a tarefa for alocada ou *false* se não for alocada respectivamente

Em seguida, os agentes utilizam a mesma função *TryAllocate*() no algoritmo 2, para tomar suas decisões sobre as tarefas que estes não sabem ainda se possui ou não o inter-relacionamentos AND (linha 18). Assim que o agente decide alocar uma tarefa, ele analisa se a tarefa escolhida possui inter-relação AND com outras tarefas (linha 19). Se este for o caso, as tarefas são analisadas na seqüencia (linhas 21 a 23) pela função *TryAllocate*(). Cada uma das tarefas inter-relacionadas que não é alocada nesse momento é incluída em um grupo de tarefas para serem compartilhadas com outros agentes (linha 23).

Os agentes podem se adaptar utilizando a função detalhada no algoritmo 3. Os agentes armazenam a melhor alocação realizada e esta é atualizada se a recompensa local obtida (w_{ij} na definição do E-GAP) pela nova alocação é maior que a recompensa alcançada pela melhor alocação armazenada (linhas 1 e 2). Se o agente está na última de $\mu\theta$ rodadas, este diminui seus limiares relacionados às tarefas alocadas na melhor alocação obtida até o momento (linhas 4 a 6), e aumenta seus limiares para as demais tarefas (linhas 7 a 9). A melhor alocação armazenada é reiniciada neste momento (linha 10).

Se os agentes ainda possuírem tarefas não alocadas neste último estágio do algoritmo principal, uma mensagem é então enviada para um dos agentes vizinhos aleatoriamente selecionados (linhas 27 a 33). Este agente não pode ter recebido nenhuma mensagem na rodada atual, para isso este é marcado como visitado (linha 30). O tamanho destas mensagens varia de acordo com seu número de tarefas.

O Swarm-GAP não lida com a resolução distribuída de conflitos na percepção das tarefas. Considera-se que a tarefa pode ser identificada como alocada ou simplesmente que as tarefas não são percebidas simultaneamente pelos agentes. Este tipo de resolução de conflito é um problema complexo e foge do escopo deste trabalho. Os demais algoritmos discutidos neste capítulo também assumem estas premissas.

Algorithm 1 Swarm-GAP($agentId$)

```
1:  loop
2:    ev ← waitEvent()
3:    if ev = task perception then
4:      J ← set of new unallocated tasks
5:      taskAlloc ← newTaskAlloc()
6:      for all t ∈ J do
7:        taskAlloc.addTask(t)
8:    else
9:      taskAlloc ← newTaskAlloc(receiveMessage())
10:
11:   τ_n ← taskAlloc.ANDTasks.getTasks()
12:   for all t ∈ τ_n do
13:     if TryAllocate(taskAlloc, t) then
14:       taskAlloc.ANDTasks.excludeTask(t)
15:
16:   τ_a ← taskAlloc.getNoANDTasks()
17:   for all t ∈ τ_a do
18:     if TryAllocate(taskAlloc, t) then
19:       if taskAlloc.isMemberOfAnyANDSet(t) then
20:         τ_g ← taskAlloc.getANDSetTasks(t)
21:         for all t' ∈ τ_g / t' ≠ t do
22:           if !TryAllocate(taskAlloc, t') then
23:             taskAlloc.ANDTasks.includeTask(t')
24:
25:   Adapt(taskAlloc)
26:
27:   τ ← taskAlloc.getAvaliableTasks()
28:   if |τ| > 0 then
29:     message ← newMessage(taskAlloc)
30:     message.visitedAgent(agentId)
31:     ι ← message.avaiableAgents()
32:     i ← rand(ι)
33:     sendMessage(i)
```

Figura 3.1: Implementação do algoritmo Swarm-GAP.
Fonte: PAULO ROBERTO FERREIRA JR. (2008)

Algorithm 2 TryAllocate($taskAlloc, t$)

```
1: if taskAlloc.avaliable(t) then
2:     r ← avaiableResources()
3:     if roulette() < T_{θ_t(s)} and r ≥ c_t then
4:         taskAlloc.allocateTask(t)
5:         r ← r − c_t
6:         return true
7: return false
```

Figura 3.2: Implementação do algoritmo Swarm-GAP – *TryAllocate*
Fonte: PAULO ROBERTO FERREIRA JR. (2008).

Algorithm 3 Adapt($taskAlloc$)

```
1: if taskAlloc.localReward()> bestAlloc.localReward() then
2:     bestAlloc ← taskAlloc
3: if last of μ_θ cycles then
4:     τ_a ← bestAlloc.getAllocatedTasks()
5:     for all t ∈ τ_a do
6:         θ_t ← θ_t − ξ
7:     τ_u ← bestAlloc.getAvaliableTasks()
8:     for all t ∈ τ_u do
9:         θ_t ← θ_t + ρ
10:    bestAlloc ← NULL
```

Figura 3.3: Implementação do algoritmo Swarm-GAP – *Adapt*
Fonte: PAULO ROBERTO FERREIRA JR. (2008)

3.2 LA-DCOP

SCERRI et al. (2005) propõem o LA–DCOP como um algoritmo de atribuição de tarefa com baixos requisitos de comunicação, para resolver problemas de alocação de tarefas especificamente em ambientes dinâmicos de larga escala, também aplicável no ambiente da Robocup Rescue. Este método, apresentado formalmente no algoritmo 4, foi especificamente desenvolvido para lidar com características especiais do E-GAP, abordando a alocação distribuída de tarefas como um problema de decisão e utilizando um valor associado a cada tarefa como estimativa para a utilidade da decisão do agente de alocá-la ou não.

Neste algoritmo, os agentes podem perceber tarefas no ambiente ou receber *tokens* de outros agentes, os quais contêm tarefas. Os agentes então decidem ou não alocar essas tarefas baseados em um limiar global, tentando maximizar o uso de seus recursos. Após decidir quais tarefas alocar, os agentes enviam *tokens* com tarefas não alocadas para outros agentes. O

limiar representa a competência dos agentes para alocar cada tarefa e pode ser fixo ou dinamicamente calculado.

Para identificar tarefas, no algoritmo 4, são utilizados *tokens*, os quais são criados quando um agente percebe uma nova tarefa (linhas 6 à 8) e a decisão que persiste é entre manter este *token*, aceitando a tarefa, ou passá-lo para outro agente (linhas 10 a 31). Mais de um *token* pode estar em posse de um agente ao mesmo tempo, logo esta decisão ocorre para cada *token* e um registro de todos os agentes que já tiveram o *token* é sustentado para que seja evitado de passá-lo a agentes que já o rejeitaram anteriormente. Com a intenção de respeitar as restrições do E-GAP, um agente deve ter recursos suficientes para todos os *tokens* que decidir manter (linha 17). A decisão quanto a este critério de capacidade suficiente de recurso para as tarefas aceitas é baseada puramente em informação local. A função *maxCap*() (linha 18) determina o subconjunto de *tokens* retidos que maximiza a recompensa local do agente. Este problema de otimização é equivalente ao problema da Mochila Binário[18], que é provado ser NP-completo (BAASE; GELDER, 2000).

Decidir aceitar um *token* ou não, considerando a utilidade para os outros agentes é a decisão que realmente influencia o desempenho global (GOLDMAN, 2004). Para este propósito, no LA-DCOP um *threshold*[19] é atribuído a cada *token* no momento de sua criação (linha 8), que representa a competência mínima que o agente deve possuir para aceitar o *token*. Sem superar essa competência mínima, o agente deve passar o *token*.

Segundo Okamoto (2003), a idéia central do método é defender que quando *thresholds* adequados são aproveitados, a solução resultante será satisfatória. Quanto ao problema de como calcular este *threshold*, o trabalho original de Okamoto (2003) apresenta que calcular *thresholds* ótimos, capazes de maximizar o desempenho do sistema, baseando-se em informação exata sobre as tarefas e os outros agentes, não é possível em sistemas de larga escala (OKAMOTO, 2003). Resultados mais gerais apontam que cálculos distribuídos desta natureza são extremamente complexos (GOLDMAN, 2004).

Okamoto (2003) afirma que em cenários dinâmicos, um *threshold* ótimo calculado pode tornar-se obsoleto rapidamente, promovendo a necessidade de calculá-lo sempre que

[18] O problema da mochila binário é NPcompleto, ou NP-difícil. Significa que possui ordem de complexidade exponencial. O esforço computacional para sua resolução cresce exponencialmente com o tamanho do problema (dado pelo número de objetos na mochila).
[19] A palavra *threshold* pode ser traduzida como “limiar”. Por se tratar do método LA-DCOP, será mantido o uso do original em inglês.

houver uma alteração de estado. Segundo o autor, a solução apropriada é determinar o *threshold* com base em informação das distribuições de probabilidade das competências dos agentes e das tarefas, porque as informações desta natureza geralmente estão disponíveis.

Okamoto (2003) sugere que, no instante em que o *token* é passado de um agente para outro, o *threshold* seja subtraído em alguma taxa, como outra forma de abranger *thresholds* apropriados, configurando assim o uso de *thresholds* dinâmicos que, para efeitos práticos, evita a postergação incerta de tarefas quando há falta de agentes com competência suficiente.

Segundo Okamoto (2003), o algoritmo LA-DCOP não define nenhuma maneira de decidir para qual agente um *token* deve ser passado quando for declinado. Neste momento pode ser utilizado conhecimento sobre quais agentes seriam mais capazes.

Algoritmo 4 LA-DCOP

```
1:  V ← ∅
2:  PV ← ∅
3:  laço
4:     msg ← recebeMsg()
5:     se msg é um token então
6:        token ← msg
7:        se token.threshold = ∅ então
8:           token.threshold ← calcThreshold(token)
9:        fim se
10:       se token.threshold < competencia(token.tarefa) então
11:          se token.potencial então
12:             PV ← PV ∪ token
13:             enviaMsg(token.dono, "retido")
14:          senão
15:             V ← V ∪ token
16:          fim se
17:          se ∑_{v∈V} v.custo ≥ recursoDisponivel() então
18:             liberados ← V − maxCap(V)
19:             para todo v ∈ liberados faça
20:                se v.potencial então
21:                   enviaMsg(v.dono, "liberar")
22:                   enviaToken(v)
23:                senão
24:                   enviaToken(v)
25:                   V ← V − v
26:                fim se
27:             fim para
28:          fim se
29:       senão
30:          enviaToken(token)
31:       fim se
32:    senão se msg é do tipo "lock v a*" então
33:       se v ∈ PV então
34:          PV ← PV − v
35:          V ← V ∪ v
36:       senão
37:          ∀a ∈ a* enviaMsg(a, "liberado")
38:       fim se
39:    fim se
40: fim laço
```

Figura 3.4: Implementação do algoritmo LA-DCOP.
Fonte: FELIPE S. BOFFO (2006)

Algoritmo 5 LA-DCOP para *tokens* potenciais

```
1:  para todo v ∈ V faça
2:     para 1 até nro. de tokens potenciais faça
3:        enviaToken(novoToken(potencial))
4:     fim para
5:  fim para
6:
7:  enquanto ∏_{v∈V} retidos[v] = 0 faça
8:     msg ← recebeMsg()
9:     se msg é do tipo "retido v" então
10:       retidos[v] ← retidos[v] ∪ msg.sender
11:    senão se msg é do tipo "libera v" então
12:       retidos[v] ← retidos[v] − msg.sender
13:    fim se
14: fim enquanto
15:
16: para todo v ∈ V faça
17:    a* ← ∀a ∈ retidos[v] competencia(a*, v) > competencia(a, v)
18:    para todo a ∈ a* faça
19:       enviaMsg(a, {"lock v", a*})
20:    fim para
21:    para todo a ∈ retidos[v] − a* faça
22:       enviaMsg(a, "libera v")
23:    fim para
24: fim para
25:
26: laço
27:    msg ← recebeMsg()
28:    se msg é do tipo "libera v" então
29:       retidos[v] ← retidos[v] − msg.sender
30:       goto 7
31:    fim se
32: fim laço
```

Figura 3.5: Implementação do algoritmo LA-DCOP para *tokens* especiais.
Fonte: FELIPE S. BOFFO (2006).

3.3 DSA

O DSA é outro algoritmo aplicável ao E-GAP e, conseqüentemente, à RoboCup Rescue. Do inglês *Distributed Stochastic Algorithm*, o DSA é um algoritmo distribuído

probabilístico e foi concebido para resolver problemas de satisfação e otimização de restrições de maneira distribuída (DSCPs e DCOPs).

Os DCOPs são uma versão distribuída dos Problemas de Otimização de Restrições - *Constraint Optimization Problem* (COP) - que são derivados dos Problemas de Satisfação de Restrições - *Constraint Satisfaction Problem* (CSP). Os Problemas de Satisfação de Restrições consistem em problemas de atribuição de valores a variáveis, respeitando restrições de valores entre elas. Cada restrição neste caso é proposicional (falso ou verdadeiro) e o objetivo final é obter uma solução que respeite todas as restrições.

Este tipo de representação baseada na satisfação de restrições não é adequado para a grande maioria dos problemas reais onde a solução pode ter graus de qualidade ou custo. Outro aspecto dos problemas reais é que estes normalmente têm muitas restrições o que torna impossível satisfazer todas elas. Para problemas deste tipo se deseja minimizar o número de restrições violadas e ainda otimizar seus valores associados. Com esse intuito, uma classe de problemas foi derivada dos CSPs e denominada Problemas de Otimização de Restrições. Neste tipo de problema uma função objetivo global é associada ao problema e o objetivo é maximizá-la ou minimizá-la. Esta função global utiliza um valor associado a cada restrição. Assim, cada valor atribuído a variáveis relacionadas por uma restrição gera um valor diferente que é utilizado pela função objetivo global. Abordagens utilizadas para solucionar CSPs também são utilizadas neste tipo de problema que, adicionalmente, oferece recursos para que heurísticas de aproximação possam ser utilizadas com eficiência uma vez que se pode prever se o caminho que está sendo utilizado na busca por uma solução é ou não promissor pela aproximação da função objetivo.

Os DCOPs diferem dos COPs, pois cada variável é associada a um agente e somente este tem controle sobre seu valor. As restrições são distribuídas entre os agentes que compõe o sistema e um processo de interação entre eles deve ser utilizado para se obter uma solução.

Formalmente um DCOP consiste de *n* variáveis $V = \{x_1 , x_2 ,..., x_n\}$ que podem assumir valores de um domínio discreto $\{D_1, D_2,..., D_n\}$ respectivamente. Cada variável é associada a um agente que tem controle sobre seu valor. O objetivo do agente é escolher um valor para sua variável que otimize uma função de objetivo global. Esta função é descrita como o somatório sobre o conjunto de restrições valoradas relacionadas a par de variáveis.

Assim, para um par de variáveis (X_i, X_j) existe uma função de custo definida como F(*ij*): D_i X D_j -> N.

No DSA os agentes escolhem aleatoriamente um valor do domínio para atribuir a sua variável. Feito isso, uma série de iterações acontecem e, em cada uma delas, um agente envia informações do seu estado atual, ou seja, o valor de sua variável, para seus vizinhos. Se o agente mudou o seu valor na etapa anterior, ele recebe a informação de seu novo estado e decide também probabilisticamente manter o seu valor atual ou mudar para um novo. O objetivo é o de mudar de valor eventualmente. Um esboço do DSA pode ser visto no algoritmo abaixo.

Algoritmo 6 **Esboço do DSA executado por todos agentes.**

```
Escolher aleatoriamente um valor
while (nenhuma condição é cumprida) do
   if (um novo valor é atribuído ) then
      enviar o novo valor para os vizinhos
   end if
   recolher novos valores dos vizinhos , se houver
   selecionar e atribuir o próximo valor
end while
```

Figura 3.6: Esboço do DSA executado por todos agentes.
Fonte: Adaptado de Weixiong Zhang et al., 2002

O passo mais crítico do DSA é de um agente decidir o próximo valor para sua variável com base no seu estado atual e acreditando nos estados dos agentes vizinhos. Se o agente não pode encontrar um novo valor para melhorar o seu estado atual, não irá alterar seu valor corrente. Se existe um valor que melhore a qualidade o agente pode ou não mudar para este novo valor baseado no sistema de regime probabilístico comentado acima.

4 EXPERIMENTOS E RESULTADOS

Os criadores do LA-DCOP argumentam que seu algoritmo é melhor que abordagens anteriormente propostas, comparando-o diretamente com o DSA (ZHANG, 2002) em experimentos conduzidos com um simulador abstrato. Seus resultados apontam que o algoritmo LA-DCOP apresenta desempenho superior ao DSA em todos os experimentos realizados.

Os autores do Swarm-GAP apresentam resultados que indicam que, apesar de seu algoritmo apresentar resultados inferiores ao LA-DCOP em simulações abstratas, obtém resultados equivalentes no simulador da RoboCup Rescue. Como dito anteriormente, este trabalho tem como objetivo retomar a comparação do desempenho do DSA com o LA-DCOP adotando o ambiente da Robocup Rescue e verificar se a superioridade publicada se faz presente também neste ambiente. Este capítulo esta organizado da seguinte forma: a seção 4.1 apresenta a implementação do DSA no ambiente da Robocup Rescue; a seção 4.2 enfatiza a definição dos experimentos com o mapa de Kobe e por fim, a seção 4.3 analisa os resultados obtidos com o DSA.

4.1 Implementação do DSA para a Robocup Rescue

O algoritmo implementado para a RoboCup Rescue não considera todos os detalhes do DSA original apresentado na seção 3.3 dado que o E-GAP, problema que modela formalmente a RoboCup Rescue, é mais simples que um DCOP. Este último é o problema para o qual o algoritmo foi originalmente desenvolvido. Os agentes executando um dos algoritmos anteriormente propostos, LA-DCOP ou Swarm-GAP, não se comunicam entre si (FERREIRA JR., BOFFO, BAZZAN, 2008). Da mesma forma, para garantir equivalência na comparação, os agentes utilizando o algoritmo DSA não irão se comunicar.

Os agentes que representam as brigadas de incêndio têm sua competência *kij* dada por uma função que considera a distância euclidiana do agente até determinado incêndio e a extensão do dano que este incêndio está causando. Desta forma, uma brigada de incêndio *i* tem competência *kij* para atuar no foco de incêndio *j* dado pelo inverso da distância de *i* até *j* e por quão severo é o dano causado por *j*. Com isso, o agente é mais competente para atuar em focos de incêndio que estão mais perto e que são menos severos para que possam ser rapidamente extinguidos e não venham a se propagar. Este cálculo está formalizado na equação 2, onde *d*(*i, j*) é a distância euclidiana mencionada e *f*(*j*) é a quantidade de fogo associada a *j*. Estes valores são equilibrados por α e (1 − α). Nos experimentos conduzidos aqui foi adotado sempre α = 0.5 para dar igual peso para ambos os fatores.

$$k_{ij} = \frac{\max_{n \in \mathcal{J}}\{d(i,n)\} - d(i,j)}{\max_{n \in \mathcal{J}}\{d(i,n)\}} * \alpha + \frac{\max_{n \in \mathcal{J}}\{f(n)\} - f(j)}{\max_{n \in \mathcal{J}}\{f(n)\}} * (1-\alpha) \quad (2)$$

Com relação às forças policiais, cujo objetivo é o desbloqueio das ruas, suas competências são determinadas pela distância euclidiana, exatamente como no caso dos bombeiros, e pelo tamanho dos bloqueios. Com isso, os agentes são mais competentes para desbloquear grandes obstruções para que se possa, rapidamente, voltar a circular pela rua mesmo que minimamente. Este cálculo está formalizado na equação 3, onde *d*(*i, j*) é a distância euclidiana mencionada e *w*(*j*) é a quantidade de escombros associada a *j*. Estes valores também são equilibrados por α e (1 − α). Nos experimentos conduzidos aqui foi adotado sempre α = 0.5 para dar igual peso para ambos os fatores.

$$k_{ij} = \frac{\max_{n \in \mathcal{J}}\{d(i,n)\} - d(i,j)}{\max_{n \in \mathcal{J}}\{d(i,n)\}} * \alpha + \left(1 - \frac{\max_{n \in \mathcal{J}}\{w(n)\} - w(j)}{\max_{n \in \mathcal{J}}\{w(n)\}}\right) * (1-\alpha) \quad (3)$$

Finalmente, com relação às ambulâncias, os agentes são mais competentes para lidar com feridos que estão mais próximos e menos feridos. Da mesma forma que com as forças policiais, objetiva-se que os menos feridos sejam rapidamente resgatados para que sua situação não se agrave. Este cálculo está formalizado na equação 4, onde *d*(*i, j*) é a distância euclidiana mencionada e *h*(*j*) é o índice de saúde associado a *j*. Estes valores também são equilibrados por α e (1 − α). Nos experimentos conduzidos aqui foi adotado sempre α = 0.5 para dar igual peso para ambos os fatores

$$k_{ij} = \frac{\max_{n \in \mathcal{J}}\{d(i,n)\} - d(i,j)}{\max_{n \in \mathcal{J}}\{d(i,n)\}} * \alpha + \frac{\max_{n \in \mathcal{J}}\{h(n)\} - h(j)}{\max_{n \in \mathcal{J}}\{h(n)\}} * (1 - \alpha) \quad (4)$$

Apesar das competências dos agentes acima descritas serem baseada em heurísticas bem simples, elas podem ser aplicadas em todos os algoritmos experimentados neste trabalho.

Dois mecanismos foram implementados para garantir a eficiência dos algoritmos dado as equações de competência acima descritas. No primeiro mecanismo os agentes verificam antecipadamente se o acesso a tarefa escolhida está ou não bloqueado. Tarefas que estão inacessíveis não são consideradas. No segundo mecanismo, uma vez que determinada tarefa foi escolhida, esta será realizada pelo agente até sua conclusão ou a escassez de seus recursos (no caso da brigadas de incêndio, o término de sua água).

O processo de escolha das tarefas pelos agentes utilizando o DSA, simplificadamente, é a seguinte: as tarefas percebidas são ordenadas pela competência do agente e este tem probabilidade *P* de escolher a primeira tarefa da lista. Caso a tarefa não seja escolhida, o agente repete essa operação para a segunda tarefa, com a mesma probabilidade, e assim por diante.

Na prática, esta é uma versão do Swarm-GAP onde a probabilidade é constante e independente da competência do agente. No Swarm-GAP essa probabilidade varia exponencialmente de acordo com a competência.

4.2 Definição dos experimentos

Os experimentos discutidos aqui foram executados em um mapa que é largamente utilizado pela comunidade da RoboCup Rescue: o mapa Kobe. As implementações do LA-DCOP e do Swarm-GAP para a RoboCup Rescue foi obtida com seus autores. O DSA foi implementado como descrito na seção anterior. Para tornar o cenário mais complexo, as quantidades de incêndio e agentes utilizados são o dobro do número padrão (cada mapa tem uma configuração padrão). Assim, são utilizados 12 equipes de ambulância, 20 bombeiros, 16 forças policiais, 144 civis e 12 focos de incêndio. A figura 4.1 mostra este mapa na sua configuração inicial.

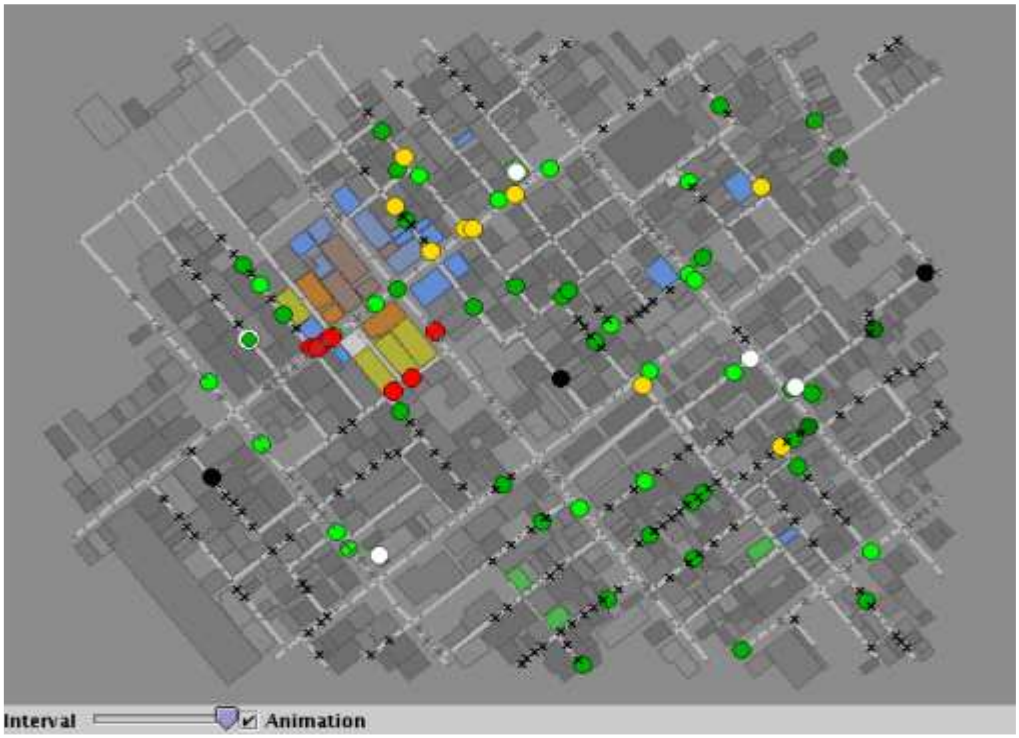

Figura 4.1: Mapa de Kobe com todos os agentes.
Fonte: TAKESHI MARIMOTO, 2002.

No mapa de Kobe foram adotados 18 agentes *Ambulance Team*, 30 agentes *Fire Brigade* e 24 agentes *Police Force*. Neste caso o mapa é configurado com 216 civis e 30 focos de incêndio iniciais. O restante dos fenômenos da catástrofe é aleatório, não havendo como prever quantos fenômenos acontecerão por simulação.

Para medir o desempenho dos algoritmos, usa-se o *Score* da norma RoboCup Rescue, equação citada na seção 2.4. Na seção 2.4 encontram-se as especificações de cada algarismo e os detalhes de como este cálculo é realizado. Foram executadas 8 repetições de cada simulação com o objetivo de garantir a validade estatística dos resultados à medida que várias questões de aleatoriedade estão envolvidas em cada simulação.

Para melhor desempenho da simulação é necessário uma máquina robusta de 4G de RAM com sistema operacional *Linux* para que os processos ocorram corretamente. Em máquina virtual não é aconselhável já que testes foram feitos não apresentando bom resultado.

4.3 Resultados obtidos

Experimentos iniciais foram executados para identificar o valor para a probabilidade *P* que levam o DSA a obter os melhores resultados. Foram experimentados valores entre 0.1 e

0.8. Valores menores ou maiores que estes levam o DSA a um comportamento dispersivo e ineficiente, alocando quaisquer tarefas ou poucas tarefas. Os resultados são resumidos na tabela 4.1 e ilustrados na figura 4.2.

Tabela 4.1: Resultados de testes para identificar valores para a probabilidade *P*

	0.2	0.3	0.4	0.5	0.6	0.7	0.8
1	71,46	93,45	107,13	117,15	93,45	93,56	61,73
2	69,84	88,12	85,9	86,89	88,12	88,75	72,95
3	73,56	99,37	105,84	107,02	99,37	95,7	72,45
4	54,12	74,55	88,3	82,27	101,55	85,67	52,34
5	78,4	78,75	74,14	77,35	104,3	88,94	59,9
6	71,34	84,99	84,9	85,4	84,99	73,5	71,8
7	78,96	93,22	95,6	114,53	93,22	92,76	77,5
8	66,58	72,34	102,34	88,26	72,34	72,09	65,09
Média	70,5325	85,59875	93,01875	94,85875	92,1675	86,37125	66,72
Desvio	7,8291	9,7165	11,6638	15,5531	10,357	8,9593	8,4050

Fonte: O Autor

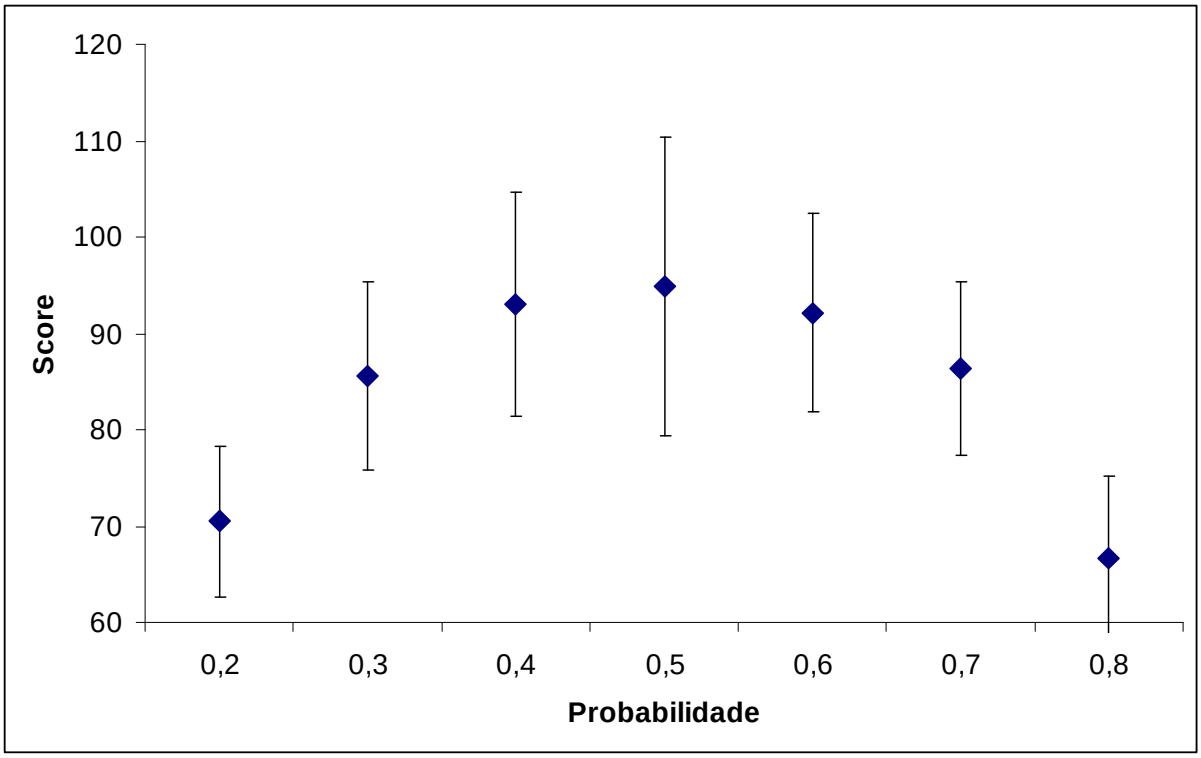

Figura 4.2: Gráfico da probabilidade *P* que levam o DSA a obter os melhores resultados.
Fonte: O Autor

Em média, os melhores resultados para o mapa Kobe foram obtidos com $P = 0.5$. A fim de fazer uma comparação justa e evitar detalhes desnecessários na comparação entre o LA-DCOP, o Swarm-GAP e o DSA, foi usada a pontuação alcançada pelo melhor valor do

threshold T no LA-DCOP, do melhor valor do estímulo *s* no Swarm-GAP e do melhor valor da probabilidade *P* no DSA. O melhor desempenho no mapa Kobe do LA-DCOP é atingido por T = 0.6, enquanto o melhor desempenho de Swarm-GAP é *s* = 0.2.

A tabela abaixo mostra os resultados alcançados nas simulações. Foram procedidas oito execuções de cada algoritmo (repetiram-se os experimentos anteriores):

Tabela 4.2: Resultado obtido nas simulações.

	LA-DCOP	Swarm-GAP	DSA
1	79,15	85,38	117,15
2	78,91	78,84	86,89
3	104,44	73,93	107,02
4	103,97	79	82,27
5	78,78	105,7	77,35
6	81,3	85,95	85,4
7	84,48	98,47	114,53
8	112,17	81,21	88,26
Média	90,4	86,06	94, 8588
Desvio	13, 972743	10, 775923	15, 5531

Fonte: O Autor

Em quase todos os testes, o Swarm-GAP e o LA-DCOP tem desempenho igualmente bom, como já demonstrado em outros trabalhos (BAZZAN A. L. et al, 2007). Não há diferença estatisticamente significativa entre LA-DCOP e Swarm-GAP para um teste ANOVA[20] com confiança 0.05, como demonstrado previamente. Em média, o LA-DCOP é ligeiramente superior.

A figura 4.3 resume estes resultados, apresentando as médias e o desvio padrão, como barras de erro, para os resultados detalhados na tabela 4.2

[20] Nome de um teste de estatística proposto para verificar a diferença entre as médias. Ele indica se existe diferença significativa ou não.

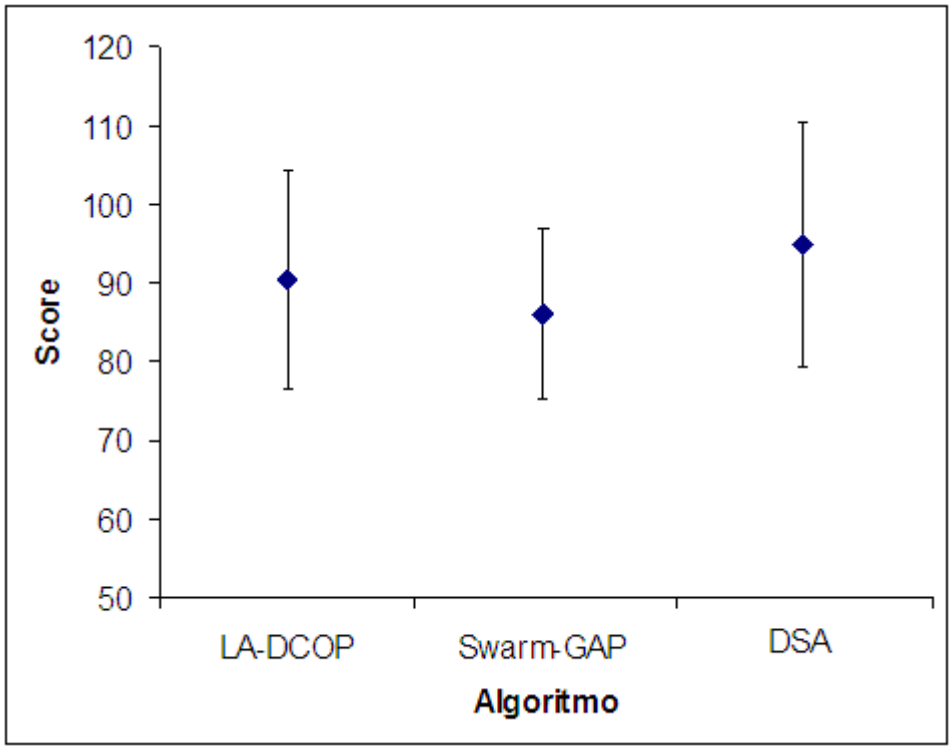

Figura 4.3: Média do *Score* com barras de erro para o mapa Kobe
Fonte: O Autor

O DSA foi o algoritmo que obteve o melhor resultado em média e o maior valor absoluto. Apesar disso, não existe diferença estatisticamente significativa entre os métodos para um teste ANOVA com confiança 0.05. Esses resultados indicam que as conclusões previamente formuladas com base em simulações abstratas não se confirmam no simulador da RoboCup Rescue. Pode-se afirmar que o DSA é equivalente ao LA-DCOP e o Swarm-GAP.

Essa equivalência estatística considerando que o processo de tomada de decisão utilizando o DSA não considera a utilização de parâmetros como a competência, traz vantagens significativas para esta abordagem. Determinar métodos para o cálculo da competência despende esforço e muito conhecimento sobre o problema em que o algoritmo está sendo empregado. Para algoritmos de uso geral, como são os estudados neste trabalho, estar livre deste tipo de heurística é bastante relevante. É possível que heurísticas mais sofisticadas melhorem os resultados dos demais algoritmos, mas não se acredita que seus resultados seriam estatisticamente melhores.

Na Robocup Rescue, os participantes do campeonato utilizam várias heurísticas que normalmente são implementadas de uma forma objetiva. Estas incluem o particionamento do mapa em setores de ação, alterações na rota, utilização de comunicação para alcançar uma boa

coordenação, etc. Os algoritmos de uso geral, como os testados aqui, tendem a ser ultrapassados pelo vencedor da competição da Robocup Rescue.

CONCLUSÃO

O resgate em situações de desastre é de fato um dos mais relevantes problemas sociais da atualidade, pois estas catástrofes muitas vezes são responsáveis por significativas causas de perdas de vidas nos países desenvolvidos. Utilizar recursos da Robótica e IA como simulações para aperfeiçoar a atuação de equipes resgate é uma forma direta de contribuir para a transformação deste quadro.

Este trabalho estudou detalhadamente os conceitos, princípios e funcionamento da RoboCup Rescue, incluindo a fundamentação dos sistemas multiagentes e conseqüentemente o modelo E-GAP para a alocação dinâmica de tarefas aos agentes neste ambiente. Foram estudados os algoritmos Swarm-GAP, LA-DCOP e o DAS, os quais seguem linhas diferentes para lidar com problemas modelados como E-GAP's, e são aplicados na RoboCup Rescue utilizando times que os implementem para que os agentes determinem quais tarefas realizar em determinado momento.

Embora já houvessem conclusões previamente formuladas em abordagens anteriormente propostas, de que o LA-DCOP é melhor e que seus resultados apontam desempenho superior ao DSA em todos os experimentos realizados e que, o Swarm-GAP apresenta resultados que indicam equivalência no simulador da RoboCup Rescue, estas não se confirmam especificamente no simulador da liga.

Este trabalho teve como objetivo retomar a comparação do desempenho do DSA com o LA-DCOP adotando o ambiente da RoboCup Rescue e verificando se a superioridade antes publicada se faz presente também neste ambiente. O DSA foi o algoritmo que obteve o melhor resultado em média e o maior valor absoluto, podendo-se ainda afirmar que o DSA é equivalente estatisticamente ao LA-DCOP e ao Swarm-GAP. Demonstra-se que o algoritmo DSA é tão eficiente para a RoboCup Rescue quanto seus sucessores mais sofisticados.

REFERÊNCIAS BIBLIOGRÁFICAS

BAASE, S.; GELDER, A. van. **Computer Algorithms**: introduction to design & analysis. 3. ed. Reading, USA: Addison-Wesley, 2000. 688p.

BOFFO, F. S. **Alocação Distribuída de tarefas em sistemas dinâmicos de larga escala: estudo empírico de um método inspirado em insetos sociais** 2006. 73 f. Trabalho de Diplomação (Graduação em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

BONABEAU, E.; DORIGO, M.; THERAULAZ, G. **Swarm intelligence: from natural to artificial systems**. 1.ed. New York, USA: Oxford University Press, 1999. 307p.

BROOKS, R. **Intelligence without Representation. Artificial Intelligence**, n. 47. Pp. 139-159. Elsevier, 1991

CLICK TEAM - Clik & Play Official Site, 1996. Disponível em: < http://www.clickteam.com> Acesso em: Jun 2008

DURFEE, E.; LESSER, V.; CORKILL, D. Trends in Cooperative Distributed Problem Solving. IEEE Transactions on Knowledge and Data Engineering, Los Alamitos, v.1, n.1, p.63–83, 1989.

DROGOUL, A., FERBER, J., **Multi-Agent Simulation as a Tool for Modeling Societies: Application to Social Differentiation in Ant Colonies, Lecture Notes In Computer Science**; Vol. 830, Springer-Verlag, London – UK, 1992.

EXCELENTE-TOLEDO, C. B.; JENNINGS, N. R. **The Dynamic Selection of Coordination Mechanisms**. Autonomous Agents and Multi-Agent Systems, Amsterdam, v.9, p.55–85, 2004.

FERREIRA JR., P. R. **Coordenação de Sistemas Multiagentes Atuando em Cenários Complexos Baseada na Divisão de Trabalho dos Insetos Sociais** 2008. 115 f. Proposta de Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

FERREIRA JR., P. R.; BAZZAN, A. **Swarm-GAP: a swarm based approximation algorithm for E-GAP**. In: INTERNATIONAL WORKSHOP ON AGENT TECHNOLOGY FOR DISASTER MANAGEMENT, 1., 2006, New York, USA. Proceedings. . . [S.l.: s.n.], 2006. p.49–55.

FERREIRA JR., P. R.; BAZZAN, A. **Self-Organization of Agents to solve Machine Sequencing Problems**. In: WORKSHOP ON ADAPTATION AND LEARNING IN AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 2006, Hakodate, Japan. Proceedings.. . [S.l.: s.n.], 2006. p.70–75.

FERREIRA JR., P. R.; BOFFO, F. S.; BAZZAN, A. L. C.: **Using Swarm-GAP for Distributed Task Allocation in Complex Scenarios**. In: AAMAS Workshops, MMAS 2006, LSMAS 2006, and CCMMS 2007, 2008, Honolulu, HI. Massively Multi-Agent Technology. Heidelberg : Springer Berlin, 2008. v. 5043. p. 107-121. Disponível em: <www.inf.ufrgs.br/maslab/pergamus/pubs/Swarm-GAP-LNCS.zip>

FROZZA, R. **SIMULA – Ambiente para desenvolvimento de Sistemas Multi-Agentes Reativos** 1999. Material de apoio (Curso de pós-graduação em Ciência da Computação) - Universidade Federal do Rio Grande do Sul. Disponível em: <http://simula.sourceforge.net> Acesso em: Jun. 2008

GOLDMAN, C.; ZILBERSTEIN, S. Decentralized Control of Cooperative Systems: categorization and complexity analysis. **Journal of Artificial Intelligence Research**, [S.l.], v.22, p.143–174, 2004.

HPP - 3D em NetLogo 2007. Disponível em: <http://ccl.northwestern.edu/netlogo/models/community/Example-HPP-3D> Acesso em jun. 2008

HUBNER, Jomi Fred; SICHMAN, Jaime Simão. **Aplicação de Organização de Sistemas Multiagentes em Futebol de Robôs**. São Paulo: p. 1-19. 2003.

JENNINGS, N. R. **Coordination Techniques for Distributed Artificial Intelligence**. In: O'HARE, G. M. P.; JENNINGS, N. R. (Ed.). Foundations of Distributed Artificial Intelligence. New York: John Wiley & Sons, 1996. p.187–210.

KITANO, H et al. **RoboCup Rescue: Search and rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research**.. In Proceedings of the IEEE Conference on Man, Systems and Cybernetics. IEEE Press, 1999.

KITANO, H. **RoboCup Rescue: a grand challenge for multi-agent systems**. In: INTERNATIONAL CONFERENCE ON MULTIAGENT SYSTEMS, 4., 2000. **Proceedings**. . .[S.l.: s.n.], 2000.

MACAL, Charles M. **Tutorial on agent-based modeling and simulation**. Winter Simulation Conference. 2005.

NAIR, R.; ITO, T.; TAMBE, M.; MARSELLA, S. Task Allocation in the Rescue Simulation Domain: a short note. In: ROBOCUP-2001: ROBOT SOCCER WORLD CUP V, 2002. Proceedings. . . Berlin: Springer-Verlag, 2002. (Lecture Notes in Computer Science).

NWANA, H. S.; LEE, L. C.; JENNINGS, N. R. **Coordination in Software Agent Systems**. **The British Telecom Technical Journal**, [S.l.], v.14, p.19–88, 1996.

OKAMOTO, S. **Allocating Roles in Large Scale Teams: an empirical evaluation**. 2003. 42 f. Dissertação (Mestrado em Ciência da Computação) – Department of Computer Science, University of Southern California, Los Angeles, EUA.

SCERRI, P.; FARINELLI, A.; OKAMOTO, S.; TAMBE, M. **Allocating tasks in extreme teams**. In: FOURTH INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 2005, New York, USA. Proceedings. . .ACM Press, 2005. p.727–734.

REIS, L.P., **Coordenação em sistemas multiagente: aplicações na gestão universitária e futebol robótico**, Tese de Doutorado, Universidade do Porto, 2003..

ROBOCUP – RoboCup Official Site, 2001. Disponível em: <http://www.RoboCup.org> Acesso em: mar. 2008.

ROBOCUP RESCUE – RoboCup Rescue Official Site, 2002. Disponível em: <http:// www.RoboCuprescue.org> Acesso em: mar. 2008.

RUSSEL, Stuart J.; Peter NORVIG, Artificial Intelligence: **A Modern Approach, Englewood Cliffs**, Nova Jersey: Prentice Hall, 1995

SANTANTANCHE A.; TEIXEIRA CAMILLO. C. A. **Múltiplas perceptivas de objetos no contexto educacional** 2000. Minicurso ministrado no Simpósio Brasileiro de Informática na Educação - Maceió, AL.

SCERRI, P.; FARINELLI, A.; OKAMOTO, S.; TAMBE, M. **Allocating tasks in extreme teams**. In: INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 4., 2005, Utrecht, The Netherlands. Proceedings. . .New York: ACM Press, 2005. p.727–734.

SKINNER, C. **RoboCup Rescue Simulation Project**. Disponível em: <http://sourceforge.net/projects/roborescue>. Acesso em: mar 2008.

STARLOGO – StarLogo on the web. Disponível em: http://education.mit.edu/starlogo Acesso em Jun 2008.

TAKAHASHI, T. **RoboCup Rescue Simulator Manual**. Disponível em: <http://sakura.meijo-u.ac.jp/ttakaHP/>. Acesso em: mar 2008.

TAKESHI MARIMOTO, **Viewer for RoboCup Rescue Simulation System**, 2002. Disponível em: <http://ne.cs.uec.ac.jp/~morimoto/rescue/viewer/>.

THERAULAZ, G.; BONABEAU, E.; DENEUBOURG, J. **Response Threshold Reinforcement and Division of Labor in Insect Societies**. In: ROYAL SOCIETY OF LONDON B, 1998. Proceedings. . . [S.l.: s.n.], 1998. n.1393, p.327–332. (Biological Sciences, v.265).

TOMOICHI TAKAHASCHI, **RoboCup Rescue Simulator Manual**, 2000. Disponível em: <http://kiyosu.isc.chubu.ac.jp/RoboCup/Rescue/>.

WOOLDRIDGE, M. J. **An introduction to multiagent systems**. New York: JohnWiley, 2002. 348p.

WEISS, G. (Ed.). Multiagent Systems: a modern approach do distributed artificial intelligence. Cambridge, Massachusetts: The MIT Press, 1999. 619p.

WEIXIONG ZHANG, G. WANG, L. WITTENBURG, Distributed stochastic search for distributed constraint satisfaction and optimization: parallelism, phase transitions and performance, in: Proc. AAAI-02 Workshop on Probabilistic Approaches in Search, 2002, pp. 53-59.