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

Um Ambiente de Evolução de Comportamentos para Sistemas Multiagentes Reativos

por

ANDRE ZANKI CORDENONSI

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Luis Otávio Alvares
Orientador

Porto Alegre, março de 2000.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Cordenonsi, Andre

Um Ambiente de Evolução de Comportamentos para Sistemas Multiagentes Reativos / por Andre Cordenonsi – Porto Alegre: PPGC da UFRGS, 1999
92 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 1999. Orientador: Alvares, Luis Otávio

1. Inteligência Artificial Distribuída. 2. Simulação. 3. Sistemas Multiagentes. 4. Sistemas Multiagentes Reativos. 5. Algoritmos Evolutivos. I. Alvares, Luis Otávio. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profa. Wrana Maria Panizzi
Pró-Reitor de Pós-Graduação: Prof. Franz Rainer Semmelmann
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador do PPGC: Prof. Carla Maria Dal Sasso
Bibliotecária-Chefe do Instituto de Informática: Beatriz Haro

Agradecimentos

Inicialmente, agradeço ao meus pais, pelo empenho, dedicação e amor demonstrados em todos estes anos, que formaram o ser humano que sou hoje. Aos meus irmãos, pela força e companheirismo durante toda a minha vida.

À minha namorada, Giliane, quase noiva, pelo amor, carinho e compreensão, principalmente nas intermináveis horas e dias que estendiam-se na frente do computador, que pareciam que não acabavam mais. À minha cunhada, Lisiane, que me agüentou durante todo este período.

Aos amigos novos e antigos, que não podem nem serão esquecidos: Leandro "Alegrete" Becker, Marcos "Tigrão" Vinícius, Mauro Munoz e Fabis Munoz, Edson Prestes, Marcelo "Treva" Thielo, Evandro Luiz Um Cara Feliz (que apresentou-me a minha namorada... Valeu!!), aos paraenses, Rodrigo "Graxaim" Maffazzioli, Luciano "Hélvio" Ferreira, Marcelo "Dunga" Stroschein (diretamente de Brasília), Felipão Müller, ao pessoal do PET Informática – UFSM, à Diretoria e o seu Adão, Renata Mattos, Guilherme "Minduim" Dhein, Marcelo Dalla Rosa, César "Laga" Leonhardt, Lara, Lenise, Fábia, Lisandra e Letícia, Cassal e Ana e a todos que não estão aqui, mas estão no meu coração.

Aos professores e funcionários do PPGC, sempre solícitos e compreensivos, que muito contribuíram para a finalização desta etapa da minha vida.

Ao professor, orientador e amigo, Luis Otávio Alvares, pela dedicação, grande orientação e companheirismo durante estes últimos anos.

Ao Frederico e a Rejane, pela ajuda e disposição.

Ao CNPq, sem o qual este trabalho não teria sido realizado.

Ao Sepultura e ao Mano Lima, que intercalaram-se nos mais diversos momentos durante a conclusão deste curso.

E, finalmente, a nobre nação colorada, sofrida mas fiel, que não merece os dirigentes que tem.

Sumário

Lista de Abreviaturas **6**
Lista de Símbolos **7**
Lista de Figuras **8**
Lista de Tabelas **9**
Resumo **10**
Abstract **11**

1 Introdução **12**

1.1 Contexto e motivações **12**
1.2 Vida Artificial e a Evolução de Comportamentos para um Sistema Multiagente Reativo – uma visão geral **13**
1.3 Objetivos e Estrutura da Dissertação **13**

2 Vida Artificial **15**

2.1 Princípios Básicos **15**
2.1.1 A biologia do possível *17*
2.1.2 Método sintético *17*
2.1.3 Vida real *17*
2.1.4 Toda vida é forma *17*
2.1.5 Construção bottom-up *18*
2.1.6 Processamento paralelo *18*
2.1.7 Condução à emergência *18*
2.2 Domínios de Problemas para Vida Artificial **18**
2.2.1 Aprendizagem *18*
2.2.2 Realidade virtual *18*
2.2.3 Agentes autônomos *19*
2.2.4 Manufatura inteligente *19*
2.2.5 Educação *19*
2.3 Modelagem de Sistemas de Vida Artificial **19**
2.3.1 Movimento de animais artificiais *19*
2.3.2 Automata celulares *20*
2.3.3 Vírus *23*
2.3.4 Minhocas *24*
2.3.5 Mundos virtuais *25*
2.4 Algoritmos Evolutivos **26**
2.5 Sistemas de Mundos Virtuais **32**

3 Sistemas Multiagentes **35**

3.1 Introdução **35**
3.1.1 Sistemas multiagentes cognitivos *36*
3.1.2 Sistemas multiagentes reativos *37*
3.2 Definição de regras em SMAR **38**

3.3 O Ambiente Simula 39
3.3.1 Módulo de definição 41
3.3.2 Módulo de execução do ambiente Simula 42

4 Proposta do Ambiente de Evolução de Comportamentos 43

4.1 Introdução ao Simula++ 43
4.1.1 Tipo de agentes 43
4.1.2 Definição das regras 43
4.1.3 Ambiente de simulação 44
4.1.4 Evolução e adaptabilidade 45
4.2 Modelo de Evolução para as Regras de Comportamento 45
4.2.1 Estrutura dos Agentes 45
4.2.2 Estratégia de Controle 46
4.2.3 Recombinação Externa 48
4.2.4 Recombinação Interna 49
4.2.5 Mutação 51
4.2.6 Função de Fitness 53

5 O Protótipo do Ambiente Simula++ 54

5.1 A interface do Protótipo 54
5.1.1 Menu Arquivo 54
5.1.2 Menu Configurações 55
5.1.3 Menu Executar 65
5.1.4 Menu Ajuda 66
5.2 Funcionalidades do Sistema 66
5.2.1 Definição dos Agentes 67
5.2.2 Definição de uma Instância de Simulação 67
5.2.3 Execução da Simulação 67

6 Estudo de Caso: o problema dos Robôs Mineradores 68

6.1 Características do Problema 69
6.2 Modelagem dos Agentes Robôs *Petit Poucet* 69
6.3 Modelagem dos Agentes Robôs no Ambiente Simula++ 73
6.4 Simulação e Análise dos Resultados 74

7 Considerações Finais 83

Anexo 1 Lista das Regras Pré-Definidas no Ambiente Simula++ 86
Bibliografia 90

Lista de Abreviaturas

AE	Algoritmos Evolutivos
IA	Inteligência Artificial
AG	Algoritmos Genéticos
IAD	Inteligência Artificial Distribuída
RL	*Reinforcement Learning*
RV	Realidade Virtual
SMA	Sistemas Multiagentes
SMAR	Sistemas Multiagentes Reativos
Fig.	FIGURA
Tab.	TABELA
Núm.	Número

Lista de Símbolos

>=	Procedência Maior ou Igual
>	Procedência Maior
<	Procedência Menor
<=	Procedência Menor ou Igual
=	Procedência Igual
+	Sinal de Soma
-	Sinal de Subtração
*	Sinal de Multiplicação
/	Sinal de Divisão
%	Resto da Divisão de Inteiros
!	Operador Lógico de Negação
&	Operador Lógico de Conjunção
|	Operador Lógico de Disjunção

Lista de Figuras

FIGURA 2.1 – Simulação de um Grupo de Boids....20
FIGURA 2.2 – O Jogo da Vida....21
FIGURA 2.3 – Mundo Virtual....25
FIGURA 2.4 – Modos de Percepção em um Mundo Virtual....26
FIGURA 2.5 – Algoritmo Genético [KOZ 98]....29
FIGURA 2.6 – Superfície de Aptidão num Espaço Bidimensional....30
FIGURA 2.7 – Algoritmo Evolutivo [KOZ 98]....31
FIGURA 2.8 – Modelo para Construção de Mundos Virtuais....32
FIGURA 3.1 – Arquitetura para um Agente Cognitivo [SIC 95]....37
FIGURA 3.2 – Arquitetura do Ambiente Simulla [FRO 97]....41
FIGURA 4.1 – Arquitetura do Ambiente Simula++....45
FIGURA 4.2 – Algoritmo de Execução dos Agentes....47
FIGURA 4.3 – Ciclo de Vida de um Agente....47
FIGURA 4.4 – Modelo para a Recombinação Externa de Um Ponto e Dois Pontos....48
FIGURA 4.5 – Exemplo de Recombinação Externa de um Ponto....49
FIGURA 4.6 – Modelo para a Recombinação Interna....50
FIGURA 4.7 – Agentes Genitores para a Recombinação Interna....50
FIGURA 4.8 – Agente Filho Gerado Utilizando a Recombinação Interna....51
FIGURA 4.9 – Operador de Mutação: Modificação do Número de Regras de Comportamento....52
FIGURA 4.10 – Operador de Mutação: Modificação das Regras de Comportamento....52
FIGURA 4.11 – Exemplo do Operador de Mutação....53
FIGURA 5.1 – Interface do Ambiente Simula++....54
FIGURA 5.2 – Menu Arquivo....55
FIGURA 5.3 – Menu Configurações....55
FIGURA 5.4 – Configurações do Algoritmo Evolutivo....56
FIGURA 5.5 – Definição dos Tipos de Agentes....57
FIGURA 5.6 – Definição da Imagem para o Tipo de Agente....58
FIGURA 5.7 – Definição de Variáveis....58
FIGURA 5.8 – Definição de Comportamentos....59
FIGURA 5.9 – Definição de Condições....60
FIGURA 5.10 – Definição de Ações....61
FIGURA 5.11 – Definição do Critério de Parada....62
FIGURA 5.12 – Definição das Dimensões do Ambiente....63
FIGURA 5.13 – Definição das Posições dos Agentes....63
FIGURA 5.14 – Definição dos Arquivos de Execução....64
FIGURA 5.15 – Menu Executar....65
FIGURA 5.16 – Ambiente de Simulação....65
FIGURA 5.17 – Menu Ajuda....66
FIGURA 6.1 – Problema dos Robôs Mineradores....68
FIGURA 6.2 – Número de Robôs Mineradores x Tempo para Coleta de Minerais [DRO 93]....70
FIGURA 6.3 – Modelo Esquemático dos Robôs Estivadores....72
FIGURA 6.4 – Fitness Máximo....74
FIGURA 6.5 – Número de Robôs Mineradores Grupo E x Tempo para Coleta de Minerais....76
FIGURA 6.6 – Número de Robôs Mineradores Grupo S x Tempo para Coleta de Minerais....78
FIGURA 6.7 – Comparação entre dados do Grupo E e o Grupo S....78
FIGURA 6.8 – Problema do Tráfego de Robôs....80
FIGURA 6.9 – O Problema do Tráfego de Robôs no Ambiente Simula++....80
FIGURA 6.10 – Energia Consumida pelos Robôs x Número de Robôs Mineradores....81
FIGURA A.1 – Movimento Direcionado....86
FIGURA A.2 – Deixa Pista....86
FIGURA A.3 – Segue Pista....86
FIGURA A.4 – Percebe Agente....88

Lista de Tabelas

TABELA 6.1 – Comportamento dos Agentes Robôs Petit Poucet 01 70
TABELA 6.2 – Comportamento dos Agentes Robôs Petit Poucet 02 71
TABELA 6.3 – Comportamento dos Agentes Robôs Petit Poucet 03 71
TABELA 6.4 – Comportamento dos Agentes Robôs Dockers.....73
TABELA 6.5 – Características do Modelo dos Robôs Mineradores 75
TABELA 6.6 – Simulações dos Robôs Mineradores Grupo E.....76
TABELA 6.7 – Performance dos Robôs Mineradores.....76
TABELA 6.8 – Comportamento dos Agentes Robôs de maior Grau de Fitness (Grupo S).....77
TABELA 6.9 – Simulações dos Robôs Mineradores Grupo S.....79
TABELA 6.10 – Tempo de Simulação vs Energia Consumida pelo Grupo S.....82

Resumo

A Inteligência Artificial Distribuída(IAD) baseia-se no comportamento social de indivíduos (humanos, animais ou artificiais), principalmente nas ações e interações entre estes. Os Sistemas Multiagentes (SMA) formam uma sub-área da IAD, que concentra-se no estudo de agentes autônomos que realizam ações num dado ambiente.

Com o objetivo de estudar a forma como o comportamento dos agentes pode ser alterado dinamicamente, com o intuito destes adaptarem-se ao ambiente, este trabalho propôs a definição e implementação de um ambiente de simulação de sistemas multiagentes reativos, denominado Simula++, onde os agentes sofressem a intervenção de um algoritmo evolutivo, que modifique suas regras de comportamento.

O ambiente Simula++ possibilita ao usuário definir SMA reativos de duas formas: a primeira, já implementada em [FRO 97], utiliza um modelo de simulação onde os agentes necessitam que suas regras sejam pré-definidas antes do início da execução; a segunda forma, proposta por este trabalho, permite a modificação destas regras pré-definidas ou criar novas regras para o ambiente modelado, através dos preceitos da Vida Artificial e dos Algoritmos Evolutivos.

Para definir as características do modelo evolutivo proposto, foram estudados e analisados os conceitos da Vida Artificial, propostos por [LAN 84], e a teoria dos Algoritmos Genéticos e Evolutivos, que foram definidos em [HOL 75]. A Vida Artificial preocupa-se em fornecer subsídios para a construção de sistemas que sigam as mesmas leis biológicas naturais, onde a sobrevivência é o único critério de avaliação do indivíduo. Os Algoritmos Evolutivos podem ser definidos como modelos computacionais que utilizam os critérios evolucionários propostos por Darwin [DAR 87] como base para o desenvolvimento e implementação de sistemas de resolução de problemas computacionais. A partir do estudo destas duas teorias, foi proposto um algoritmo evolutivo para as regras comportamentais dos agentes, assim como um modelo de simulação baseado na teoria da Vida Artificial.

A validação do modelo envolveu o estudo de uma aplicação clássica dos SMA reativos, conhecida como o problema dos Robôs Mineradores.

Palavras-chave: Inteligência Artificial Distribuída, Simulação, Sistemas Multiagentes, Sistemas Multiagentes Reativos, Algoritmos Evolutivos, Vida Artificial.

Title: "An Evolutionary Behavior Environment for Reactive Multi-agent Systems"

Abstract

Distributed Artificial Intelligence (DAI) is based on the social behavior of individuals (human, animal or artificial), studying the actions and interactions among them. The Multi-agent Systems (MAS) are a sub-area of DAI, which focus on the study of autonomous agents and their actions in an environment.

As we have the goal of studying a model to dynamically change the agents' behavior in order to adapt the agents in their environment, this work intends to define and develop a simulation environment for Reactive Multi-agent Systems called Simula++, where an evolutionary algorithm can modify the set of behavior rules of each agent.

In the Simula++ environment, an user can define Reactive MAS in two ways: the first, already implemented in [FRO 97], uses a simulation model where the agents have their behavior rules defined before the beginning of the simulation; the second form, proposed by this work, allows the modify the predefined rule set to create new rules during the simulation, through the precepts of Artificial Life and Evolutionary Algorithms.

To define the characteristics of the proposed evolutionary model, the concepts of the Artificial Life, as proposed by [LAN 84], and the theory of the Genetic and Evolutionary Algorithms defined in [HOL 75] were studied and analyzed. Artificial Life supplies subsidies for the development of systems which follow the natural biological laws, where the survival is the only approach to an individual's evaluation. Evolutionary Algorithms can be defined as computational models which use the same evolutionary approaches proposed by Darwin as a base for the resolution of computational problems. From the study of these two theories, an evolutionary algorithm was proposed, which acts on sets of behavior rules, and a simulation model was developed, based on the theory of the Artificial Life.

The validation of the model involves the study of a classic application on Reactive MAS, known as the *Food Foraging Problem*.

Keywords: Distributed Artificial intelligence, Simulation, Multi-agent Systems, Reactive Multi-Agent Systems, Evolutionary Algorithms, Artificial Life.

1 Introdução

1.1 Contexto e motivações

A Inteligência Artificial Distribuída (IAD) tem como um de seus principais objetivos a construção de sistemas inteligentes formados por entidades autônomas (agentes), as quais são caracterizadas por sua própria competência e dirigidas por objetivos próprios. O modelo de inteligência utilizado em IAD é baseado no *comportamento social*, sendo a ênfase colocada em ações e interações entre estes agentes. A idéia de distribuir inteligência entre um conjunto de agentes é sustentada pelo fato de que sistemas grandes e complexos, constituídos de um único módulo, seriam difíceis de desenvolver e evoluir, além de trazerem problemas complexos em sua manutenção.

Os Sistemas Multiagentes (SMA) formam uma sub-área da IAD e concentram-se no estudo de agentes *autônomos* em um universo multiagente. Para os SMA, o termo autônomo designa o fato de que os agentes têm uma existência própria, independente da existência de outros agentes. Usualmente, cada agente possui um conjunto de capacidades comportamentais que definem sua competência, um conjunto de objetivos, e a autonomia necessária para utilizar suas capacidades comportamentais a fim de alcançar seus objetivos. Um agente é uma entidade computacional com um comportamento autônomo que lhe permite decidir suas próprias ações[ALV 97]. A decisão de qual ação levar a cabo é determinada pelo agente, tendo em consideração as mudanças acontecidas no ambiente em que atua e o desejo de alcançar seus objetivos. A idéia principal em um sistema multiagente é que um comportamento global inteligente pode ser alcançado a partir do comportamento individual dos agentes. Em um SMA não é necessário que cada agente seja individualmente inteligente para alcançar um comportamento global inteligente.

A utilização de sistemas multiagentes parte da premissa de que os agentes irão interagir com um mundo dinâmico e complexo. Como o ambiente em questão apresenta formas complexas e dinâmicas, torna-se interessante utilizar alguma forma de adaptação e/ou evolução nos comportamentos definidos para cada agente. A concepção de sistemas formados por entidades autônomas, que atuam socialmente para a resolução de determinados problemas, seria facilitada se tais entidades (agentes) apresentassem condições de se adaptar, aprender e evoluir. Para tanto, além da definição formal do ambiente e dos agentes que atuam sobre este, é necessário um mecanismo pelo qual os agentes tenham condições de se adaptarem dinamicamente ao ambiente em que atuam. Tal mecanismo facilitaria a reutilização de agentes entre ambientes distintos, além de promover uma melhora de performance nos sistemas desenvolvidos.

1.2 Vida Artificial e a Evolução de Comportamentos para um Sistema Multiagente Reativo – uma visão geral

Um SMA Reativo pode ser definido como um sistema formado por um conjunto de agentes simples, que percebem o ambiente à sua volta e atuam de acordo com esta percepção. Os agentes reativos não planejam e nem mantém uma memória de suas ações. Além disso, também não possuem uma representação do ambiente. O seu comportamento é definido por um conjunto de regras pré-definidas baseadas no par *estímulo-resposta*. A eficiência de um SMA Reativo é dependente dos comportamentos pré-definidos determinados para cada agente.

Os Algoritmos Evolutivos podem ser definidos como modelos computacionais que utilizam os critérios evolucionários propostos por Darwin [DAR 87] como base para o desenvolvimento e implementação de sistemas de resolução de problemas computacionais. Estes sistemas devem ser submetidos aos paradigmas propostos por Darwin: indivíduos variam de todas as formas viáveis nos ambientes em que ocupam e tal variação é hereditária; pelo resultado do embate pela sobrevivência, os indivíduos melhor adaptados irão gerar um maior número de descendentes na próxima geração.

O domínio da Vida Artificial preocupa-se exatamente em fornecer subsídios para a construção de sistemas que sigam as mesmas leis biológicas naturais, tais como a apresentada acima. Estes sistemas partem de uma população (conjunto) de agentes iniciais que, após um determinado período, morrem (ou seja, são eliminados do ambiente). Durante este período, os indivíduos melhor adaptados ao ambiente acabam por gerar descendentes, que possuem as principais características dos seus genitores. Para um SMA Reativo, estas características são representadas pelos comportamentos que definem a atuação do agente. Estas regras sofrem alterações, quando da geração de um novo agente para uma nova população. Através destas alterações, expande-se o espaço de busca do sistema, o que gera uma melhor adaptabilidade dos agentes ao ambiente.

A cada nova geração de agentes, novos comportamentos são gerados e testados dinamicamente, enquanto que outros são preservados. A preservação de um comportamento se dá pela atuação deste na sobrevivência deste indivíduo. Se o agente sobreviver por um longo período, ele gerará um maior número de descendentes, aumentando o número de indivíduos com características semelhantes a ele.

1.3 Objetivos e Estrutura da Dissertação

O objetivo central deste trabalho é especificar, implementar e avaliar um Ambiente de Simulação para Sistemas Multiagentes Reativos, denominado Simula++, em que o comportamento dos agentes pode se modificar durante a simulação. O ambiente foi baseado no sistema Simula [FRO 97], sendo composto por três elementos principais:

- ambiente de simulação: ambiente de definição, execução e simulação de sistemas multiagentes reativos;

- agentes reativos: definição e concepção dos agentes que atuarão no ambiente de simulação;
- algoritmo evolutivo: algoritmo que será utilizado para a evolução dos comportamentos dos agentes e do sistema como um todo.

Como objetivos secundários, podemos citar:

- estudos de caso, baseados na literatura, para o teste da validade e da performance do ambiente implementado;
- análise do comportamento de sistemas multiagentes reativos que apresentem características de aprendizagem e evolução.

No capítulo 2 são apresentados os fundamentos e o surgimento da Vida Artificial, concentrando-se nos modelos propostos para a criação de sistemas de Vida Artificial e seus diversos componentes. É feita uma breve introdução aos Algoritmos Evolutivos, que servirá como base para a descrição do sistema proposto. Ainda neste capítulo, é introduzida uma formalização [LAT 98] para a descrição e implementação de ambientes evolutivos.

Fundamentados nestes conceitos básicos, no capítulo 3 são investigados os SMA, dando uma maior ênfase aos SMA Reativos e na sua principal característica: a definição das regras de comportamento. Neste capítulo também é apresentado o sistema Simula, proposto em [FRO 97], que é a base para o ambiente desenvolvido nesta dissertação.

Após estas considerações, são apresentados no capítulo 4 as definições teóricas do ambiente Simula++, apresentando detalhadamente seus principais algoritmos e a forma como estes atuam frente ao objetivo maior do trabalho, que é a evolução de comportamentos para um SMA Reativo. O capítulo 5 apresenta o protótipo desenvolvido, concentrando-se nas suas principais características e no seu funcionamento.

No capítulo 6 é definido um caso de estudo, retirado da literatura, que visa à análise e validação do ambiente proposto. Finalmente, no último capítulo apresentamos as considerações finais sobre o trabalho, assim como as dificuldades encontradas e os possíveis caminhos para a continuação do mesmo.

2 Vida Artificial

2.1 Princípios Básicos

De acordo com seu fundador [LAN 89], a Vida Artificial é o campo de estudo devotado ao entendimento da vida através da abstração dos seus princípios dinâmicos fundamentais que formam a base dos fenômenos biológicos, e a recriar tais dinâmicas em outras mídias físicas, tais como os computadores, o que os tornaria acessível a novos tipos de manipulações experimentais e testes. Logo, podemos caracterizar a Vida Artificial como o estudo de sistemas construídos por seres humanos (artificiais) que exibem comportamentos característicos de sistemas vivos naturais (biológicos).

Ao estender as fundações empíricas sobre a biologia além da vida baseada em cadeias de carbono que envolvem toda a Terra, a Vida Artificial pode contribuir com a biologia teórica mudando da visão da *vida-como-nós-a-conhecemos* para uma visão maior da *vida-como-ela-poderia-ser*. O mais importante na definição de Langton é que a Vida Artificial deveria ver a vida como uma propriedade da organização da matéria, em vez de uma propriedade da matéria que foi organizada. Enquanto que a Biologia concentra-se principalmente com as matérias básicas da vida, a Vida Artificial está preocupada com as bases formais da vida. Ela inicia-se de baixo, vendo o organismo como uma grande população de máquinas simples, e trabalha sinteticamente a partir daí – construindo grandes agregados de objetos simples governados por regras, os quais interagem com os demais de uma forma não-linear, suportando dinâmicas globais, similares aos dos organismos vivos. O principal conceito em Vida Artificial é o comportamento emergente.

Segundo [STE 94] um comportamento do sistema é dito emergente se ele é definido utilizando categorias descritivas que não são necessárias para descrever o comportamento dos componentes constituintes do sistema. Em sistemas físicos, a temperatura e a pressão são exemplos de fenômenos emergentes – eles ocorrem em grandes conjuntos de moléculas, devido as interações a nível molecular; no entanto uma única molécula não possui as propriedades de pressão e temperatura. A natureza nos mostra dezenas de exemplos em que ações simples, de interação local entre seus componentes, acaba dando origem a comportamentos globais altamente organizados. Nestes exemplos incluem-se as colônias de insetos, montagem de células, a retina ocular e o sistema imunológico.

Logo, a Vida Artificial preocupa-se com a orquestração dos comportamentos destas máquinas simples de baixo nível, para que o comportamento emergente no nível global seja essencialmente o mesmo que os comportamentos exibidos pelos sistemas vivos naturais. Ou seja, a Vida Artificial concentra-se na geração de comportamentos parecidos com os dos sistemas naturais.

A metodologia apresentada por Langton é compatível com as noções de emergência: a partir de iterações não-lineares de componentes simples e mecânicos, pode-se observar a emergência de comportamentos complexos, imprevisíveis, em suma, parecidos com os comportamentos naturais. Organismos vivos naturais também são compostos por componentes não-vivos. O problema da biologia é precisamente a

emergência da vida a partir de componentes não-vivos. As substâncias destes componentes seguem, e são completamente descritas, pelas leis físicas; no entanto uma explicação física sobre a totalidade dos sistemas vivos é inexistente.

Similarmente, na Vida Artificial, tem-se componentes formais obedecendo um conjunto particular de axiomas, e a partir de suas interações, alguns comportamentos globais emergem, os quais não são totalmente explicados pelo simples somatório das regras formais locais. Analogamente, as regras formais atuam como leis físicas artificiais e o comportamento global, se for reconhecido como similar aos comportamentos naturais, atua como uma biologia artificial, numa aproximação *bottom-up* para os comportamentos complexos.

Obviamente, o principal axioma assumido pela Vida Artificial é que a forma lógica (comportamentos) de um organismo pode ser separada de seus materiais básicos dos quais esse é formado, e que a vida em si pode ser encarada como uma propriedade desta forma lógica, ao invés das substâncias que a compõem. A idéia aqui é que é possível descobrir os princípios básicos da organização de sistemas vivos, e as substâncias usadas para a criação da vida são irrelevantes. Ao investigar estes princípios básicos, estuda-se não apenas a vida biológica, baseada em cadeias de carbono (vida-como-nós-a-conhecemos), mas também as regras universais da vida (vida-como-ela-poderia-ser).

Apesar de ser, reconhecidamente, uma das bases da Vida Artificial, a abordagem de Langton apresenta alguns problemas. A observação de comportamentos emergentes em sistemas formais complexos, na busca por comportamentos interessantes, indica uma certa circularidade. Se a Vida Artificial concentra-se na busca de comportamentos parecidos com os naturais em sistemas artificiais, universais, as pesquisas apresentadas nos últimos anos vêm entrelaçando os conceitos da vida-como-ela-poderia-ser com a vida-como-nós-a-conhecemos através da análise de algumas semelhanças subjetivas. Esta abordagem dificilmente pode ser aceita como a busca por princípios universais.

O problema surge na análise dos comportamentos observados nos sistemas desenvolvidos. Esta análise, normalmente, é feita de forma subjetiva, baseada nas crenças pessoais de como a vida deve ser, como ela deve desenvolver-se, evoluir e, principalmente, como ela se apresenta diante de nossos olhos. Estas crenças são formadas pela nossa própria convivência com o único sistema biológico natural disponível para análise, que envolve todo o planeta Terra.

Mas mesmo se a própria Terra pudesse repetir seu processo evolutivo, seria altamente improvável que o sistema biológico global reaparecesse com suas formas atuais. O mesmo se aplica a qualquer outro planeta, ou sistema, eventualmente habitável: as chances de surgirem sistemas genéticos iguais aos nossos, com seleção das mesmas singularidades levando às mesmas combinações de genes são extremamente remotas.

Podemos perceber, então, dois objetivos diferentes para a Vida Artificial, de acordo com [ROC 97]. O primeiro, mais rígido, concentra-se na sintetização da Vida Artificial a partir de componentes computacionais ou materiais (robôs auto-situáveis, de acordo com os conceitos de [BRO 91]). O segundo está interessado em obter comportamentos semelhantes aos dos sistemas naturais, o qual ainda carece de uma

definição formal mais explícita. Para ser aceito como um campo científico, a Vida Artificial, além de imitar comportamentos subjetivos, deve concentrar-se na investigação das regras que nos permitem distinguir a vida dos elementos inanimados e que possam ser replicados experimentalmente em trabalhos científicos.

Segundo [ROC 97], não importa em qual dos dois objetivos está a meta, os artefatos e modelos a serem construídos devem sempre explicitar o conjunto de regras nos quais eles foram baseados, para que possam demonstrar porque tal organização artificial está viva ou observa alguns comportamentos parecidos com os dos sistemas vivos.

Segundo Claus Emmeche [EMM 94], existem sete pontos centrais que resumem a visão da Vida Artificial na sua forma mais ambiciosa. Estes pontos devem ser aplicados para a criação desta nova área de pesquisa e na realização técnico-científica dos novos tipos de sistemas desenvolvidos.

2.1.1 A biologia do possível

A Vida Artificial não se aplica à vida baseada em carbono tal como a conhecemos na Terra, que é o objeto de pesquisa da biologia experimental. O objetivo primordial da Vida Artificial é a vida-como-ela-poderia-ser. Como a biologia é baseada somente num exemplo, a vida na Terra, ela torna-se limitada na formulação de teorias gerais sobre a construção da vida.

2.1.2 Método sintético

Enquanto as pesquisas biológicas tradicionais têm sua ênfase na análise dos seres vivos, explicando-os em termos de suas partes menores, a perspectiva da Vida Artificial esforça-se na síntese de processos ou comportamentos parecidos com a vida natural, em outras mídias, tais como o computador.

2.1.3 Vida real

A Vida Artificial é o estudo de sistemas criados artificialmente por seres humanos que exibem características de sistemas vivos naturais. Os comportamentos apresentados devem ser tão genuínos quanto os comportamentos apresentados por organismos vivos. O termo artificial desta ciência refere-se aos seus componentes (computador, fórmulas, regras, etc.) que são construídas pelos pesquisadores. O comportamento, no entanto, deve ser produzido pelo sistema em si.

2.1.4 Toda vida é forma

Nenhuma forma de vida possível pode ser determinada pela matéria pela qual esta é constituída. A vida é um processo, e a forma como este processo ocorre (não a matéria) é a essência da vida. Logo, é possível ignorar a matéria e abstrair desta a lógica que governa o processo da vida. A vida é, fundamentalmente, independente da mídia.

2.1.5 Construção *bottom-up*

O melhor paradigma para a construção de sistemas de Vida Artificial é o princípio da programação *bottom-up*. A base do sistema é formada por uma coleção de inúmeras pequenas unidades que interagem entre si através de regras simples bem definidas. A partir destas interações, surge um comportamento global coerente num nível mais geral, que não havia sido previamente especificado pelas regras. Este estilo de programação contrasta com os princípios dominantes da Inteligência Artificial tradicional. Na IA, têm-se a construção de sistemas inteligentes a partir do paradigma *top-down*: o comportamento global esperado é programado pela divisão e subdivisão deste em inúmeras camadas, cada vez mais específicas. O método *bottom-up* da Vida Artificial imita e/ou simula os processos da natureza que se auto-regulam. Estes processos são denominados *simulação da auto-organização*.

2.1.6 Processamento paralelo

Enquanto que o processamento da informação em sistemas clássicos ocorre seqüencialmente, o princípio para o processamento da informação na Vida Artificial é baseado no grande paralelismo que ocorre na vida real (ex.: células nervosas do cérebro). As redes neurais artificiais são um exemplo típico do processamento de informações em paralelo, que podem ser utilizadas em sistemas de Vida Artificial.

2.1.7 Condução à emergência

A característica essencial da Vida Artificial é que esta não é pré projetada, o que ocorre, por exemplo, no desenvolvimento de sistemas para robôs autônomos. Os exemplos mais interessantes de sistemas de Vida Artificial exibem um comportamento emergente. Em termos computacionais, é a abordagem bottom-up que conduz à emergência de novos fenômenos nos níveis mais altos, o que é essencial nos sistemas vivos.

2.2 Domínios de Problemas para Vida Artificial

2.2.1 Aprendizagem

Há uma interconexão intensa entre os campos de redes neurais e a inteligência artificial. O aprendizado em sistemas multiagentes apresenta numerosos desafios para as teorias originalmente desenvolvidas para explicar a aprendizagem em indivíduos isolados.

2.2.2 Realidade virtual

A Realidade Virtual (RV) é um campo da ciência da computação com aplicações práticas em diversos campos, inclusive com a Vida Artificial. Tanto a RV como a Vida Artificial utilizam o computador para representar processos que imitam a realidade operando em mundos artificiais. No entanto, há uma diferença evidente de estilo entre os dois campos. Na RV o usuário é, necessariamente, um participante ativo nas

atividades do mundo artificial, enquanto que na Vida Artificial tal processo não pode ocorrer.

2.2.3 Agentes autônomos

Um agente autônomo é um programa que contém uma série de sensores e um sistema de execução. O agente opera em um ambiente, seja ele um sistema operacional, uma base de dados, uma rede de computadores ou um ambiente próprio de simulação. Os sensores são utilizados para observar os componentes do ambiente externo. O sistema de execução pode alterar o estado do ambiente ou o estado dos demais agentes. Cada agente possui determinados objetivos, individuais ou por grupo de agentes. Um dos principais objetos de pesquisa é o entendimento dos mecanismos pelos quais um organismo (real ou artificial) seleciona um comportamento a partir de uma variada coleção (normalmente mutuamente incompatíveis) para ser executada num determinado momento. O desenvolvimento de sistemas nesta área tem uma implicação direta na programação de robôs. Em respeito aos simuladores de Vida Artificial, a literatura sobre agentes autônomos é uma rica fonte de inspiração e metodologia.

2.2.4 Manufatura inteligente

Nesta abordagem cada máquina ou processo é controlada por um agente autônomo. O agente monitora o estado de sua máquina, tentando satisfazer suas necessidades em termos de matéria-prima, fluxo de entrada de materiais, etc., enquanto compete com os demais agentes por recursos.

2.2.5 Educação

A Inteligência Artificial tem sido extensivamente utilizada como base para programas educativos. Diversos simuladores de Vida Artificial têm sido desenvolvidos para o ensino dos princípios da biologia, especialmente para crianças.

2.3 Modelagem de Sistemas de Vida Artificial

Neste capítulo apresentaremos alguns dos principais modelos propostos para a construção de sistemas de Vida Artificial.

2.3.1 Movimento de animais artificiais

O primeiro modelo para o movimento de animais artificiais foi proposto por Craig Reynolds [REY 87]. Seu objetivo era produzir criaturas artificiais que apresentassem um comportamento semelhante a um bando de pássaros. A cada passo, cada pássaro virtual (*boid*) atualiza seu vetor velocidade e move-se ao longo desse vetor até a sua nova posição.

Um *boid* simples não possui nenhum estado interno e sempre reage da mesma maneira em situações iguais. Além disso, um único *boid* colocado num ambiente vazio não realizará nenhum comportamento complexo. Ele apenas voará numa determinada direção, em linha reta. Quando são colocados diversos *boids* num ambiente, o número

de situações possíveis torna-se extremamente grande, pois cada *boid* reage diferentemente de acordo com a posição e a velocidade de outros *boids*.

As três regras que compõem o vôo geométrico dos pássaros virtuais de Reynolds estão descritas a seguir:

- evitar colisões: cada *boid* deve manter-se numa distância de cruzeiro otimizada, uma espécie de zona de segurança, entre ele mesmo e seus companheiros nas cercanias de sua vizinhança. Se um outro *boid* estiver dentro da zona de segurança, há o perigo de colisão. Os *boids* evitam as colisões diminuindo a velocidade, se o vizinho que atravessou a zona de segurança estiver na sua dianteira, ou aumentando a velocidade, se este estiver na traseira. Assim como o boid não deve aproximar-se demais de outro boid, ele também não deve afastar-se demais do boid que estiver mais próximo;
- igualar direção: cada boid tenta voar paralelamente ao seu vizinho mais próximo. Esta regra é realizada pelo ajustamento da tangente do vetor posição para que este se aproxime da tangente do vetor posição do vizinho mais próximo. Esta regra não modifica a velocidade do boid;
- concentração do bando: cada boid tenta manter-se rodeado por outros boids por todos os lados. Isto é realizado calculando a posição média (*centroid*) em relação aos outros boids, e movendo-se para esta nova posição. Para realizar isso, o boid calcula o vetor que aponta para o centroid e gira a tangente do seu vetor posição para que os dois ângulos aproximem-se. Esta regra não modifica a velocidade do boid.

Um exemplo da atuação de um grupo de boids num ambiente com obstáculos é dado pela fig. 2.1.

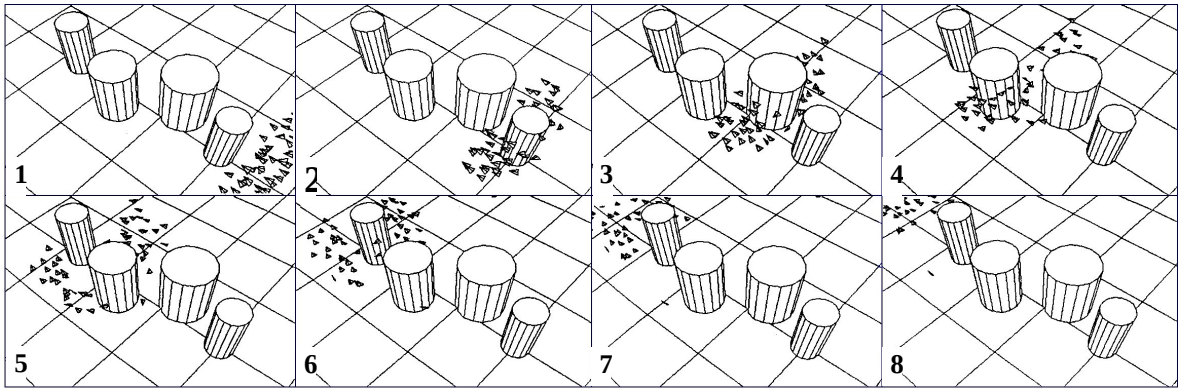


FIGURA 2.1 – Simulação de um Grupo de Boids (adapatado de [EMM 94])

2.3.2 Automata celulares

No final da década de 40, o matemático e físico John von Neumann [NEU 66], interessado em provar a possibilidade da auto-replicação em máquinas, iniciou uma investigação na lógica necessária para a replicação; ele não estava interessado em construir uma máquina em nível bioquímico ou genético, mesmo porque, naquela época, o DNA não havia sido descoberto ainda como o material genético da natureza.

Para conduzir uma investigação matemática formal, ele usou um modelo concebido pelo seu colega, o matemático Stanislaw Ulam. O modelo, conhecido como *autômato celular*, consistia numa grande grade de células, similar a um tabuleiro de xadrez, onde cada célula possuía um certo estado num dado momento. O número de possíveis estados por célula era finito e, usualmente, pequeno (normalmente cada estado era representado por uma cor diferente). Todas as células mudavam seu estado simultaneamente de tal modo que o estado da célula no próximo tempo dependia unicamente do estado corrente da célula e dos estados das células vizinhas. O princípio que guiava as transformações das células eram aplicados identicamente em todas as células, e este princípio era referenciado como a regra do sistema.

Uma máquina para o modelo de autômato celular era uma coleção de células que podiam ser consideradas como operando em harmonia. Desse modo, era possível observar *criaturas* simples que eram aptas a moverem-se mesmo neste universo simples.

As células estão em um de dois estados: viva ou morta. Cada célula tem oito possíveis vizinhos, formados pelas células ao lado e as do canto. Se a célula, num determinado passo, está viva, ela continuará viva no próximo passo se houver dois ou três vizinhos vivos. Ela irá morrer por superpopulação se houver mais de três vizinhos vivos, e irá morrer por exposição se houver menos de dois vizinhos vivos. Se uma célula estiver morta, ela permanecerá morta no próximo passo enquanto não houver exatamente três, dos seus oito vizinhos, vivos. Neste caso, a célula irá nascer no próximo passo [LEV 92], conforme podemos observar na fig. 2.2.

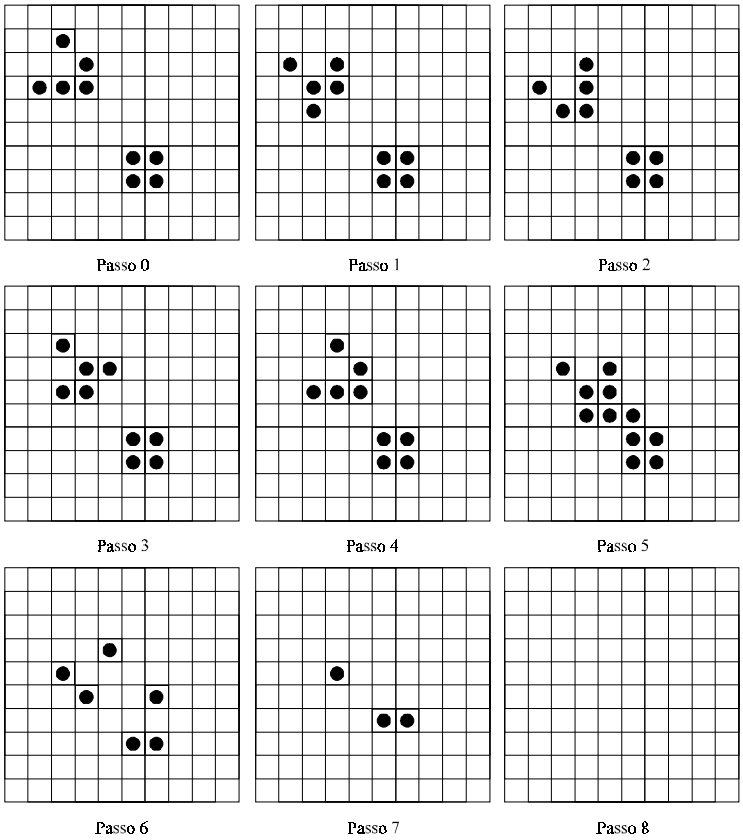


FIGURA 2.2 – O Jogo da Vida

No passo 0, havia dois modelos, um quadrado estacionário (*bloco*) e uma criatura movendo-se (*planador*). O planador move uma célula, diagonalmente, a cada quatro passos. Ao encontrar outras criaturas, como o bloco, as duas criaturas são sujeitas a uma simples regra universal (leis da física deste mundo), que causa uma aniquilação mútua, neste caso. O importante é notar que as ações, como o movimento do planador, estão puramente nos olhos do observador – o nível mais básico consiste de simples transformações de estado, sem nenhuma movimentação aparente. No entanto, considerando uma visão de mais alto nível do sistema, introduz-se novos termos para a descrição dos fenômenos observados. Os autômatos celulares exibem um comportamento emergente, ou seja, o aparecimento de capacidades de processamento de informações globais que não eram representadas explicitamente nos componentes elementares do sistemas, ou em suas interconexões.

Von Neumann utilizou este modelo simples para descrever uma máquina construtora universal, que poderia ler instruções de montagem de qualquer máquina dada, e construir a máquina de acordo. Estas instruções seriam coleções de células de vários estados, como deveriam ser as da nova máquina após a montagem – na verdade, qualquer elemento composto na grade era simplesmente uma coleção de células. O construtor universal poderia construir qualquer máquina, dado o *genoma* apropriado (instruções de montagem); ou seja, dada a sua própria descrição, ele era capaz de construir uma cópia de si mesmo (auto–replicação). Para que os descendentes também pudessem realizar a auto-replicação, era necessário copiar as instruções de montagem para estes. Desta maneira, von Neumann mostrou que processos replicativos eram possíveis em máquinas artificiais.

Uma das principais conclusões de von Neumann foi que o processo replicativo utilizava as instruções de montagem em duas maneiras distintas: como um código a ser interpretado (durante a montagem), e como dados não-interpretáveis (durante a cópia das instruções de montagem para seus descendentes). Durante as décadas que se seguiram, quando os mecanismos genéticos básicos começaram a ser desmembrados, ficou claro que o processo de von Neumann havia copiado o modelo genético. O processo pelo qual as instruções de montagem (DNA) são utilizados para construir máquinas operadoras (proteínas) também utiliza uma forma dual de informação: como um código interpretado e como dados não-interpretáveis. O primeiro é conhecido em biologia como tradução, o segundo como transcrição e ocorre durante a duplicação celular.

Um dos principais problemas com o autômato celular de von Neumann é a complexidade do construtor, que requer centenas de milhares de células. Além disso, cada célula pode estar em um de vinte e nove possíveis estados, ao invés de apenas dois, como exemplificado acima. Mais tarde, Christopher Langton [LAN 84] observou que, embora a capacidade de construção universal seja uma condição suficiente para a auto-replicação, ela não era necessária; mesmo porque, os sistemas naturais não eram capazes de realizar uma construção universal. Langton desenvolveu autômatos auto-replicáveis que eram muito mais simples que o construtor universal de von Neumann. Para estas máquinas, no entanto, não havia qualquer capacidade de construção ou computação, sua única funcionalidade era a auto-replicação.

2.3.3 Vírus

O termo vírus de computador é derivado e análogo ao vírus biológico. As infecções biológicas expandem-se quando o vírus (um pequeno casulo contendo material genético) consegue injetar-se numa grande célula do corpo. A célula infectada é convertida numa fábrica biológica para a replicação do próprio vírus.

Similarmente, um vírus de computador é um segmento de código de máquina (tipicamente entre 200 e 4000 bytes) que replica seu código, quando ativado, em um ou mais programas hospedeiros. Quando os programas infectados são executados, o código viral também é executado, o que acaba expandindo a infecção.

Um vírus possui três estágios de 'vida':

- execução: para que um vírus se espalhe, ele precisa ser executado. Esta fase ocorre diretamente quando um usuário invoca um programa infectado, ou indiretamente, quando o sistema executa o código viral como parte de sua seqüência de inicialização ou ao executar uma tarefa administrativa em segundo plano;
- replicação: o vírus, ao ser executado, replica-se para um ou mais programas escolhidos aleatoriamente;
- realização de tarefas: além de auto-replicar-se, os vírus podem realizar outras tarefas paralelamente, que apresentam uma variedade de efeitos (benéficos ou malévolos) que acabam por indicar a presença do vírus.

Algumas considerações podem ser feitas em relação às semelhanças entre os vírus e a Vida Artificial [SPA 91]:

- o vírus como um molde no espaço-tempo: os vírus são representados como moldes de instruções que existem ao mesmo tempo em muitos sistemas computadorizados. Os vírus não são associados com o maquinário físico, mas com as instruções executadas por este hardware;
- auto-replicação: os vírus de computador possuem a habilidade de reproduzir-se, gerando uma cópia exata ou mutante de seu código genético, o que constitui um dos princípios básicos da vida;
- armazenamento da informação de sua própria representação: assim como os vírus biológicos, que utilizam o DNA para armazenar toda a informação necessária para a replicação da célula, os vírus de computador utilizam o seu código como um gabarito ao auto-replicar-se;
- metabolismo virótico: esta propriedade envolve a energia ou a matéria que um organismo vivo necessita transformar para executar as suas atividades. Os vírus de computador utilizam a energia elétrica do sistema para executar as suas atividades. Eles não convertem matéria, mas utilizam esta energia para executar as suas instruções e infectar outros programas;
- interações funcionais com o ambiente: os vírus executam inspeções em seus sistemas hospedeiros como parte de sua atividade. Eles examinam as arquiteturas de memória e do disco rígido, alterando os endereços para que possam ocultar a si mesmos, ou seja, eles alteram seu ambiente para suportar a sua existência;

- interdependência entre todas as partes do vírus: assim como os vírus biológicos, se um vírus de computador for dividido, ele será destruído e cessará suas atividades normais;
- estabilidade perante perturbações: alguns vírus de computador podem rodar numa grande variedade de máquinas, sob diferentes sistemas operacionais. Muitos vírus podem constranger e derrotar programas antivírus;
- evolução: esta é a maior diferença entre os vírus biológicos e os vírus de computador, pois o segundo não apresenta evolução. Apesar de ser possível construir um vírus capaz de evoluir de uma forma contínua, este seria tão grande e complexo, que dificilmente deixaria de ser detectado e eliminado. No entanto os vírus de computador apresentam mutações. Normalmente estas mutações variam poucas instruções, principalmente as que imprimem mensagens, ou os tipos de formas de ativação e replicação;
- crescimento: o crescimento do vírus pode ser medido pelo número de arquivos infectados no sistema. Certos vírus podem invadir todos os arquivos em algumas poucas ativações;
- outros comportamentos: os vírus de computador agrupam-se em espécies com nichos ecológicos bem definidos, baseados no tipo de máquina hospedeira e no seu sistema operacional. Estas espécies são adaptadas para um ambiente específico e não sobrevivem se movidas para um ambiente diferente. Além disso, alguns vírus exibem comportamentos predatórios. Como exemplo, o vírus DenZuk procura e replica-se através das instâncias do vírus Brain, se os dois estiverem presentes no mesmo sistema. Outros vírus apresentam um comportamento territorial, marcando seus domínios territoriais para que outros vírus do mesmo tipo não possam entrar em sua área, competindo com o vírus original.

2.3.4 Minhocas

Uma minhoca de computador (*worm*) é um programa de computador que, parecido com os vírus, pode auto replicar-se. A diferença é que um *worm* não expande-se infectando programas executáveis hospedeiros. Ou seja, ao contrário de um vírus, um *worm* contamina a memória, a ponto de impedir o uso desta pelos outros programas. Alguns *worms* mandam cópias de si mesmo para outros usuários utilizando o correio eletrônico.

Os *worms* rodam independentemente e viajam de máquina para máquina através das conexões de rede; os *worms* podem ter porções espalhadas rodando em diversas máquinas e, usualmente, não modificam outros programas. Apesar disso, eles podem carregar outros códigos que infectam o sistema (vírus).

Poucos *worms* foram construídos até hoje, principalmente os que executam funções danosas, pois eles são difíceis de construir. Os *worms* necessitam um ambiente de rede e um autor que esteja familiarizado com os serviços e facilidades da rede, assim como as operações necessárias para suportar o seu transporte de uma máquina para outra. O *worm* mais conhecido foi criado por um estudante da Cornell University, e disparado em novembro de 1988 na Internet (*Internet Worm*). Este *worm* afetou pelo menos 6000 sistemas, reduzindo instantaneamente a performance dos mesmos. Como o *worm* só ataca a memória do sistema, a sua eliminação é relativamente simples: reinicializar o sistema.

2.3.5 Mundos virtuais

A construção de modelos de Vida Artificial utilizando mundos virtuais, usualmente bidimensionais, é a forma mais difundida e utilizada no estudo científico da Vida Artificial. As simulações computadorizadas de Vida Artificial são feitas construindo-se um mundo virtual no qual agentes (ver capítulo 3.1) podem mover-se, competir e evoluir.

Um agente neste mundo virtual contém, normalmente, um número pessoal de identificação, sua posição atual, e alguns valores que são utilizados para a comparação e definição dos comportamentos a serem seguidos. Estes comportamentos podem ser representados por tabelas, algoritmos genéticos e/ou redes neurais. Além disso, o agente pode manter dados adicionais sobre o seu estado, sua velocidade, os membros de sua colônia, o tipo de identificação de suas presas e predadores.

O mundo virtual, que é compartilhado por todos os agentes, é formado por uma grade de células, parecida com um tabuleiro de xadrez. Cada agente habita uma única célula de cada vez. O mundo apresenta um marcador ou uma trilha individual para cada agente, que representa as mudanças de posição do agente. Além disso, o mundo mantém também algumas marcas que os agentes podem interpretar como comida, veneno, muros, caminhos, etc., tal como observado na fig. 2.3.

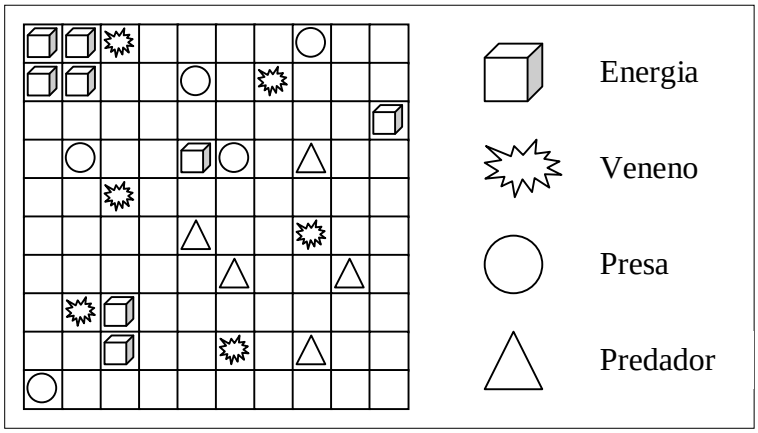


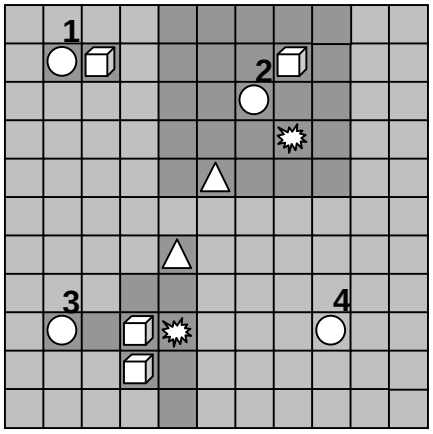
FIGURA 2.3 – Mundo Virtual

Os agentes trabalham dentro de um laço de execução, usualmente definido como o que segue:

- receba e analise as informações do mundo;
- calcule a saída, a nova posição e os novos valores para os marcadores internos;
- repasse a saída para o mundo.

Os agentes podem receber vários tipos de informações do mundo, dependo do poder computacional da máquina e das variáveis envolvidas e desejadas pelo pesquisador. Em termos de processamento, o modo mais vantajoso é receber informações apenas da célula para a qual o agente está movendo-se. Alternativamente, pode-se construir um agente que receba informações de todos os seus vizinhos.

Outros modos incluem a concepção de visão, onde o agente recebe e calcula a distância e a direção dos outros agentes que estejam dentro do seu campo de visão. Finalizando, um agente ainda pode receber informações sobre todos os agentes do mundo, com a sua localização e tipo. Este tipo de informação, além de apresentar o maior custo computacional, não apresenta nenhuma similaridade com os modelos biológicos existentes. Os quatro principais modos de percepção de um agente podem ser visualizados na fig. 2.4:



1 – Informações sobre a célula na direção do seu movimento
2 – Informações sobre as células vizinhas, num raio pré-determinado
3 – Informações sobre as células contidas num campo de visão limitado
4 – Informações sobre todas as células do mundo virtual

FIGURA 2.4 – Modos de Percepção em um Mundo Virtual

A partir da percepção do mundo que os cerca, os agentes podem tomar suas decisões. O processo de tomada de decisões é o mais importante e, potencialmente, o mais difícil de ser definido e executado. É a partir da definição destes comportamentos básicos que o sistema será posto a prova quanto à emergência de uma inteligência, normalmente não explicitada.

2.4 Algoritmos Evolutivos

A evolução biológica pode ser definida como a mudança progressiva no material genético de uma população por muitas gerações. Em seu livro, Charles Darwin [DAR 87] apresenta o que viria a ser o núcleo de toda a teoria evolucionária atual, cujos principais conceitos podem ser resumidos em quatro pontos principais:

- indivíduos variam de todas as formas viáveis nos ambiente em que ocupam;
- a variação é hereditária;
- a auto-replicação de indivíduos tende a produzir mais descendentes que possam sobreviver com os recursos limitados disponíveis no ambiente;
- pelo resultado do embate pela sobrevivência, os indivíduos melhor adaptados irão gerar um maior número de descendentes na próxima geração.

Os dois principais conceitos apresentados acima são a seleção natural e a variação genética. A seleção natural é o processo pelo qual os indivíduos mais aptos produzem mais descendentes, em média, em gerações sucessivas, do que os indivíduos menos aptos. A variabilidade genética resulta dos eventos aleatórios que podem ocorrer aos indivíduos de uma determinada população. Estes eventos incluem a mutação

aleatória do material genético, a morte acidental de um indivíduo com alto grau de adaptabilidade, antes que esse tivesse a oportunidade de reproduzir-se, etc.

O gene é a unidade da hereditariedade. Cada gene, agindo sozinho ou com outros genes, determina uma ou mais características do organismo resultante. Todas as formas de vida conhecidas são baseadas em códigos genéticos. Mesmo os organismos unicelulares são construídos a partir do material genético do núcleo celular. Um conjunto completo de genes é conhecido como genótipo.

Os genes são encontrados no interior do material genético denominado cromossomo. Na maioria das células, cada gene ocupa uma posição em particular num cromossomo específico. Os cromossomos podem quebrar-se, durante a reprodução celular, e parte de seus genes são transferidos de um lugar para outro, dentro do mesmo cromossomo, ou para outros cromossomos (reprodução sexuada). Quando isso acontece, novas combinações dos genes são formadas.

Os genes também podem mudar na sua composição química. Neste tipo de recombinação alterada ou variação química, eles acabam por produzir diferentes elementos a partir de genes inalterados. Esse processo é conhecido por mutação e tem um papel decisivo na evolução das espécies. Muitos fatores ambientais podem acabar alterando a estrutura da molécula de DNA (material genético). Alguns fatores podem ser físicos, e outros são químicos. Uma mutação ocorre quando estas alterações acabam por provocar mudanças permanentes na seqüência básica do DNA, que provocam uma mudança hereditária da síntese de proteínas.

Ou seja, o fenótipo de um indivíduo é o conjunto de comportamentos deste, em relação ao ambiente em que vive. Comportamentos estes definidos pelo código genético (genótipo) e pela capacidade de percepção e aprendizado do indivíduo frente ao mundo que o cerca. Essa definição é de suma importância em Vida Artificial, pois é a partir dos comportamentos dos indivíduos que a ciência da Vida Artificial busca a compreensão da vida e da sua evolução.

A seleção natural favorece determinados genótipos. O sucesso reprodutivo de determinados genótipos é determinado pela totalidade de traços e qualidades produzidas num dado ambiente. A aptidão de um genótipo em particular é determinada pelos eventos em todos os estágios do seu ciclo de vida. O elemento mais óbvio da aptidão é a viabilidade, ou seja, a habilidade de um organismo desenvolver-se e sobreviver desde o nascimento até a fase adulta. Os efeitos combinados de todos estes componentes determinam a aptidão de um dado genótipo. É preciso ressaltar que um indivíduo, na reprodução sexuada, mesmo que viável e saudável na fase adulta, terá um grau de aptidão zero se falhar em encontrar um parceiro compatível durante a seleção sexual.

Em termos mais genéricos, o processo evolucionário ocorre quando da satisfação destas quatro pré-condições [KOZ 98]:

- uma entidade (x) possui a habilidade de reproduzir-se;
- existe uma população destas entidades capazes de se reproduzirem (P(x));
- há uma variedade nesta população de entidades;
- a diferença para a sobrevivência no ambiente é associada com a variedade.

A variedade é manifestada como as diferentes formas que o cromossomo pode assumir em uma determinada população. Esta variação é traduzida no comportamento das entidades em seu ambiente. A diferença de comportamentos refletem nas taxas de sobrevivência e reprodução das entidades. As entidades com uma maior capacidade de adaptação ao ambiente possuem maiores taxas de sobrevivência e reprodução.

Em seu livro, John Holland [HOL 75] desenvolveu um algoritmo genérico que buscava abranger todos os sistemas adaptativos (naturais ou artificiais), mostrando como os processos evolucionários podem ser aplicados em sistemas artificiais. Para Holland, qualquer sistema adaptativo poderia ser formulado utilizando termos genéticos, através de seu algoritmo (Algoritmos Genéticos - A.G.).

Os Algoritmos Genéticos são algoritmos matemáticos, altamente paralelizáveis, que transformam um conjunto (população) de dados individuais (tipicamente representado por um vetor de caracteres de tamanho fixo), onde cada indivíduo é associado a um valor que mede a sua adaptabilidade ao ambiente (*fitness*), em uma nova população, utilizando operações (recombinação sexuada) de acordo com os princípios das taxas de sobrevivência e reprodução apresentadas por Darwin.

Existem quatro passos na formalização de um problema utilizando Algoritmos Genéticos:

- determinação do esquema de representação: cada indivíduo de uma população representa uma possível solução do problema. O esquema de representação destas soluções (usualmente, um vetor de caracteres) é dependente do tipo de problema analisado;
- determinação da medida do grau de adaptabilidade: a medida do grau de adaptabilidade de cada indivíduo (*fitness*) deve ser baseada em critérios bem definidos. Estes critérios também são dependentes do problema analisado;
- determinação dos parâmetros e variáveis que controlam o algoritmo: dentre estes, os mais importantes são a percentagem de mutação (alterações randômicas no cromossomo) e a taxa de reprodução;
- determinação do critério de parada do algoritmo: este parâmetro controla a execução do algoritmo, sendo dependente do problema analisado.

Analisando os passos apresentados acima, percebe-se que a utilização de Algoritmos Genéticos (A.G.), apesar de ser um modelo genérico para a resolução de sistemas adaptativos, é extremamente dependente do problema analisado. Um fluxograma genérico para a execução de um AG é apresentado na fig. 2.5.

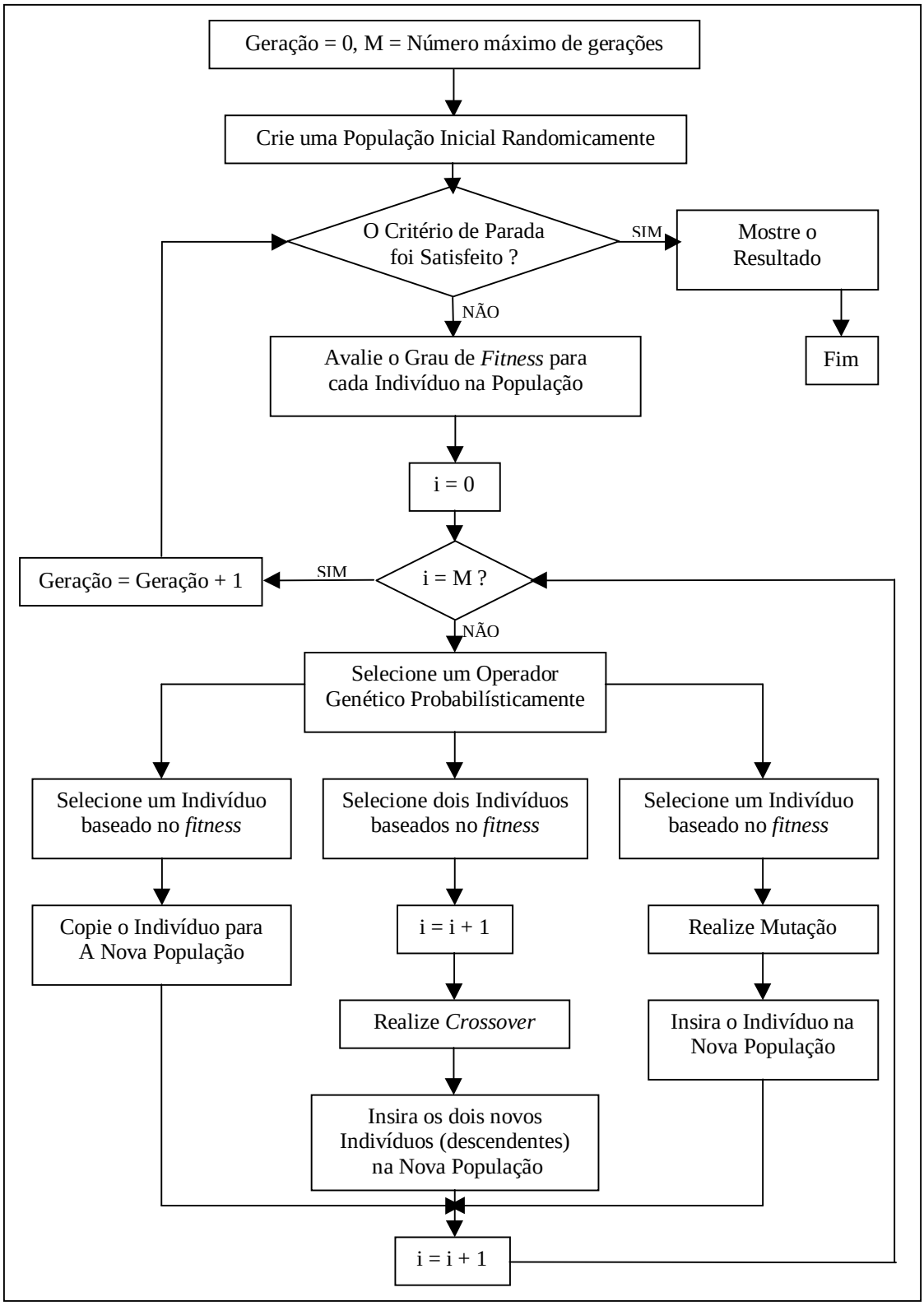


FIGURA 2.5 – Algoritmo Genético [KOZ 98]

Os Algoritmos Evolutivos (AE) surgiram como uma extensão, mais generalista, dos Algoritmos Genéticos. Os AE usam modelos computacionais de processamento evolucionário como base principal para o desenvolvimento e implementação de sistemas de resolução de problemas computacionais. Uma grande variedade de modelos computacionais evolutivos têm sido apresentados. Todos eles apresentam uma base

conceitual comum para a simulação da evolução das estruturas dos indivíduos através de processos de seleção, mutação e reprodução.

Mais precisamente, os Algoritmos Evolutivos mantém uma população de estruturas (agentes) e as regras para a seleção e reprodução, que são referenciados como Operadores Genéticos (ex.: recombinação e mutação). Para cada indivíduo na população, é possível calcular sua adaptabilidade no ambiente (*fitness*). A reprodução[1] ocorre tendo em vista os indivíduos com maior grau de aptidão. A recombinação e mutação modificam estes indivíduos, que serão novamente avaliados no próximo passo. Apesar de ser extremamente simplista, do ponto de vista biológico, estes algoritmos são suficientemente complexos para prover mecanismos de busca robustos e extremamente adaptativos.

Para ilustrarmos o conceito de aptidão de um indivíduo e suas conseqüências para a seleção natural, partiremos do estudo de [WRI 32] que apresenta a idéia da *paisagem adaptativa* ou *superfície de aptidão*, que é um meio útil de visualizarmos como a seleção atua numa população que está evoluindo. A idéia aqui é traçar a o grau de *fitness* como uma função sob o espaço de todas as combinações genéticas possíveis para uma espécie em um ambiente em particular, como visto na fig. 2.6.

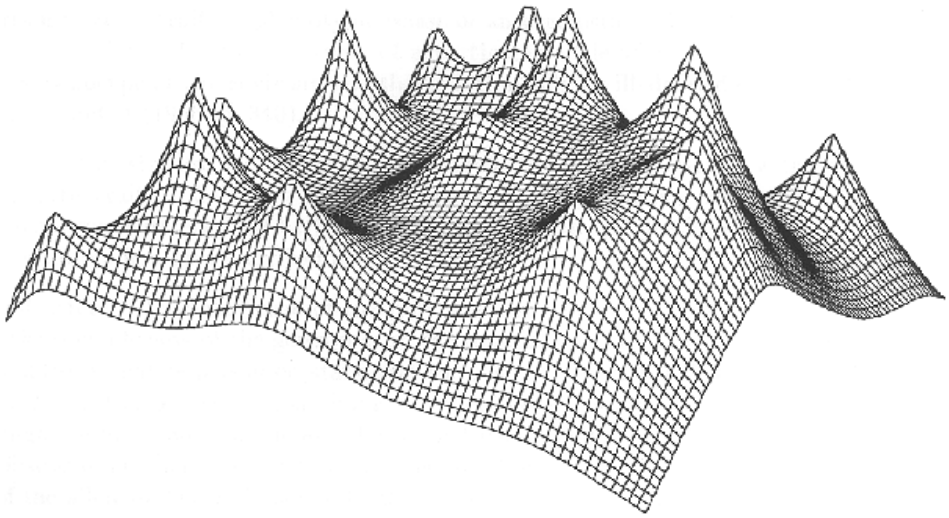


FIGURA 2.6 – Superfície de Aptidão num Espaço Bidimensional
A altura da superfície é dada pelo grau de aptidão para cada
combinação de genes num ambiente específico.

O gráfico resultante forma a superfície de aptidão. O espaço de possíveis organismos têm milhares de dimensões, e as superfícies de aptidão tem tipicamente um grande número de picos e vales, variando na altitude, pois algumas combinações dos genes são mais adaptativas do que outras. A população é representada como uma nuvem de pontos na superfície de aptidão, um ponto para cada organismo na população. Quanto mais variável for a população, mais esparramada será a representação desta no gráfico.

---

[1] No modelo de AE adotado, a reprodução ocorre sempre de forma sexuada, ou seja, há um embaralhamento (recombinação) do material genético (cromossomo) na formação de um novo descendente.

Os descendentes dos indivíduos mais aptos em uma população tendem a ser mais numerosos. Logo, após muitas gerações, a seleção natural tende a mover a população para os picos (aptidão elevada), enquanto que o caráter aleatório das mutações tende a mover a população de forma esparsa na superfície do gráfico. Se a seleção natural dominar a evolução da população, a tendência é um aumento gradual no gradiente da superfície de aptidão; se a aleatoriedade dominar, a evolução tende a desviar-se por toda a superfície.

Muitos programas evolutivos podem ser formulados para um dado problema. Estes programas diferem em vários modos: eles podem usar diferentes estruturas de dados, operadores genéticos, parâmetros (tamanho da população, probabilidades de aplicação dos operadores genéticos, operadores, etc.). No entanto, o princípio básico permanece: uma população de indivíduos sofre diversas transformações, e durante este processo, os indivíduos tentam sobreviver. Um algoritmo genérico para os AE pode ser descrito como apresentado na fig. 2.7:

1. Inicialize o algoritmo (t=0)
2. Inicialize uma população randômica de indivíduos
3. Calcule o grau de aptidão (*fitness*) para todos os indivíduos da população (avalie P(t))
4. Enquanto uma condição de parada não for satisfeita
   - Incremente o contador (t = t + 1)
   - Selecione uma sub-população para reprodução (P'(t) = select P(t))
   - Recombine os genes desta sub-população (recombine P'(t))
   - Perturbe a população (mutação P'(t))
   - Recalcule o novo grau de aptidão (avalie P'(t))
   - Selecione os sobreviventes atuais (P(t+1) = sobrev P(t) + P'(t))

FIGURA 2.7 – Algoritmo Evolutivo [KOZ 98]

Como visto antes, os Algoritmos Evolutivos surgiram baseados nos Algoritmos Genéticos [HOL 75], tentando resolver seus principais problemas. Basicamente, os AE são AG que apresentam, incorporados, conhecimentos do mundo real para um dado problema, tentando representá-los da forma mais "natural" possível. Existem dois principais aspectos que diferem Algoritmos Genéticos de Algoritmos Evolutivos:

- não há restrições na representação: os Algoritmos Genéticos clássicos envolvem uma codificação das soluções dos problemas como uma cadeia de caracteres (normalmente para representação binária), conhecida como genótipo. Em Algoritmos Evolutivos, a representação é baseada no problema. Uma rede neural pode ser representada da mesma maneira como é implementada, por exemplo, pois as operações de mutação não necessitam uma codificação linear;

- mutações: as operações de mutação simplesmente modificam aspectos da solução de acordo com uma distribuição estatística que gera pequenas variações no comportamento dos descendentes, não provocando grandes alterações. Além disso, a severidade das mutações pode ser reduzida na mesma proporção em que aumenta a probabilidade de encontrar-se um ótimo global. Esta característica pode ser alcançada através da diminuição do parâmetro que representa a

porcentagem de genes alterados a cada nova geração, de acordo com a qualidade da solução encontrada nas últimas gerações. Se a melhor solução não apresentou melhorias nas últimas *k* gerações, pode-se dizer que o A.E. encontrou um máximo ou mínimo, local ou global.

2.5 Sistemas de Mundos Virtuais

Segundo [RAY 94], existe uma distinção entre simulação e instanciação de sistemas de Vida Artificial. Na simulação, são modeladas populações de estruturas de dados em um programa de computador que é utilizado para representar populações de entidades biológicas reais (predador e presa, formigas, células animais ou vegetais, etc.). A grande maioria dos sistemas desenvolvidos até o presente momento utilizam esta abordagem.

Em uma instanciação de um sistema de Vida Artificial, as populações de estruturas de dados não representam explicitamente nenhum organismo ou processo vivo, apesar de obedecerem as mesmas regras abstraídas das leis que governam sistemas naturais.

Apesar de largamente utilizado, a formalização de ambientes de evolução (Mundos Virtuais) é um campo ainda em aberto, no que se refere a sistemas que utilizam os paradigmas da Vida Artificial. O modelo apresentado nesta seção foi proposto por Claude Lattaud e Cristina Cuenca [LAT 98], que visavam definir um modelo para a construção de ambientes evolutivos que integrassem todos os elementos componentes destes, provendo um *framework* para a construção de ambientes dinâmicos. O modelo define cinco classes para a construção de ambientes evolutivos. Duas destas classes são abstratas e as demais são concretas. A fig. 2.8 mostra a hierarquia entre as classes.

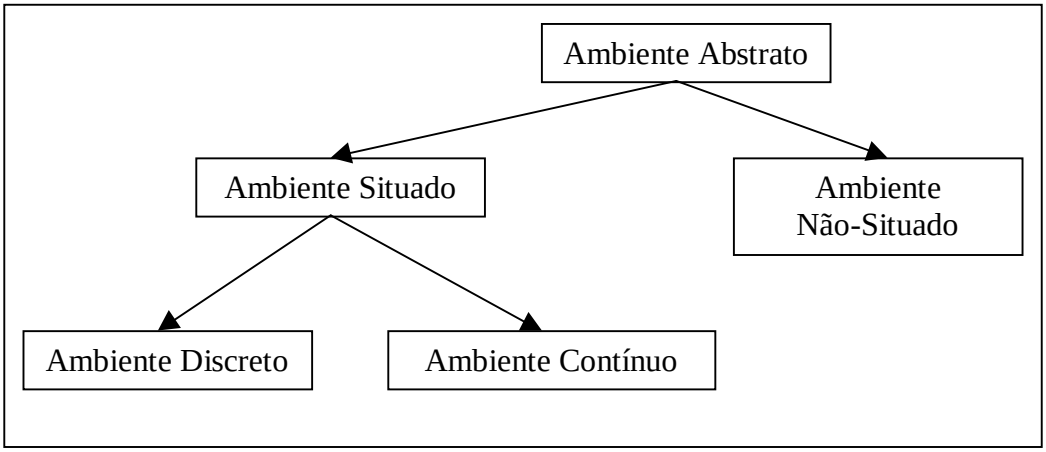


FIGURA 2.8 – Modelo para Construção de Mundos Virtuais

A classe *Ambiente Abstrato* é raiz de toda a hierarquia. Ela possui diversos parâmetros, que são comuns a todas as suas subclasses:

- tamanho do ambiente (S): este parâmetro determina se o ambiente é finito ou infinito. No primeiro caso, S conterá um valor positivo, cuja unidade depende do tipo de ambiente desenvolvido;
- conjunto de objetos do ambiente (O): divididos, usualmente, em três partes, objetos ativos (agentes), objetos inativos e sub-ambientes;
- conjunto de leis de interação (I): conjunto de regras que regem as interações entre agentes ↔ objetos, agentes ↔ agentes, ambiente ↔ objetos e ambiente ↔ agentes;
- conjunto de leis de evolução (E): determina as regras de evolução do ambiente, além das funções de transição que determinam a passagem de um estágio para outro.

A classe *Ambiente Situado* é utilizada geralmente em simulações de sistemas multiagentes e para a resolução de diversos problemas em Vida Artificial. Um determinado sistema é dito *Situado* se ele for construído num espaço delimitado, numa métrica conhecida, e se seus objetos ocuparem uma posição relativa neste espaço. Os parâmetros para esta classe são definidos a seguir:

- forma (F): definição das diferentes estruturas existentes (círculos, esferas, quadros, cilindros, etc.);
- dimensão (D): 1 a n dimensões;
- métrica (M): define a unidade de medida e a escala para a definição das coordenadas dos objetos do ambiente;
- conjunto de propriedades de gerência do espaço (P): este conjunto é formado por definições para a visualização do ambiente, tais como propriedades de projeção, de rotação, etc.

A classe *Ambiente Não-Situado(N)* pode ser utilizada para definir um ambiente em sistemas baseados em regras. Um ambiente é dito *Não-Situado* se os conceitos de espaço e posição relativa não forem aplicados ao sistema. Esta classe não possui nenhum parâmetro específico.

Um ambiente situado é definido como discreto (classe *Ambiente Discreto(D)*) se ele pode ser divido em um número finito de partes elementares, denominadas células, onde cada célula possui um número fixo de vizinhos. Os parâmetros para este tipo de ambiente são:

- estrutura das células (L): forma da estrutura da célula (quadrado, hexágono ou triangular, para ambientes bidimensionais);
- métrica inteira para o ambiente (Mi): o ambiente pode ser dividido em partes elementares. A computação das coordenadas e os movimentos são realizados no conjunto inteiro, ou em algum de seus subconjuntos.

A classe *Ambiente Contínuo (C)* é uma classe complementar à classe D. Um ambiente situado é denominado contínuo se, para quaisquer dois pontos, há pelo menos um ponto entre eles. Um ambiente contínuo pode ser construído utilizando uma métrica real, baseado em sistemas vetoriais reais. Este tipo de métrica é o único parâmetro para esta classe.

Segundo [LAT 98], a partir desta estrutura hierárquica, pode-se definir um meta-ambiente como sendo um tipo especializado de um *Ambiente Abstrato*, que possui as seguintes propriedades:

- tamanho infinito;
- conjunto de objetos, exclusivamente ambientes;
- conjunto de leis para as interações entre os sub-ambientes;
- conjunto de regras internas para a evolução.

Os meta-ambientes possuem as seguintes características, no que concerne ao seu funcionamento interno:

- dependência entre ambientes: um ambiente é completamente dependente dos demais e só pode realizar suas ações de acordo com o meta-ambiente;
- comunicação: um ambiente pode estabelecer uma comunicação com outros ambientes para troca ou transferência de informações;
- transferência: um ambiente pode transferir agentes, objetos e/ou regras para outros ambientes;
- união: dois ambientes, que possuam elementos idênticos, podem reunir-se num ambiente único, composto pelo conjunto de todos elementos pertencentes aos dois ambientes originais.

Como qualquer outro ambiente, um meta-ambiente pode evoluir, passando por diferentes estados. O meta-ambiente possui um estado *Inicial*, quando este for criado, um conjunto de estados *Transitórios*, durante seu período de vida e um estado *Final*, após a sua finalização. Todos estes estados são relacionados através de funções de transição. O conceito de evolução é importante para o meta-ambiente, que necessita reconhecer a situação de cada sub-ambiente para realizar as mudanças necessárias, tal como a criação ou a destruição destes.

Existem quatro tipos básicos de evoluções existentes para um ambiente (ou meta-ambiente):

- evolução estrutural: modificação na estrutura do ambiente. Por exemplo, para um ambiente discreto bidimensional, definido por uma grade de 15x15, após uma evolução estrutural a grade poderia aumentar para 16x16 ou diminuir para 14x14;
- evolução comportamental: as leis internas do ambiente são modificadas. Por exemplo, uma lei que controla a replicação de recursos renováveis pode ser alterada para incrementar ou diminuir a competição por este recurso específico;
- evolução de componentes: o ambiente cria ou destrói classes de objetos. Novos tipos de objetos ou agentes podem ser criados, assim como todos os objetos de um certo tipo de objeto pode ser eliminado completamente do ambiente;
- evolução geográfica: o ambiente cria ou destrói instâncias de objetos. Após a realização de uma evolução geográfica, novos objetos, obstáculos, recursos e agentes podem ser criados ou destruídos em posições específicas do ambiente.

3 Sistemas Multiagentes

3.1 Introdução

A metáfora de inteligência utilizada pelos Sistemas Multiagentes é a *comunidade inteligente*, ou seja, o comportamento social que é base para a inteligência do sistema. A metáfora utilizada pela IA clássica é basicamente de origem psicológica, enquanto que aquela utilizada pela IAD pode ser de natureza sociológica ou etológica.

Um abordagem sociológica/etológica é interessante quando se deseja resolver problemas complexos, que requeiram conhecimento de vários domínios e que podem envolver dados fisicamente distribuídos. Segundo [DEM 95], podemos decompor um sistema segundo uma metodologia de IAD, através dos conceitos definidos a seguir.

Dado um determinado sistema, denomina-se *agente* cada uma de suas entidades ditas ativas. Este conjunto de agentes forma uma *sociedade*. As entidades passivas serão designadas pelo termo *ambiente*. Um agente recebe informações e raciocina sobre o ambiente, sobre outros agentes e decide quais ações deve realizar e quais objetivos deve seguir. Um agente é uma entidade ativa, ou seja, capaz de controlar suas ações, diferentemente das noções estáticas tais como módulos, conjunto de regras e bases de conhecimentos.

Denomina-se *interação* entre agentes ou entre agente/ambiente uma troca de informações, que pode ser realizada de forma direta (comunicação explícita) ou de modo indireto (emissão de sinais através do ambiente). Uma *organização* define todas as restrições aplicadas aos agentes pertencentes a uma determinada sociedade, ou seja, os meios através dos quais o projetista do sistema pode garantir que cada agente desejará e realizará a resolução dos problemas propostos.

Não existe uma definição para *agente* que seja aceita por toda a comunidade de IAD. Uma possível definição é proposta por [FER 91]: chama-se *agente* uma entidade real ou abstrata que é capaz de agir sobre ela mesma e sobre seu ambiente, que dispõe de uma representação parcial deste ambiente que, em um universo multiagente, pode comunicar-se com outros agentes, e cujo comportamento é conseqüência de suas observações, de seu conhecimento e das interações com outros agentes.

A definição proposta acima preocupa-se com os mecanismos internos para o tratamento de cada agente, não estabelecendo o tipo de comunicação possível nem a granularidade dos agentes. Em [GAS 92] temos uma definição que ressalta o aspecto da *identidade* de cada agente: um agente é uma entidade à qual pode-se associar uma identidade única, e que é capaz de realizar tarefas formalizadas. Um agente pode ser considerado como um meio que produz um certo número de ações a partir dos conhecimentos e mecanismos internos que lhe são próprios.

Os SMA podem ser caracterizados didaticamente em duas classes, que serão apresentadas a seguir. A primeira denomina-se *Sistemas Multiagentes Reativos* e trabalha com o desenvolvimento de sistemas que utilizam um grande número de agentes simples para a resolução de um determinado problema. A segunda abordagem,

denominada *Sistemas Multiagentes Cognitivos* trabalha com poucos agentes que realizam tarefas mais complexas que os primeiros.

3.1.1 Sistemas multiagentes cognitivos

Os agentes cognitivos são baseados em organizações sociais humanas como grupos, hierarquias e mercados. Segundo [FER 91], as principais características dos Sistemas Multiagentes Cognitivos são as seguintes:

- representação explícita do ambiente e dos outros agentes da sociedade;
- podem manter um histórico das interações e ações passadas e, graças a esta memória, são capazes de planejar suas ações futuras;
- seu sistema de percepção, que permite examinar o ambiente, e o de comunicação, que permite a troca de mensagens entre agentes, são distintos. A comunicação entre agentes é realizada de modo direto, através do envio e recebimento de mensagens;
- seu mecanismo de controle é deliberativo. Os agentes cognitivos raciocinam e decidem em conjunto sobre quais ações devem ser executadas, que planos seguir e que objetivos devem alcançar;
- considerando a última asserção, os modelos de organização dos SMA cognitivos são modelos sociológicos, como as organizações humanas;
- um SMA cognitivo contém, usualmente, poucos agentes, na ordem de algumas dezenas, no máximo.

Considerando como exemplo a arquitetura definida em [SIC 95], apud [ALV 97] p.29, pode-se visualizar a estrutura interna de um agente cognitivo, como visto na fig. 3.1. Este agente é composto por mecanismos de *percepção* e *recepção de mensagens* (entrada de dados), *ação* e *emissão de mensagens* (saída de dados), *raciocínio* e *decisão* (estados internos) e *revisão* (revisão das crenças do agente). O agente raciocina socialmente sobre os demais agentes. Tal mecanismo utiliza as informações que o agente tem sobre os demais, armazenadas numa estrutura denominada *descrição externa*. O agente também possui diversos estados internos, como o *estado de raciocínio* (RS), de *decisão* (DS) e de *engajamento* (CS). Tais estados são modificados através da execução dos diversos mecanismos internos.

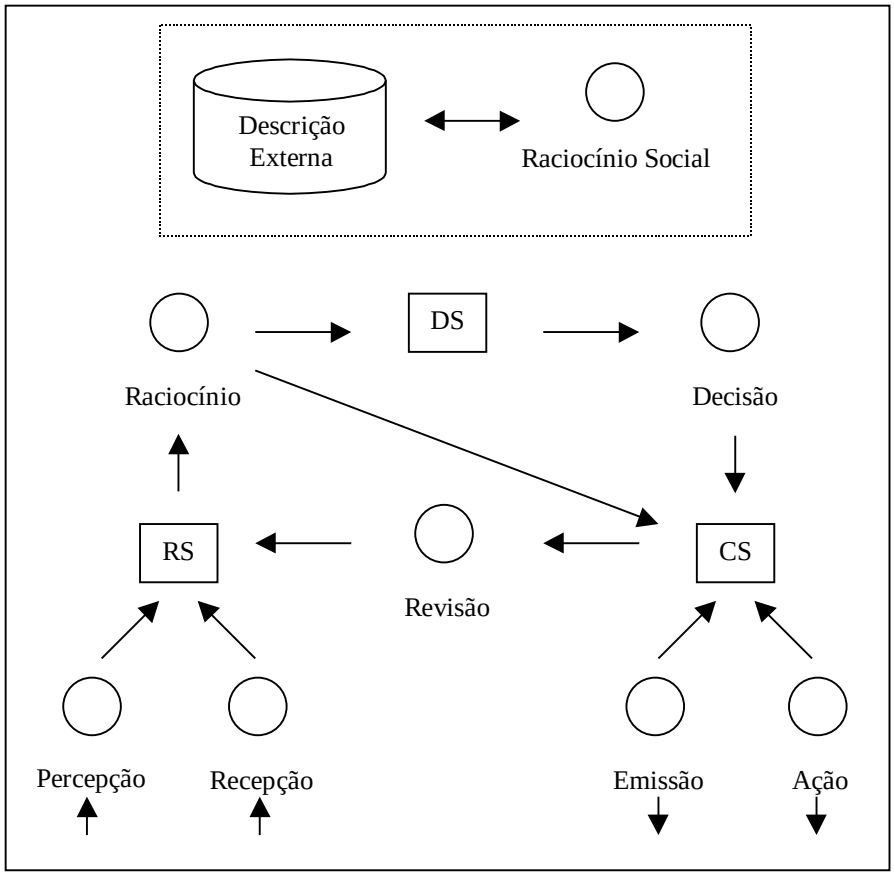


FIGURA 3.1 – Arquitetura para um Agente Cognitivo [SIC 95]

3.1.2 Sistemas multiagentes reativos

Os agentes reativos são baseados em modelos de organização biológica ou etológica (formigas, cupins, abelhas, etc.). O modelo de funcionamento de um agente reativo é formado pelo par *Estímulo-Resposta* (Ação-Reação). As principais características dos agentes e dos sistemas multiagentes reativos são destacadas a seguir [FER 91]:

- não há representação explícita do conhecimento: o conhecimento dos agentes é implícito (as suas regras de comportamento) e sua manifestação se externa através do seu comportamento e dos demais agentes;
- não há representação do ambiente: o comportamento (resposta) de cada agente é baseado no que ele percebe (estímulo) a cada instante. Não há uma representação interna explícita do ambiente;
- não há memória das ações: os agentes reativos não mantém nenhum tipo de histórico de suas ações, ou seja, o resultado de uma determinada ação passada não influencia diretamente na decisão de uma ação futura;
- organização etológica: a forma de organização dos SMA reativos é similar à observada por animais que vivem em grandes comunidades;
- grande número de membros: em geral, os SMA reativos possuem um grande número de agentes, com populações que podem chegar à ordem de milhares de membros.

Em suma, os agentes reativos são muito simples e não possuem representação do seu ambiente. Suas reações dependem unicamente de sua percepção deste ambiente. A Inteligência Artificial clássica define um problema de uma maneira global, criando métodos de resolução que se aplicam diretamente e exclusivamente sobre esta definição. Os modelos de SMA Reativos, por outro lado, concebem o problema como sendo um conjunto de agentes interagindo entre si, onde cada um destes possui seus próprios objetivos individuais. Uma forma usual de representar os comportamentos dos agentes é através de um conjunto de regras.

Dentre as principais aplicações que podem ser modeladas através de SMA Reativos, podemos destacar:

- Vida Artificial: Os SMA reativos abriram um amplo horizonte para o desenvolvimento de sistemas de Vida Artificial, devido a similaridade entre as suas características principais: criação de entidades simples, independentes e capazes de atuar num meio complexo. Neste caso, o conceito de agente reativo é modelado como um ser independente, que possui a capacidade de gerar novos agentes (descendentes) se uma determinada condição for satisfeita.
- Recuperação de Informações: busca de informações altamente específicas em ambientes de troca de dados heterogêneos, tais como rede de computadores.
- Telecomunicações: SMA reativos podem ser utilizados na construção de sistemas de telecomunicações mais abertos, flexíveis e, principalmente, interativos. De acordo com as ações realizadas nos pontos de transmissão e recepção, os agentes poderiam reconfigurar o sistema, adequando-se ao novo ambiente apresentado.

3.2 Definição de regras em SMAR

Segundo [RIC 93] um conjunto de regras deve representar o conhecimento sobre os relacionamentos do mundo e o conhecimento sobre como solucionar problemas usando o conteúdo das regras. Existem duas formas de representar um conhecimento através de um conjunto de regras:

- representação declarativa: para este modelo, o conhecimento é especificado, mas o uso que será feito dele não é fornecido. Para utilizar este estilo de representação, deve-se ampliá-lo com um programa que especifique o que deve ser feito com o conhecimento e como. É importante salientar que existe uma maneira diferente de visualizar as definições lógicas declarativas, visualizando-as como um programa e não como dados de um programa. Neste aspecto, as declarações de implicação definem os caminhos legítimos de raciocínio e as declarações atômicas fornecem os pontos de partida desses caminhos;
- representação procedual: neste caso, as informações de controle necessárias ao uso do conhecimento estão embutidas no próprio conhecimento. Para usar uma representação procedimental, precisa-se ampliá-la com um interpretador que siga as instruções fornecidas no conhecimento.

A diferença entre as visões declarativa e procedimental está em onde residem as informações de controle. Considerando a seguinte base de conhecimento:

humano (Luísa)
humano (Lenise)
pessoa (Palhares)

$\forall x$: *humano (x)* $\rightarrow$ *pessoa (x)*

Ao extrairmos desta base de conhecimentos a resposta para a seguinte pergunta:

$\exists y$: *pessoa (y)*

A base de conhecimentos apresentada justifica qualquer uma das seguintes respostas:

y = Luísa
y = Lenise
y = Palhares

Como existe mais de um valor que satisfaz a pergunta, mas apenas uma resposta é necessária, esta dependerá da ordem em que as definições são examinadas durante a busca pela resposta. Se a base de conhecimentos for declarativa, não há nada que diga como estas serão examinadas. Se a base for procedimental, a resposta dependerá da ordem em que os procedimentos forem implementados. O método de busca utilizado pelo motor de inferência (busca em profundidade, busca em largura, raciocínio para frente, raciocínio para trás [RIC 93]) também influenciará o resultado final.

Uma forma de resolver esta ambigüidade é atribuir uma forma de preferência com base nas próprias regras. Existem duas formas de atribuir a preferência entre as regras. A primeira considera que estas tenham sido explicitadas em uma determinada ordem, por exemplo, a ordem física em que foram apresentadas ao sistema (a prioridade é atribuída às regras na ordem em que elas aparecem). Um outro esquema comum de preferência direcionada por regras é prioridade de regras mais específicas sobre regras mais genéricas.

3.3 O Ambiente Simula

O ambiente Simula [FRO 97], que serviu de base para o sistema desenvolvido neste trabalho, foi concebido tendo como objetivos a construção de um ambiente didático de desenvolvimento de aplicações em SMAs reativos. O sistema permite ao usuário criar aplicações em sistemas multiagentes reativos, através de elementos de uma interface gráfica, os quais permitem a este usuário determinar os agentes envolvidos no problema e como eles agirão no processo de resolução do mesmo. O usuário define a sua aplicação criando um modelo para ela.

A modelagem consiste em representar um problema utilizando um grupo de agentes reativos que interagem entre si e com o ambiente no qual estão inseridos,

visando atingir uma solução. Portanto, a função do usuário é definir a situação inicial do problema e determinar como os agentes agirão. Após estas definições, o ambiente se encarrega de executar a simulação e apresentar uma situação final, atingida pela atuação dos agentes.

As principais características do ambiente de desenvolvimento Simula são listadas a seguir:

- apresenta uma interface gráfica, baseada em janelas e menus, que permite definir um problema ou uma situação como um sistema multiagente reativo em um ambiente bidimensional;
- permite descrever os tipos de agentes envolvidos no problema ou na situação, que agirão em busca de uma solução;
- permite descrever regras de comportamento dos agentes definidos: as regras de comportamento determinam como os agentes agirão (quais são as suas ações) para atingir uma solução, e quando (em que situação) a regra será ativada. O sistema Simula possui um conjunto de comportamentos pré-definidos que são utilizados para montar as regras de comportamento dos agentes;
- permite situar os agentes em seu ambiente, determinando a posição inicial de cada um;
- a apresentação do processo de resolução do problema ou da situação, com a atividade dos agentes em seu ambiente, é feita de forma gráfica.

O ambiente Simula destina-se a atender usuários que tenham conhecimento da tecnologia do uso de agentes para a construção de sistemas científicos, didáticos, simulações, etc. O objetivo do sistema é diminuir o esforço de programação do usuário para criar suas aplicações, estimulando o mesmo a projetar novos sistemas com o uso de agentes reativos.

A fig. 3.2 apresenta um esquema de como o usuário do ambiente Simula deve proceder para desenvolver suas aplicações em sistemas multiagentes reativos e quais são as etapas internas de execução até ser atingida uma solução (situação final) para a aplicação descrita.

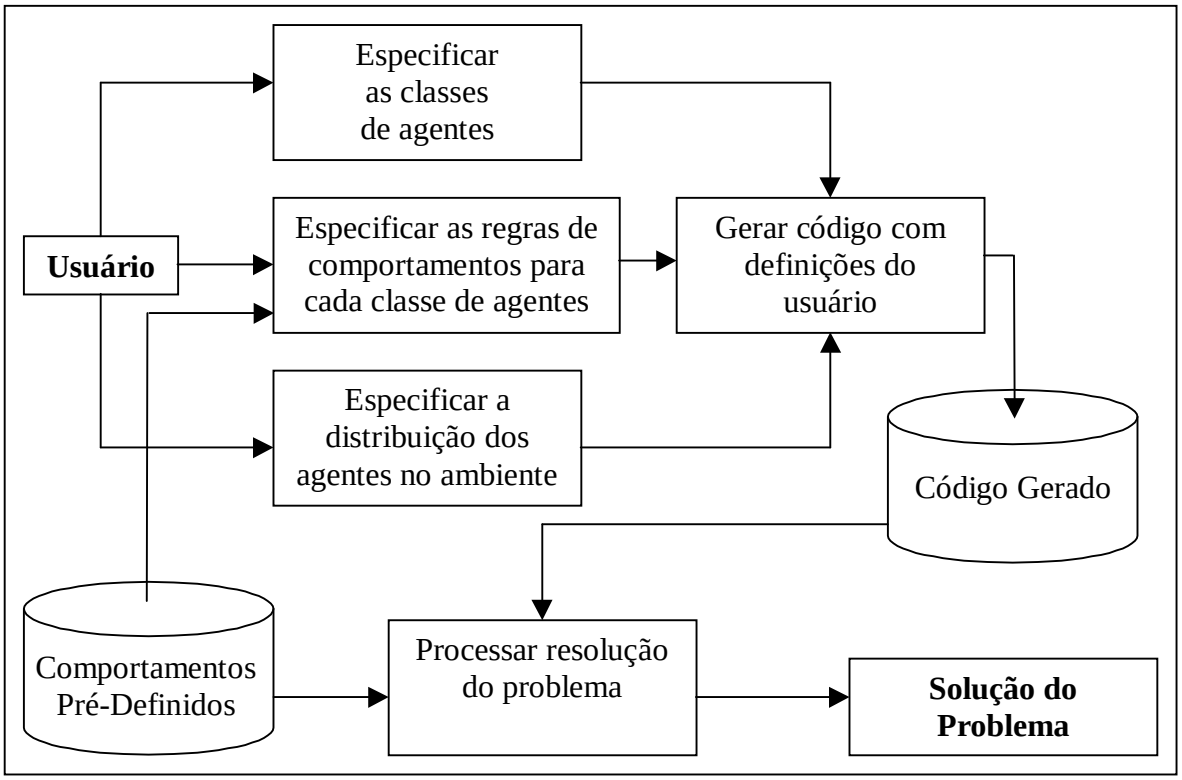


FIGURA 3.2 – Arquitetura do Ambiente Simulla [FRO 97]

O ambiente Simula foi estruturado em dois módulos: módulo de definição e módulo de execução.

3.3.1 Módulo de definição

O usuário interage com o ambiente Simula através do módulo de definição. Nesta etapa o usuário deverá descrever o problema (aplicação) sob a forma de uma população de agentes, atribuindo comportamento aos mesmos. Os comportamentos representam o processo de ação dos agentes, no sentido de atingirem uma situação final, vista pelo usuário, como uma solução do problema.

Este módulo divide-se em três partes:

- especificação dos tipos de agentes que farão parte do sistema: o usuário especifica quais serão os agentes envolvidos no processo, associando um nome para cada classe de agentes. Agentes da mesma classe possuem as mesmas características e as mesmas ações;
- especificação das regras de comportamento dos agentes: as regras de comportamento são especificadas para cada classe de agente e são elas que guiam as ações dos agentes no processo de resolução de problemas. Para especificar estas regras, o usuário utiliza um conjunto de comportamentos pré-definidos do ambiente e também pode utilizar variáveis criadas pelo próprio usuário;
- especificação da disposição dos agentes no ambiente: pode-se alterar as dimensões do ambiente e a determinar a posição inicial de cada agente no ambiente no qual estes atuarão.

A especificação das regras de comportamento para cada classe de agentes constitui-se na principal tarefa no desenvolvimento de aplicações que utilizam o paradigma dos sistemas multiagentes reativos. No ambiente Simula, cada regra pode possuir até três elementos:

- pré-condição: condição para a execução do comportamento;
- ação-ativada: comportamento a ser executado;
- prioridade: ordem para a execução das regras.

O ambiente Simula utiliza uma representação procedimental para a construção das regras de comportamento. Cada regra é representada por um procedimento construído a partir de comportamentos atômicos pré-definidos, que são ligados através de conetivos lógicos, como visto nos exemplos abaixo:

Agente Caçador

Regra 1:
pré-condição: atinge_agente(Presa)
ação-ativada: mata_agente(Presa)
prioridade: 0

Regra 2:
pré-condição: percebe_agente(Presa) AND NOT(percebe_agente(Caçador))
ação-ativada: persegue_agente(Presa)
prioridade: 1

A prioridade de cada regra é definida explicitamente pelo usuário, que define as regras e a ordem de precedência, atribuindo um valor menor para regras mais prioritárias. É importante salientar que a construção de regras para cada classe de agentes é definida completamente pelo usuário, e sua performance depende unicamente do quão bem estas regras de comportamento foram concebidas.

3.3.2 Módulo de execução do ambiente Simula

Após serem realizadas todas as definições necessárias, o usuário poderá acionar o módulo de execução do ambiente Simula. Este módulo divide-se em duas partes.

- geração de código: este processo utilizará as definições feitas pelo usuário para a sua aplicação, juntamente com os comportamentos pré-definidos do ambiente para gerar um código (programa em linguagem Java), que será responsável pelo processo de resolução da aplicação;
- execução: corresponde à execução do código gerado e representa a ativação do processo de resolução da aplicação definida. Este processo é mostrado de forma gráfica, através de uma simulação da atuação dos agentes e de seus movimentos/deslocamentos no ambiente. Cada agente é representado por um ícone definido pelo usuário. O término do processo indica que os agentes atingiram uma situação final, que pode ser considerada como uma solução.

4 Proposta do Ambiente de Evolução de Comportamentos

4.1 Introdução ao Simula++

O ambiente Simula++ foi desenvolvido a partir do modelo proposto em [FRO 97], visto na seção 3.3, que tinha como objetivos a construção de um protótipo de um ambiente de desenvolvimento de aplicações em SMA reativos. A partir deste protótipo, foram especificadas e implementadas extensões que permitem ao sistema trabalhar com agentes cujos comportamentos possam evoluir, utilizando os paradigmas da evolução biológica, da Vida Artificial e dos Algoritmos Evolutivos, apresentados no capítulo 2. O principal objetivo do ambiente Simula++ é prover uma ferramenta didática de modelagem e execução de Sistemas Multiagente Reativos, onde os comportamentos definidos para cada classe de agentes evoluam durante o tempo de execução.

As principais características do Simula++ são descritas a seguir:

4.1.1 Tipo de agentes

A arquitetura do Simula++ pressupõe a definição de SMAs reativos. Entre as principais motivações para a utilização de sistemas reativos, podemos citar: as soluções encontradas tendem a ser mais robustas, pois o sistema é mais tolerante a falhas ou ao mal funcionamento de algum agente; simplicidade, não havendo necessidade da elaboração de planos de ação ou coordenação entre os agentes; facilidade no desenvolvimento, pois, a princípio, não há necessidade de ter um conhecimento mais aprofundado sobre SMAR e seus componentes, quando estes forem definidos.

4.1.2 Definição das regras

O sistema utiliza uma representação declarativa para a definição das regras e sua utilização no ambiente. As regras de comportamento dos agentes possuem três componentes:

- pré-condição: condição para a execução do comportamento;
- ação-ativada: comportamento a ser executado;
- prioridade: ordem para a execução das regras.

No entanto, as regras de comportamento do ambiente Simula++ não são estáticas, tais como ocorria no sistema Simula. As regras podem sofrer modificações dinamicamente, em tempo de execução, evoluindo a partir de um estado inicial, buscando uma melhor adaptação ao ambiente definido. Além disso, cada agente, individualmente, evolui o seu próprio conjunto de regras, de forma independente. A noção de um *conjunto de regras* para cada *tipo de agente* (grupo) é trocada para uma relação direta entre *conjunto de regras* por *agente* (indivíduo).

4.1.3 Ambiente de simulação

Utilizando as definições de [LAT 98] (ver capítulo 2.5), o ambiente Simula++ pode ser classificado como *discreto*, que é uma sub-classe da classse *situado*, que por sua vez é uma sub-classe da classe *abstrato*. Ou seja, o ambiente implementado possui características das três classes citadas acima, que são descritas à seguir:

Classe Abstrato

- S (tamanho do ambiente): um ambiente definido no Simula++ é finito, sendo seu tamanho definido pelo usuário;
- O (conjunto de objetos do ambiente): o ambiente proposto pode ser formado por agentes ativos e/ou agentes inativos;
- I (conjunto de leis de interação): as regras gerais para o Simula++ podem ser assim definidas: cada agente possui um conjunto de regras e um estado interno próprio; todos os agentes pertencem a um único ambiente; um agente pode modificar o ambiente em que se encontra, atuando neste de forma construtiva (originando novos agentes) ou destrutiva (eliminando agentes); a cada passo de simulação, um agente percebe o mundo à sua volta e, baseado nas suas regras, no seu estado interno e na visualização do ambiente, toma uma decisão, agindo sobre o ambiente;
- E (conjunto de leis de evolução): o ambiente não sofre evolução, apesar de poder variar dinamicamente. Apenas os agentes estão sob a influência do Algoritmo Evolutivo.

Classe Situado

- F (forma): o ambiente Simula++ utiliza ícones formados por quadrados de 16x16 *pixels*, definidos pelo usuário, para representar os diferentes tipos de agentes. Cada ícone pode ser desenhado utilizando até no máximo onze cores diferentes;
- D (dimensão): o ambiente é bidimensional, sendo representado por uma matriz *mxn*, com *m* linhas por *n* colunas;.
- M (métrica): a unidade de medida e a escala para a definição das coordenadas estão baseadas nos ícones, que representam os agentes. Por exemplo, um ambiente 10x10 é formado por um retângulo de 160 por 160 pixels, divididos em 100 quadrados de 16x16 pixels;
- P (conjunto de propriedades de gerência do espaço): a visualização do ambiente é realizada através de uma janela de visualização formada por uma matriz 25x25. Ambientes com dimensões maiores podem ser visualizados através de barras de rolagem que ficam ao lado da câmara sintética.

Classe Discreto

- L (estrutura das células): como visto anteriormente, a célula é representada por um quadrado de 16x16 *pixels*;
- Mi (métrica inteira para o ambiente): as partes elementares (células), são formadas por quadrados de 16x16 pixels.

4.1.4 Evolução e adaptabilidade

A principal característica do sistema implementado está concentrada na adaptabilidade dos agentes, conseguida através da utilização do algoritmo evolutivo sobre o conjunto de regras individuais de cada agente. Através da recombinação e mutação das regras iniciais, novas regras e comportamentos são construídos e testados dinamicamente pelo sistema, como veremos na seção 4.2.

A arquitetura funcional do ambiente Simula++ pode ser vista na fig. 4.1:

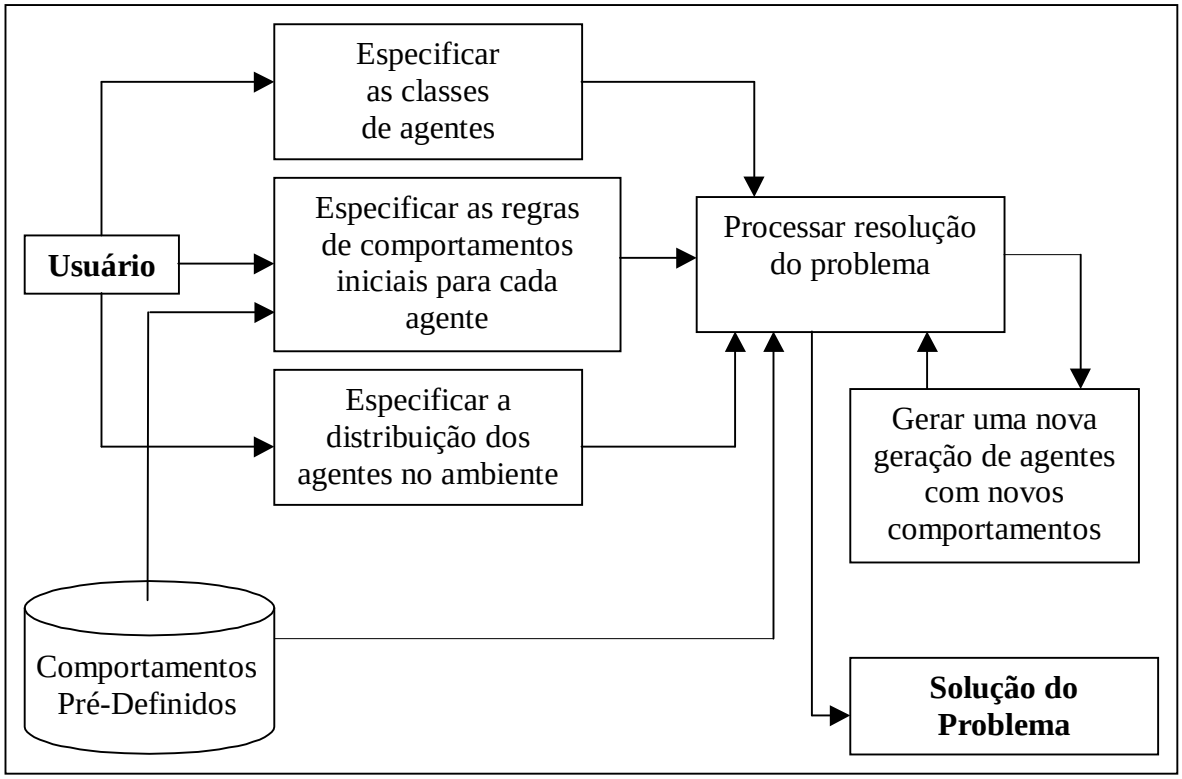


FIGURA 4.1 – Arquitetura do Ambiente Simula++

4.2 Modelo de Evolução para as Regras de Comportamento

Nesta seção apresentaremos o modelo utilizado para a evolução das regras de comportamento de cada agente do sistema.

4.2.1 Estrutura dos Agentes

A estrutura dos agentes do ambiente SIMULA++ está dividida em duas partes:

- Elementos Independentes: variáveis que não sofrem modificações pelo algoritmo evolutivo durante o processo de simulação. Estas variáveis são utilizadas para o controle do processo de evolução. As principais variáveis estáticas estão listadas à seguir:

- Ei (Energia Inicial): quantidade de energia inicial disponível para o agente.
- qE (Quantidade Energia): quantidade de energia do agente. Esta quantidade diminui de uma unidade a cada passo de simulação. O agente pode aumentar sua quantidade de energia, retirando-a do ambiente.
- tM (Tempo Máximo Vida): tempo máximo de simulação para cada agente.
- tV (Tempo Vida): número de passos de simulação já executados pelo agente.
- tS (Tempo Maturidade Sexual): tempo necessário para que o agente tenha a capacidade de gerar descendentes.
- tNG (Tempo Nova Geração): tempo necessário para que o agente gere um novo descendente, após o período definido pelo tS. Usualmente tNG << tS.

- Cromossomo: quantidade e definição das regras de comportamento. Cada agente possui um cromossomo que codifica as regras de comportamento deste agente. Estas regras podem ser modificadas através dos operadores de recombinação e mutação, que estão definidos nas próximas seções.

4.2.2 Estratégia de Controle

O controle da simulação utiliza duas estruturas principais:

- vA (Vetor Agentes): armazena as estruturas de todos agentes que ainda estão atuando no sistema.
- vAF (Vetor Agentes Férteis): armazena uma lista de ponteiros para todos os agentes ainda ativos, que já ultrapassaram o Tempo de Maturidade Sexual.

Como visto anteriormente, todos os agentes possuem um período máximo de vida (tM) e de maturidade sexual (tS) pré-determinados. Estes períodos são iguais para todos os agentes da mesma classe, o que garante a eqüidade de condições de sobrevivência para todos os indivíduos constituintes do sistema.

O período de maturidade sexual deve ser considerado como um teste para as regras de comportamento de cada agente. Se o agente permanecer vivo após este espaço de tempo, ele é considerado um agente adaptado ao ambiente e, por isso, capaz de gerar descendentes. Quando um agente atinge o tempo de maturidade sexual, ele é incluído no Vetor de Agentes Férteis, que contém todos os agentes aptos a gerar novos descendentes. Além disso, ele deve escolher desta lista um segundo agente, que é utilizado como par para a geração de um novo descendente. Após ser incluído na lista de agentes aptos a reproduzir, cada agente gerará um novo descendente em ciclos pré definidos pelo usuário (tNG).

A fig. 4.2 apresenta o algoritmo de execução para todos os agentes definidos no sistema Simula++, considerando *nAgentes* o número de agentes do ambiente modelado. A fig. 4.3 mostra um possível ciclo de vida para um agente com as seguintes características tM = 100, tS = 80 e um tNG = 7.

É importante salientar que um agente não vive obrigatoriamente até o final de seu tempo máximo de vida (tM) e, portanto, não gerará sempre um número fixo de descendentes. Se a quantidade de energia interna do agente (qE) chegar a zero, o agente é eliminado. A sobrevivência do agente depende do ambiente dinâmico em que este vive e de sua adaptabilidade ao mesmo.

Os novos agentes gerados herdam as regras de comportamentos de seus pais, através da aplicação dos três operadores genéticos, que são explicados a seguir: recombinação externa, recombinação interna e mutação.

```
PARA i = 0 ATÉ nAgentes
  vA[i].Inserir(Agente[i])
nFerteis = 0
ENQUANTO Critério_Parada = false
  PARA i = 0 ATÉ nAgentes
    SE ( vA[i].qE > 0 ) AND ( vA[i].tV <= tM ) ENTÃO
      vA[i].Perceba( Ambiente )
      vA[i].Perceba( Estado_Interno )
      vA[i].ExecuteAcao( Escolha_Acao() )
      vA[i].tV = vA[i] + 1
      SE ( vA[i].tV >= tS ) ENTÃO
          SE ( vA[i].tV = tS ) ENTÃO
              vAF[nFerteis].Inserir(vA[i])
              nFerteis ++
              Pai2 = Escolha_Pai2( vAF )
              Filho = GereNovoAgente( vA[i], Pai2 )
              nAgentes++
              vA[nAgentes].Inserir(Filho)
          SENÃO
              SE ( vA[i].tV % tNG = 0 ) ENTÃO
                  Pai2 = Escolha_Pai2( vAF )
                  Filho = GereNovoAgente( Pai2 )
                  nAgentes++
                  vA[nAgentes].Inserir(Filho)
              FIM-SE
          FIM-SE
      FIM-SE
    SENÃO
      vAF.RemoveAgente(vA[i])
      vA[i].RemoveAgente()
      nAgentes = nAgentes - 1
    FIM-SE
  FIM-PARA
FIM-ENQUANTO
```

FIGURA 4.2 – Algoritmo de Execução dos Agentes

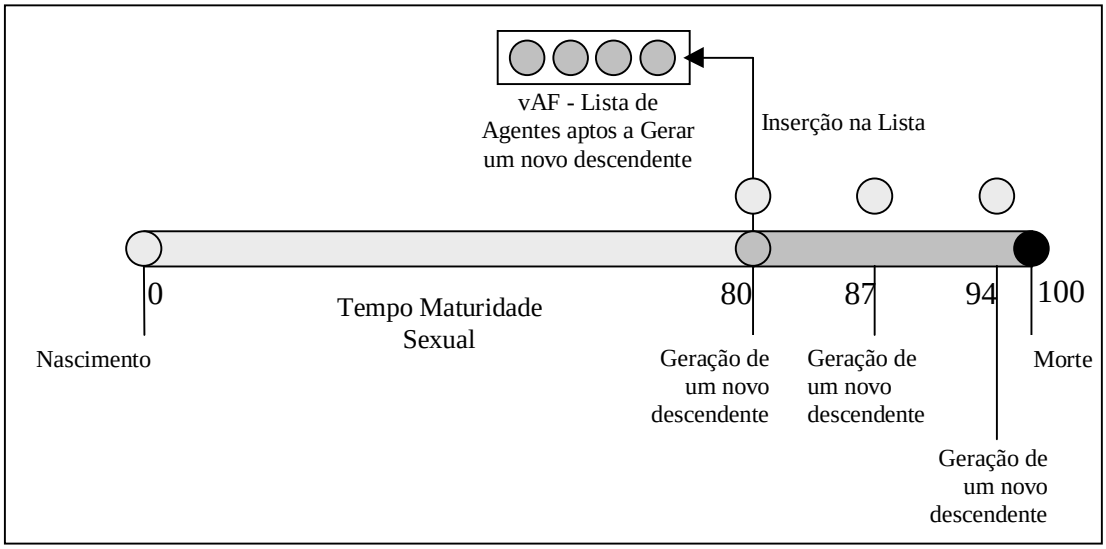


FIGURA 4.3 – Ciclo de Vida de um Agente

4.2.3 Recombinação Externa

Os operadores de recombinação são utilizados durante a geração de um novo descendente, utilizando os cromossomos (representando as regras de comportamento) dos agentes genitores. Consideremos as seguintes definições:

- PaiUm: agente que deve gerar um novo descendente.
- PaiDois: agente escolhido na Lista de Agentes Férteis. Existem dois métodos de escolha: aleatória ou através do conceito de elitismo. Neste último caso, é escolhido o agente com o maior valor de *fitness*.
- Filho: agente que está sendo gerado.

O operador Recombinação Externa é utilizado durante a reprodução e tem como objetivo a geração de um novo cromossomo, baseado nos agentes PaiUm e PaiDois. Este cromossomo representa a nova coleção de regras de comportamento do agente Filho gerado. O número de regras do agente Filho é igual a média do número de regras dos agentes PaiUm e PaiDois.

A Recombinação Externa embaralha as regras dos agentes PaiUm e PaiDois, gerando um descendente que possui características (regras de comportamento) herdadas dos dois pais. Este processo trabalha com as regras inteiras. Existem duas opções implementadas para a Recombinação Externa: recombinação de um ponto e recombinação de dois pontos. A recombinação de um ponto realiza uma quebra em uma posição aleatória do conjunto de regras dos genitores, selecionando as primeiras regras do agente PaiUm e as demais do agente PaiDois. A recombinação de dois pontos realiza a quebra em duas posições aleatórias do conjunto de regras dos genitores. Este tipo de recombinação seleciona as primeiras regras do agente PaiUm, as regras intermediárias do agente PaiDois e a últimas regras novamente do agente PaiUm. Para os dois tipos de recombinação, a prioridade das regras não muda quando da cópia das mesmas para o cromossomo do agente Filho. A fig. 4.4 apresenta a forma como são realizadas a recombinação externa de um ponto e de dois pontos. A fig. 4.5 apresenta um exemplo de uma recombinação externa de um ponto no sistema Simula++.

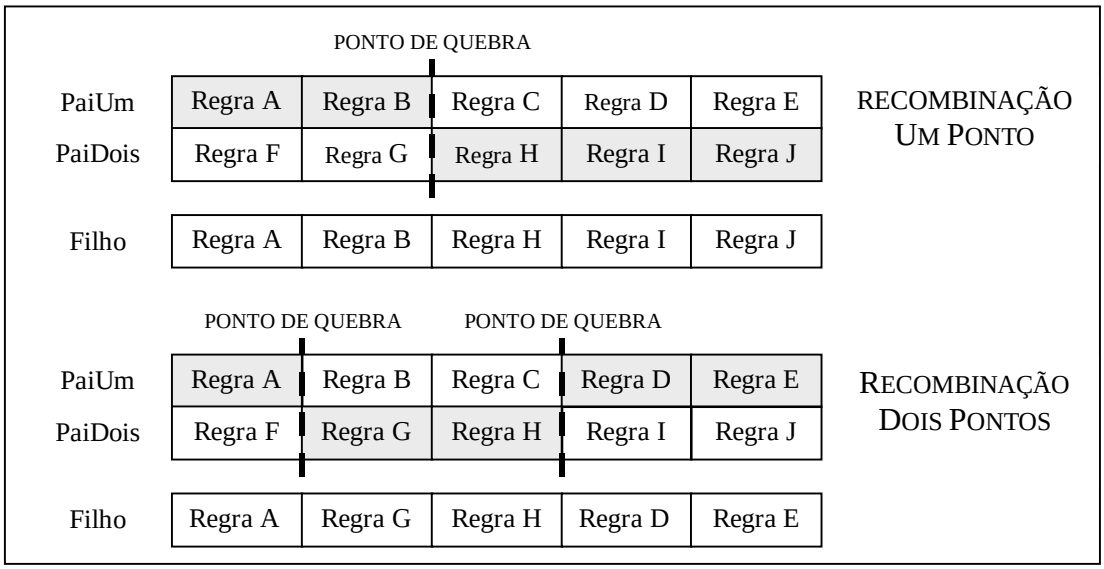


FIGURA 4.4 – Modelo para a Recombinação Externa de Um Ponto e Dois Pontos

Cromossomo - Agente PaiUm	
Regra 1.1	SE percebe_agente(Minério) ENTÃO segue_agente(Minério)
Regra 1.2	SE atinge_agente(Minério) ENTÃO pega_carga(Minério)
Regra 1.3	SE carga=TRUE ENTÃO segue_agente(Base)
Regra 1.4	SE !percebe_agente(Minério) ENTÃO movimento_aleatório( )

Cromossomo - Agente PaiDois	
Regra 2.1	SE percebe_agente(Minério) ENTÃO segue_agente(Minério) AND remove_pista( )
Regra 2.2	SE atinge_agente(Minério) ENTÃO pega_carga(Minério)
Regra 2.3	SE carga=TRUE ENTÃO segue_agente(Base) AND deixa_pista( )
Regra 2.4	SE percebe_pista( ) ENTÃO segue_pista( )

Cromossomo - Agente Filho	
Regra 1.1	SE percebe_agente(Minério) ENTÃO segue_agente(Minério)
Regra 1.2	SE atinge_agente(Minério) ENTÃO pega_carga(Minério)
Regra 2.3	SE carga=TRUE ENTÃO segue_agente(Base) AND deixa_pista( )
Regra 2.4	SE percebe_pista( ) ENTÃO segue_pista( )

FIGURA 4.5 – Exemplo de Recombinação Externa de um Ponto

No exemplo mostrado na fig. 4.5, temos a geração de uma nova coleção de regras (cromossomo) para o agente Filho, através do embaralhamento das regras dos agentes PaiUm e PaiDois, utilizando a recombinação externa de um ponto. No caso específico mostrado acima, o agente Filho recebeu a primeira e a segunda regra do agente PaiUm e a terceira e quarta regra do agente PaiDois.

4.2.4 Recombinação Interna

O operador Recombinação Interna é utilizado para o mesmo propósito do operador Recombinação Externa: a geração de um novo descendente (agente Filho), a partir dos cromossomos dos agentes genitores (PaiUm e PaiDois). A diferença entre os dois operadores está na forma como é construído o novo cromossomo a partir das regras dos agentes PaiUm e PaiDois.

A Recombinação Interna trabalha com os dois componentes das regras de comportamento (pré-condição e ação-ativada) e utiliza somente a recombinação de um ponto. Esta recombinação realiza uma quebra em uma posição aleatória do conjunto de

regras dos genitores, separando-as em dois grupos, que serão embaralhados individualmente para a geração do cromossomo do agente Filho. O primeiro grupo é formado pelos elementos de pré-condição do agente PaiUm e pelos elementos de ação-ativada do agente PaiDois, enquanto que o segundo grupo é formado pelos elementos de pré-condição do agente PaiDois e os elementos de ação-ativada do agente PaiUm, como visto na fig. 4.5. O conjunto formado pela união dos dois grupos será utilizado como o novo cromossomo do agente Filho.

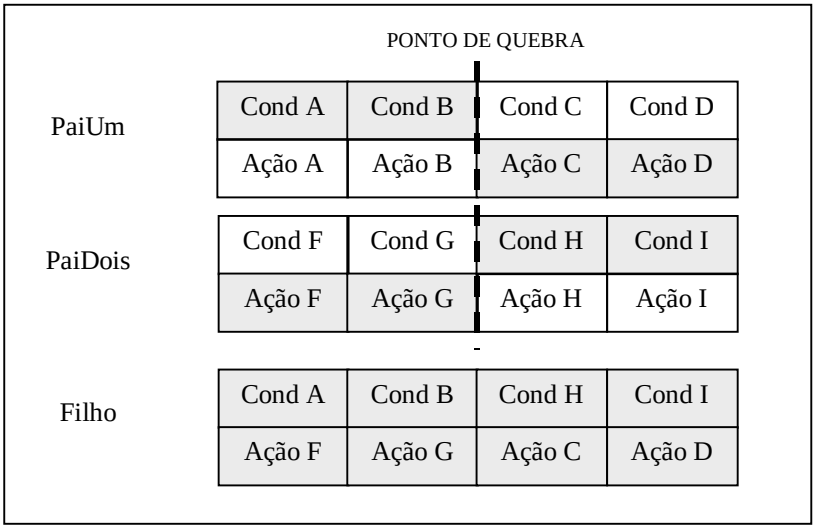


FIGURA 4.6 – Modelo para a Recombinação Interna

Cromossomo – Agente PaiUm	
Regra 1.1	
Cond	Se percebe_agente(Inimigo)
Ação	Então foge_de_agente (Inimigo)
Regra 1.2	
Cond	Se percebe_agente(Comida)
Ação	Então segue_agente (Comida)

Cromossomo – Agente PaiDois	
Regra 2.1	
Cond	Se ! percebe_agente(Comida)
Ação	Então movimento_aleatório ( )
Regra 2.2	
Cond	Se percebe_agente(Inimigo)
Ação	Então mata_agente (Inimigo)

FIGURA 4.7 – Agentes Genitores para um exemplo de Recombinação Interna
As regras estão divididas em suas duas parcelas principais.

Cromossomo – Agente Filho	
Regra 3.1	
Cond	Se percebe_agente(Inimigo)
Ação	Então movimento_aleatório ( )
Regra 3.2	
Cond	Se percebe_agente(Inimigo)
Ação	Então segue_agente (Comida)

FIGURA 4.8 – Agente Filho Gerado Utilizando a Recombinação Interna

A fig. 4.8 demonstra um possível filho gerado pela Recombinação Interna dos dois cromossomos PaiUm e PaiDois, visualizados na fig. 4.7. O novo cromossomo é gerado através da permuta entre os componentes (pré-condição e ação ativada) dos dois agentes genitores. A prioridade de cada nova regra do agente Filho é definida através de um sorteio entre as duas prioridades das regras que a originaram.

A Recombinação Interna expande o espaço de busca do sistemas, pois acaba gerando um grande número de regras que não foram testadas anteriormente, enquanto que a Recombinação Externa acaba por gerar apenas novas combinações de regras já geradas pelo sistema. Apesar desta aparente vantagem, a Recombinação Interna, em sua constante geração de novas regras, acaba produzindo um grande número de agentes que não apresentam boas regras de comportamento, aumentando o tempo de processamento do sistema.

4.2.5 Mutação

O operador de mutação é utilizado para modificar, aleatoriamente, o cromossomo do agente Filho, ou seja, alterar o conjunto de regras de comportamento deste agente após a sua geração pelo operador de recombinação. Esta alteração é realizada através da troca randômica de elementos da regra por expressões de mesmo valor sintático, que são escolhidas através de um sorteio dirigido.

Este operador atua após a geração do novo cromossomo de duas formas bem distintas: alteração no número de regras do cromossomo e modificação de uma regra específica. A primeira forma aumenta ou diminui o número de regras do cromossomo, utilizando o algoritmo descrito na fig. 4.9:

1. Na inicialização do sistema, é definido um percentual de mutação(M), normalmente entre 0.01 e 0.02, que é utilizado durante toda a simulação.

2. Para cada novo agente gerado, é sorteado um número randômico(R) entre 0 e 1. Caso R<= M, o cromossomo do agente será alterado.

   Alteração:
   - É definido se haverá uma inserção de novas regras no cromossomo (através da cópia randômica de uma regra de um dos dois agentes genitores) ou uma remoção (através da eliminação de uma regra randomicamente) por sorteio, onde as duas ações concorrem com 50% de chance de tornarem-se vencedoras.
   - Após a escolha da ação a ser executada, é sorteado o número de regras que devem ser inseridas ou removidas. Este sorteio não pode inserir ou remover mais do que ¼ do número total de regras do agente Filho.

FIGURA 4.9 – Operador de Mutação: Modificação do Número de Regras de Comportamento

A segunda forma do operador de mutação é utilizada sob o novo conjunto de regras do cromossomo do agente Filho, que já sofreu as modificações discutidas na fig. 4.9. Neste caso, o operador de mutação tem a função de fazer pequenas modificações nas expressões utilizadas pelas regras de comportamento, com a finalidade de alterá-las e, conseqüentemente, expandir o espaço de busca.

O algoritmo para a modificação do cromossomo (regras de comportamento) pelo operador de mutação pode ser visualizado na fig. 4.10.

1. Na inicialização do sistema, é definido um percentual de mutação(M2), normalmente entre 0.01 e 0.02, que é utilizado durante toda a simulação.

2. Para cada elemento constituinte da regra de comportamento, é sorteado um número randômico(R). Caso R<=M2, é realizada a troca da expressão.

FIGURA 4.10 – Operador de Mutação: Modificação das Regras de Comportamento

O operador de mutação, utilizado na modificação do cromossomo, atua como um gerador de novas regras de comportamento, pois mesmo pequenas modificações podem alterar sensivelmente o desempenho de um agente. Isto ocorre devido a forma como foram construídas as regras pré-definidas que, quando alteradas, modificam completamente o sentido da regra de comportamento. A fig. 4.11 apresenta uma possível modificação das regras de comportamento gerado por mutação.

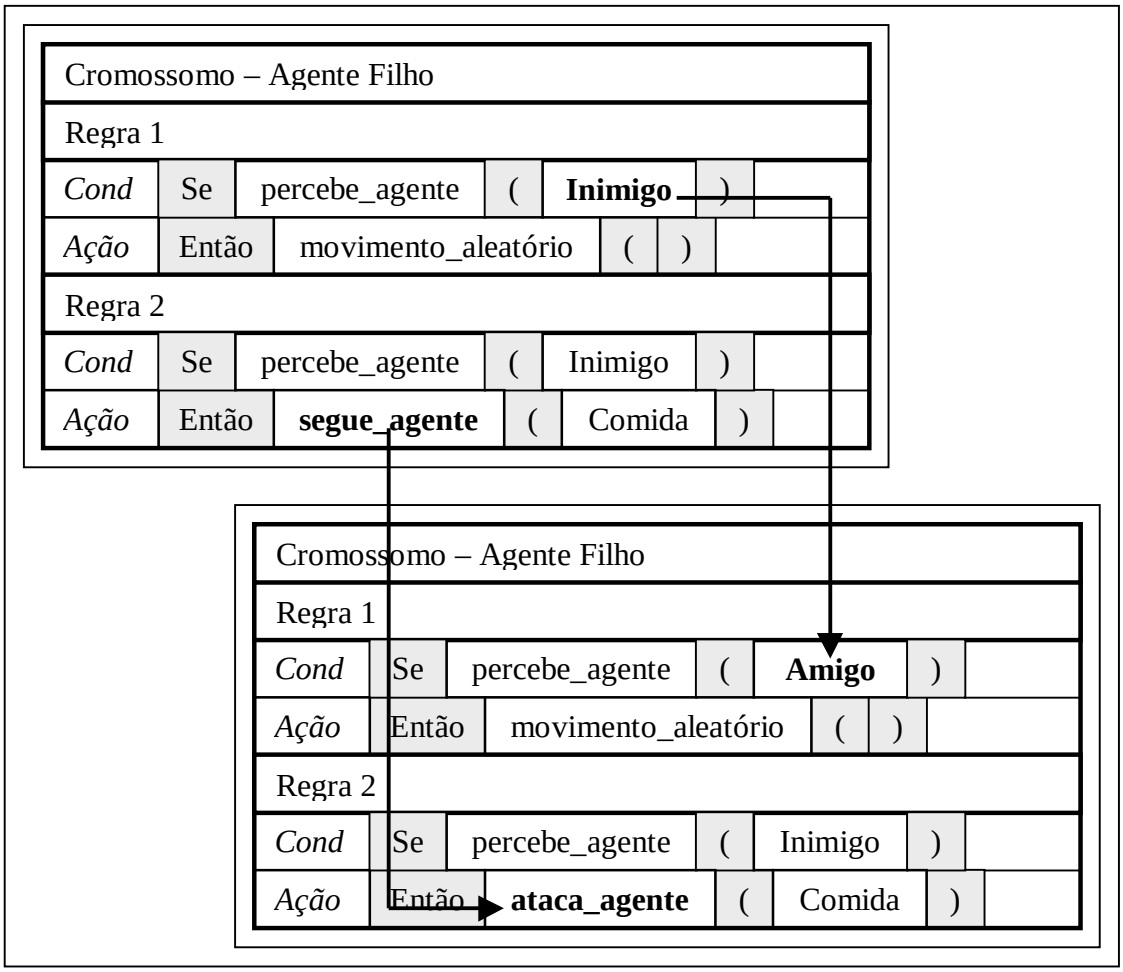


FIGURA 4.11 – Exemplo do Operador de Mutação

Os dois elementos em negrito são modificados aleatoriamente, gerando o agente Filho final, que será incluído no Vetor de Agentes. Os elementos hachurados em cinza não podem ser modificados, pois fazem parte da definição sintática das regras de comportamento.

4.2.6 Função de Fitness

A princípio, não existe uma função de *fitness* explícita no sistema, pois o grau de adaptabilidade do agente é medida por sua sobrevivência. Ou seja, quanto mais tempo um agente estiver em simulação, maior será o seu grau de *fitness*. No entanto, para algumas modelagens específicas, pode ser importante ou até mesmo essencial a definição de uma função explícita. O ambiente Simula++ define três variáveis (sucesso, carga e energia) para cada agente, que podem ser utilizadas para a definição de uma função de *fitness* explicitamente. Esta questão será abordada mais profundamente nos capítulos 6 e 7.

5 O Protótipo do Ambiente Simula++

Foi desenvolvido um protótipo do ambiente Simula++ objetivando a verificação e validação dos algoritmos propostos anteriormente através de estudos de caso que serão discutidos no próximo capítulo.

O protótipo foi desenvolvido na linguagem de programação Java da Sun Microsystems, devido a sua portabilidade, o que permite a utilização deste em ambientes baseados na plataforma Windows/PC e em plataformas que utilizem o sistema operacional Unix, entre outros. Foi utilizado o Kit de Desenvolvimento Java 1.1.6.

5.1 A interface do Protótipo

A figura 5.1 mostra a interface do protótipo desenvolvido.

FIGURA 5.1 – Interface do Ambiente Simula++

A interface do ambiente é constituída por um menu com quatro opções, através das quais o usuário define e modela todas as partes constituintes da aplicação desejada. Cada um dos menus é descrito nas seções seguintes.

5.1.1 Menu Arquivo

A definição do ambiente SMAR, assim como seus respectivos agentes, são armazenados em disco, no formato de um arquivo de dados. Neste menu, o arquivo pode ser criado (*Novo*), aberto (*Abrir*) ou salvo (*Salvar* e *Salvar como*) para posterior reutilização. É importante salientar que um arquivo corresponde a um SMAR completo, com todas as suas definições (agentes, regras de comportamento, posições iniciais, etc.). A opção *Sair* finaliza o simulador. A fig. 5.2 apresenta o menu *Arquivo* com todas as suas opções.

FIGURA 5.2 – Menu Arquivo

5.1.2 Menu Configurações

Este menu permite definir e configurar o SMA reativo desejado pelo usuário. Ele subdivide-se em sete componentes principais, como visto na fig. 5.3, que serão discutidos a seguir.

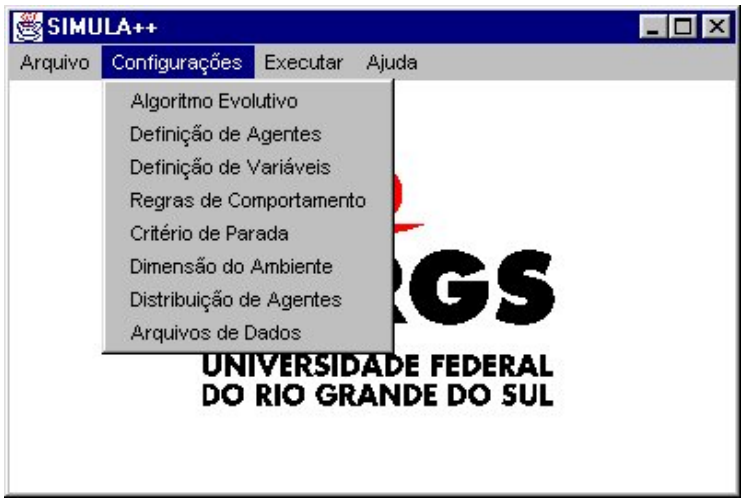


FIGURA 5.3 – Menu Configurações

- Algoritmo Evolutivo:

Este componente (fig. 5.4) define se será utilizado o Algoritmo Evolutivo durante o processo de simulação e quais as características utilizadas por este. As características que podem ser modificadas pelo usuário são a escolha da forma de recombinação (externa de um ponto, externa de dois pontos ou interna), se há elitismo ou não, e os percentuais de mutação utilizados pelo Algoritmo Evolutivo. Além destas características, já discutidas no capítulo anterior, pode-se configurar uma variável que indica o percentual de agentes com regras pré-definidas (ou seja, que foram definidas no componente Regras de Comportamento, a ser visto neste capítulo) que serão simulados

pelo sistema. Um percentual igual a zero indica que todos os agentes iniciarão a execução com regras de comportamento aleatórias.

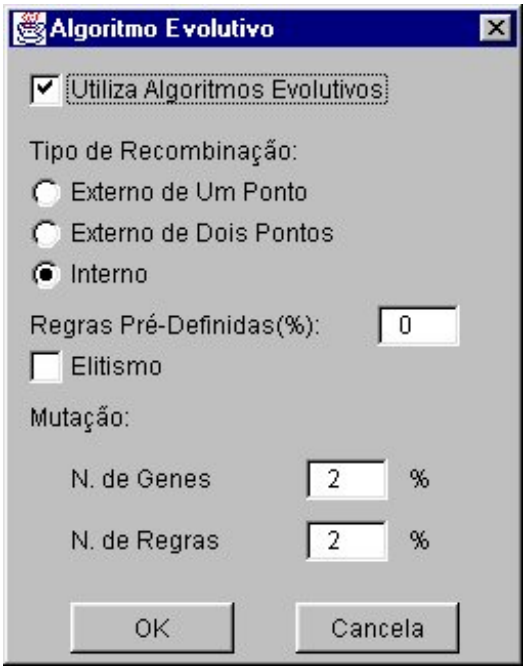


FIGURA 5.4 – Configurações do Algoritmo Evolutivo

- Definição de Agentes

Neste componente (fig. 5.5) são definidos os tipos de agentes presentes no sistema, assim como o ícone representativo de cada tipo de agente. As características necessárias para a definição dos agentes são listadas a seguir.

- Nome da Classe de Agente: nome que identificará o tipo de agente;
- Número de Agentes: número inicial de agentes deste tipo que estarão presentes no sistema;
- Área de Percepção: raio de alcance da visão do agente;
- Energia e Carga: variáveis internas que podem ser utilizadas pelo usuário;
- Tempo Máximo Vida: tempo máximo de simulação para cada agente;
- Tempo Mínimo para Fertilização: tempo mínimo necessário para que o agente possa gerar descendentes;
- Tempo Entre Fertilizações: intervalo de tempo para que o agente gere um novo descendente;
- Bloqueia Percepção: os agentes que possuem esta opção marcada funcionam como uma espécie de muro visual, que impede a percepção dos outros agentes além da barreira formada por este tipo de agente;
- Imagem: clicando no ícone mostrado, será aberto o componente Imagem do Agente, como visto na fig. 5.6, que é utilizado para o desenho representativo para este tipo de agente.

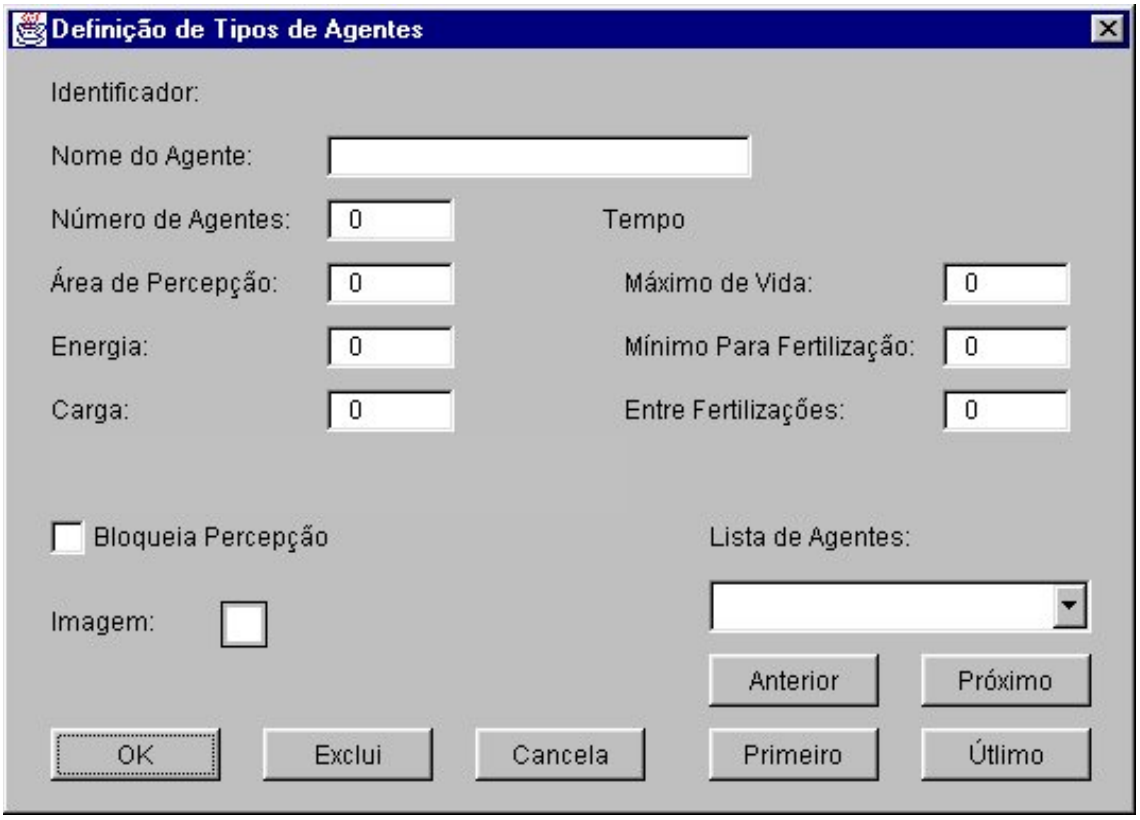


FIGURA 5.5 – Definição dos Tipos de Agentes

Os botões do componente mostrado na fig. 5.5, oferecem as seguintes funcionalidades:

- Botão *OK*: botão de confirmação das definições efetuadas pelo usuário;
- Botão *Anterior*: visualiza o tipo de agente definido anteriormente pelo usuário;
- Botão *Próximo*: visualiza a definição do próximo tipo de agente;
- Botão *Primeiro*: visualiza o primeiro tipo de agente definido pelo usuário;
- Botão *Último*: visualiza o último tipo de agente definido pelo usuário;
- Botão *Excluir*: exclui a definição de tipo de agente;
- Botão *Cancela*: não considera as definições efetuadas pelo usuário.
- Lista de Agentes: visualiza diretamente o tipo de agente selecionado.

A fig. 5.6 apresenta o componente utilizado para o desenho da imagem que irá representar este tipo de agente.

FIGURA 5.6 – Definição da Imagem para o Tipo de Agente

- Definição de Variáveis

Este componente é utilizado pelo usuário para definir variáveis globais para o sistema. As variáveis podem ser de três tipos: *int*, *char* e *string*. Para cada variável é definido um nome, um tipo e um valor inicial, que deve ser condizente com o tipo escolhido. As variáveis globais são utilizadas, usualmente, nas regras de comportamento, no campo Pós-Condição, para armazenar ou atualizar algum valor importante que o usuário deseja visualizar durante a simulação. A fig. 5.7 apresenta a caixa de diálogo para a definição de variáveis.

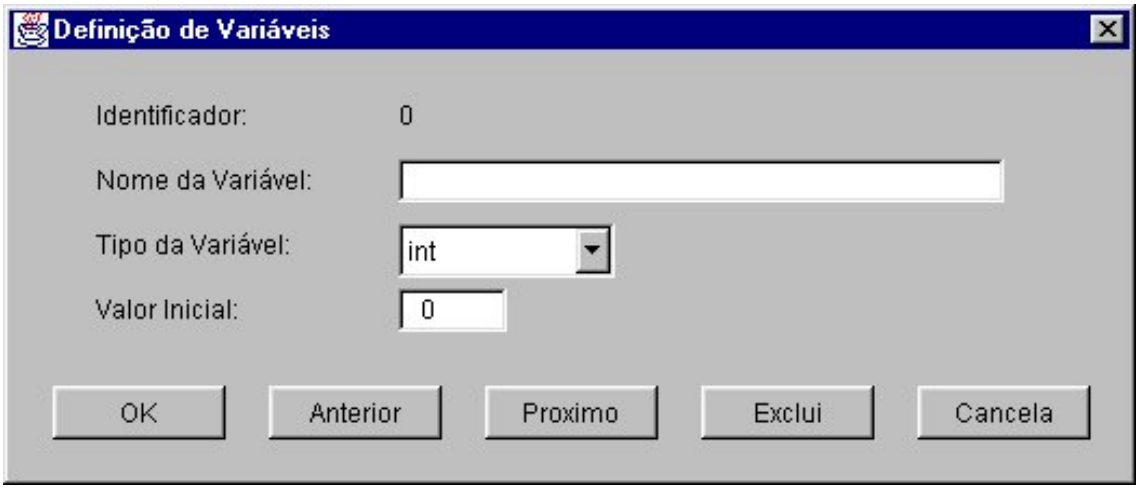


FIGURA 5.7 – Definição de Variáveis

- Regras de Comportamento

Neste componente são construídas todas as regras de comportamento para cada tipo de agente definido anteriormente. As regras de comportamento são construídas a partir de uma coleção de regras pré-definidas, sendo que estas podem ser concatenadas através dos operadores lógicos AND, OR e NOT. Para cada regra de comportamento é obrigatória a definição da Prioridade (de 0 a n, sendo 0 a regra de maior prioridade), da

Pré-Condição e da Ação Ativada. O campo de Pós-Condição não é obrigatório. Normalmente este é utilizado para a atualização de variáveis de controle global definidas pelo usuário. A fig. 5.8 mostra a caixa de diálogo para a definição de comportamentos.

FIGURA 5.8 – Definição de Comportamentos

No campo Agente, é selecionado qual tipo de agente terá suas regras alteradas. Os botões oferecem as seguintes funcionalidades:

- Botões *Alterar*: os botões alterar são utilizados para executar outros componentes, mostrados à seguir, que são utilizados para definir as pré-condições para a aplicação de uma regra, a ação ativada desta regra e as atualizações de variáveis de controle.
- Botão *Primeiro*: visualiza a primeira regra definida para aquele tipo de agente;
- Botão *Último*: visualiza a última regra definida para aquele tipo de agente;
- Botão *OK*: botão de confirmação das definições efetuadas pelo usuário;
- Botão *Anterior*: visualiza a regra definida anteriormente pelo usuário;
- Botão *Próximo*: visualiza a próxima regra;
- Botão *Excluir*: exclui a regra;
- Botão *Cancela*: não considera as definições efetuadas pelo usuário.

A alteração das regras de comportamento utiliza um conjunto de regras pré-definidas, que estão listadas no Anexo 1. A caixa de diálogo que aparece na fig. 5.9 demonstra como são construídos os componentes de condição para a regra de comportamento que está sendo definida.

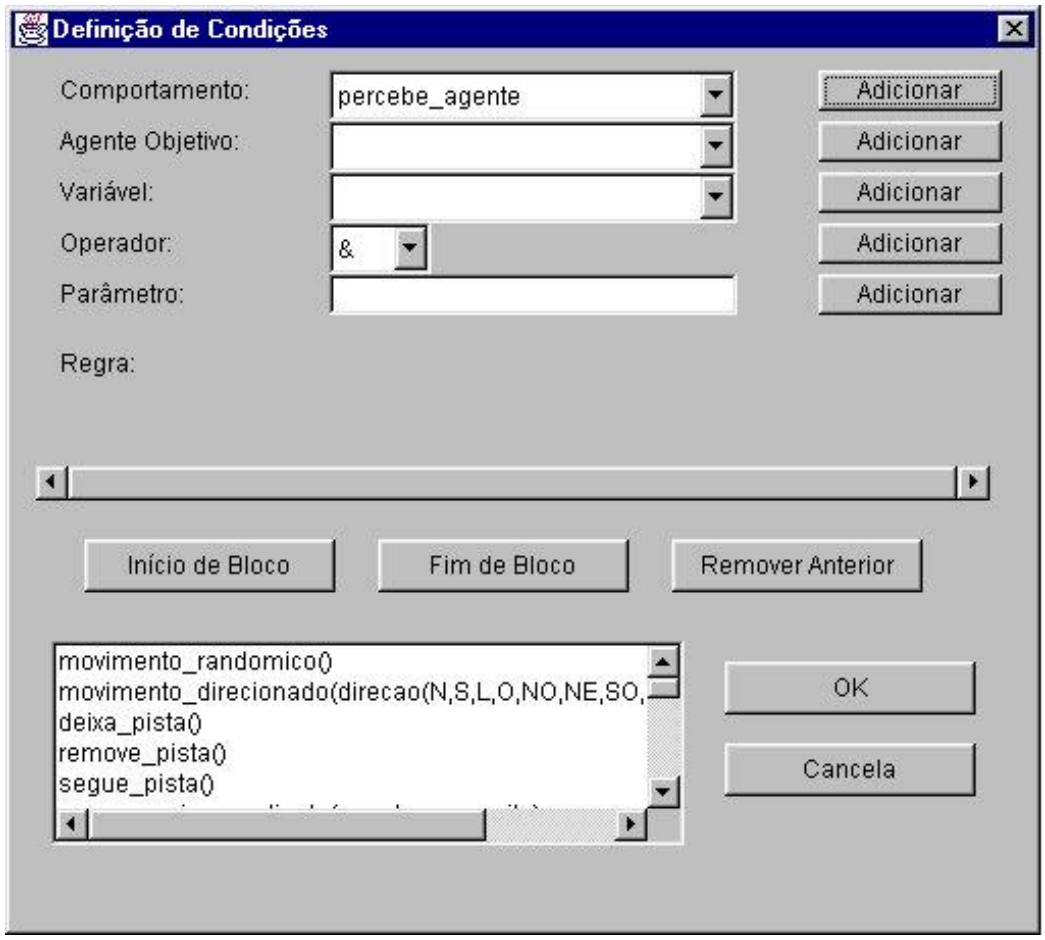


FIGURA 5.9 – Definição de Condições

Esta caixa de diálogo possui cinco elementos principais, que são utilizados para a construção da regra:

- Comportamento: neste campo são escolhidos os comportamentos dentre os pré-definidos no sistema. Cada comportamento pode possuir um ou mais parâmetros, que estão explicados na lista que aparece na parte de baixo da caixa de diálogo. Para adicionar o comportamento à regra, deve-se pressionar o botão *Adicionar* que fica ao lado do campo. Este procedimento é válido para todos os demais elementos.
- Agente objetivo: neste campo é escolhido um agente, entre todos os agentes definidos pelo sistema, que pode ser utilizado como parâmetro do comportamento escolhido.
- Variável: neste campo é escolhida uma variável, entre todas as pré-definidas no ambiente e as definidas pelo usuário, que pode ser utilizada como parâmetro do comportamento escolhido ou como um operando.
- Operador: operadores lógicos e aritméticos válidos para a construção de regras (&, |, !, >, <, >=, <=, =, +, -, *, /).
- Parâmetro: parâmetro de comparação, escolhido pelo usuário, que deve ser do mesmo tipo da variável a qual ele vai ser comparado.

Os botões oferecem as seguintes funcionalidades:

- Botão *Adicionar*: estes botões adicionam o parâmetro definido à sua esquerda na regra que está sendo construída, que pode ser visualizada logo abaixo, no campo Regra;
- Botão *Início de Bloco*: adiciona um novo início de bloco lógico, através da inserção de um abre-parênteses na regra;
- Botão *Fim de Bloco*: finaliza um bloco lógico, através da inserção de um fecha-parênteses na regra;
- Botão *Remover Anterior*: remove o elemento imediatamente anterior ao corrente;
- Botão *OK*: botão de confirmação das definições efetuadas pelo usuário;
- Botão *Cancela*: não considera as definições efetuadas pelo usuário.

A fig. 5.10 mostra a caixa de diálogo para a definição da ação da regra que está sendo construída. O funcionamento deste componente é igual ao componente de definição de condições, exceto pela lista de comportamentos pré-definidos, que diferem devido à natureza de percepção na definição das pré-condições e de ação na definição da ação executada.

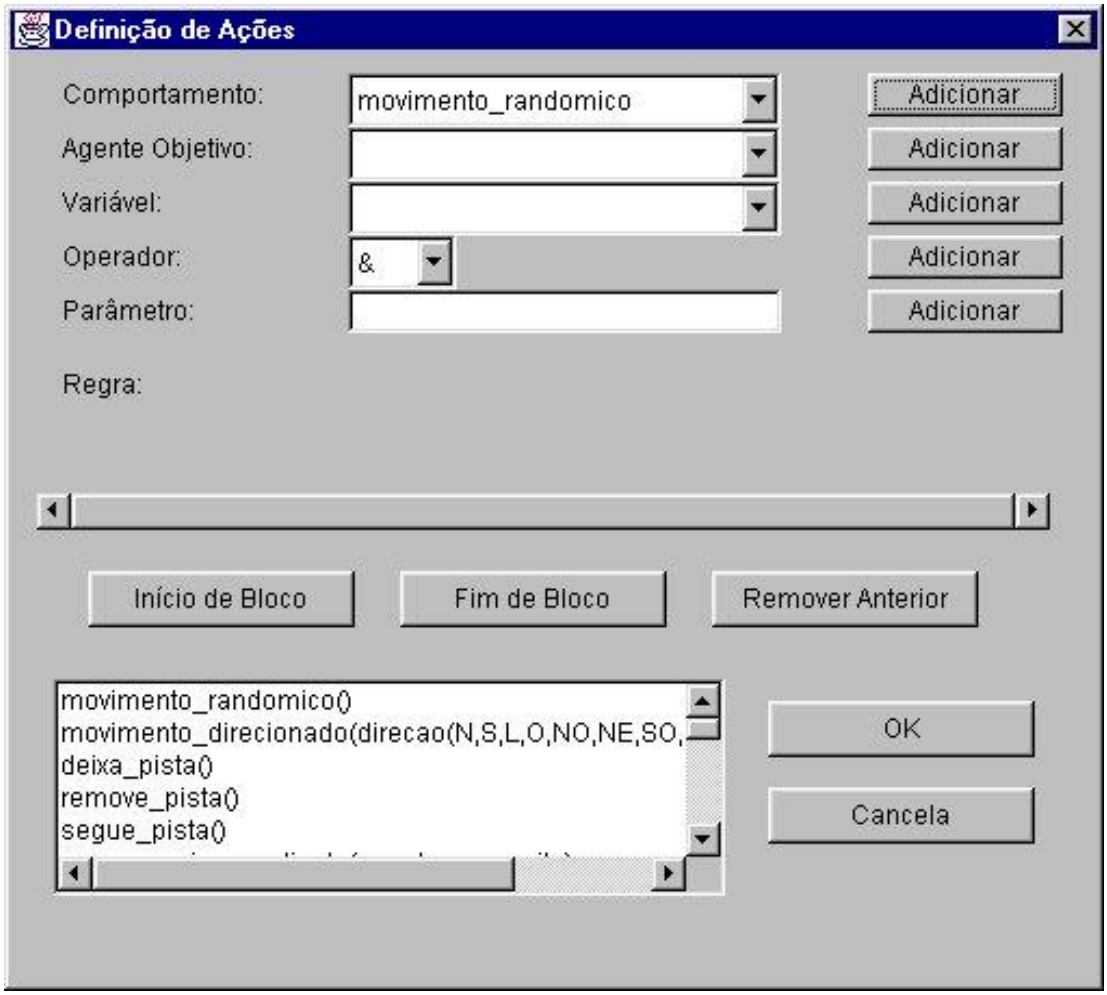


FIGURA 5.10 – Definição de Ações

- Critério de Parada

A fig. 5.11 mostra a caixa de diálogo para a definição do critério de parada para o SMAR a ser simulado. O usuário define uma condição para finalizar a execução do processo, utilizando as variáveis definidas por ele próprio na caixa de diálogo *Definição de Variáveis* ou as variáveis pré-definidas para cada classe de agente. Estas variáveis podem ser utilizadas, através de operadores lógicos e aritméticos, para construir expressões que são testadas a cada novo ciclo de simulação.

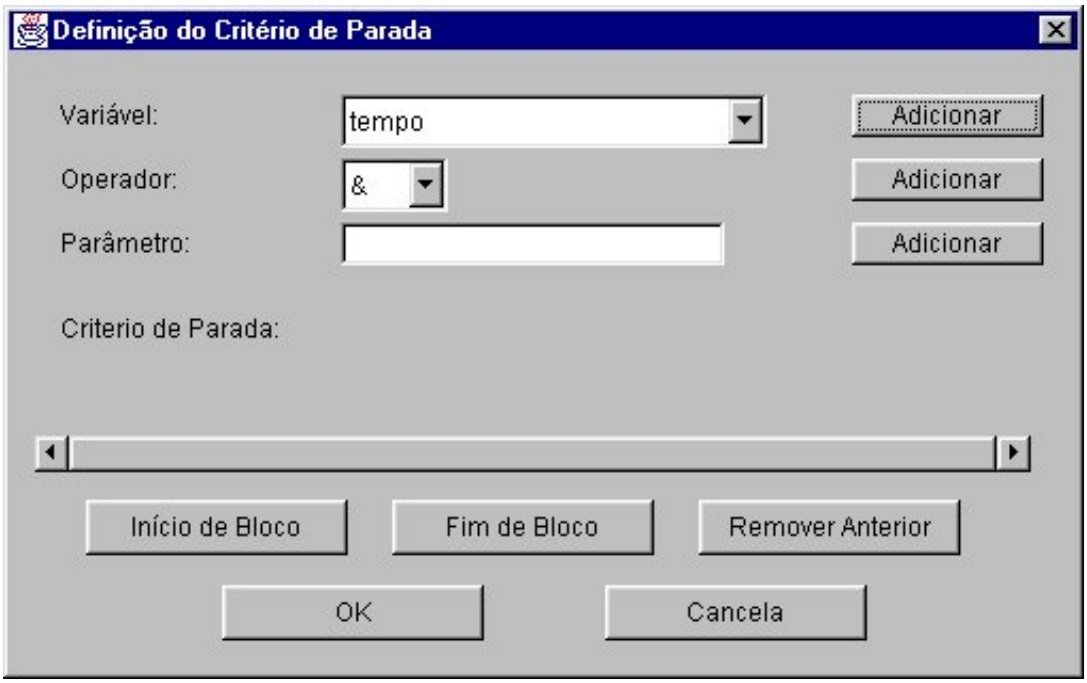


FIGURA 5.11 – Definição do Critério de Parada

Os botões oferecem as seguintes funcionalidades:

- Botões *Adicionar*: estes botões adicionam o parâmetro definido à sua esquerda na expressão que está sendo construída, que pode ser visualizada logo abaixo, no campo Critério de Parada;
- Botão *Início de Bloco*: adiciona um novo início de bloco lógico, através da inserção de um abre-parênteses na regra;
- Botão *Fim de Bloco*: finaliza um bloco lógico, através da inserção de um fecha-parênteses na regra;
- Botão *Remover Anterior*: remove o elemento imediatamente anterior ao corrente;
- Botão *OK*: botão de confirmação das definições efetuadas pelo usuário;
- Botão *Cancela*: não considera as definições efetuadas pelo usuário.

- Dimensão do Ambiente

Este componente, representado na fig. 5.12, é utilizado para a definição das dimensões do ambiente dos agentes. O ambiente de simulação é representado por uma

matriz *n x m*, onde o usuário pode definir o número de linhas e número de colunas. Caso não seja definido, o programa assumirá um valor pré-determinado de 25x25.

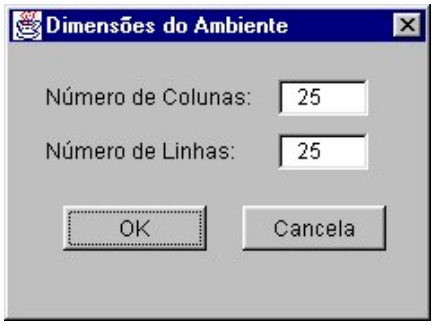


FIGURA 5.12 – Definição das Dimensões de um Ambiente

Os botões oferecem as seguintes funcionalidades:

- Botão *OK*: botão de confirmação das definições efetuadas pelo usuário;
- Botão *Cancela*: não considera as definições efetuadas pelo usuário.

- Distribuição de Agentes

Nesta caixa de diálogo são definidas as posições relativas na matriz para cada um dos agentes do sistema. Caso estas posições não sejam declaradas, o sistema distribuirá os agentes aleatoriamente.

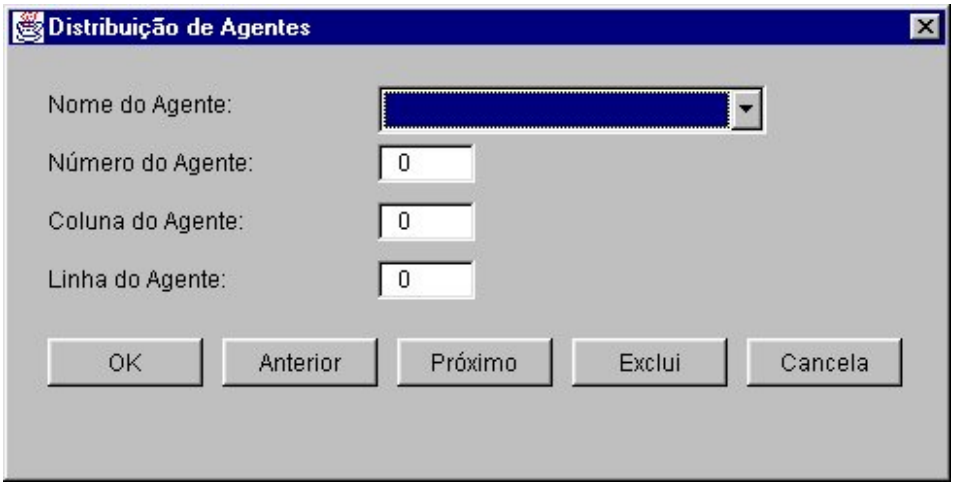


FIGURA 5.13 – Definição das Posições dos Agentes

Para cada agente, devem ser preenchidos os seguintes campos:

- Nome do agente: tipo de agente já definido pelo usuário;
- Número do agente: número de identificação do agente, que varia entre *0* e *n-1*, sendo *n* o número de agentes deste tipo, definido no componente Definição dos Tipos de Agentes.

Os botões oferecem as seguintes funcionalidades:

- Botão *OK*: botão de confirmação das definições efetuadas pelo usuário;
- Botão *Anterior*: visualiza a posição definida pelo usuário para o agente anterior;
- Botão *Próximo*: visualiza a próxima definição de posição;
- Botão *Excluir*: exclui a definição ataul;
- Botão *Cancela*: não considera as definições efetuadas pelo usuário.

- Arquivos de Dados

Este componente é utilizado para configurar os nomes para os arquivos de dados gerados durante a simulação. Para cada classe de agente definido é gerado um arquivo, com o mesmo nome da classe do agente, que contém todas as regras de comportamento para cada agente gerado deste tipo. Além destes arquivos, são gerados o Arquivo de Quantidades e o Arquivo de Adaptação, que armazenam, a cada 10 passos de simulação, respectivamente, o número de agentes ativos e o maior grau de *fitness*, para cada tipo de agente. A fig. 5.14 mostra o componente para a definição dos Arquivos de Execução.

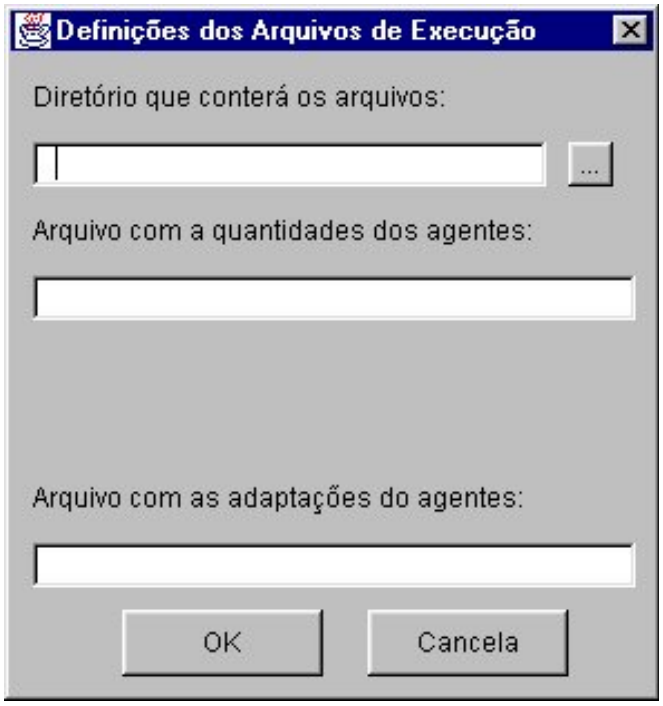


FIGURA 5.14 – Definição dos Arquivos de Execução

Para cada simulação, devem ser preenchidos os seguintes campos:

- Diretório: diretório em que deverão ser gravados os arquivos gerados;
- Arquivo com as quantidades dos agentes: nome do arquivo que conterá a quantidade de agentes ainda ativos, para cada classe de agentes, a cada dez passos de simulação;
- Arquivo com o grau de adaptação dos agentes: nome do arquivo que conterá o grau de *fitness* para cada tipo de agente definido pelo usuário, a cada dez passos de simulação.

Os botões oferecem as seguintes funcionalidades:

- Botão *OK*: botão de confirmação das definições efetuadas pelo usuário;
- Botão *Cancela*: não considera as definições efetuadas pelo usuário.

5.1.3 Menu Executar

Este menu executa a simulação do SMA definido pelos componentes de configuração descritos na seção anterior.

FIGURA 5.15 – Menu Executar

A simulação é realizada no ambiente com as dimensões definidas pelo usuário. Um exemplo de execução de uma simulação aparece na fig. 5.16.

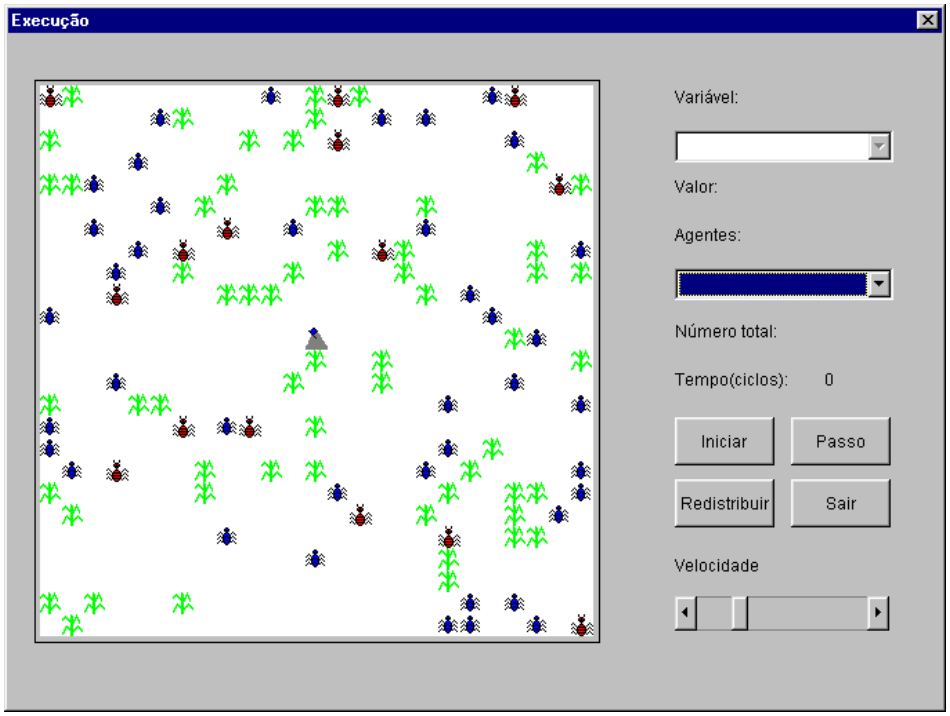


FIGURA 5.16 – Ambiente de Simulação

No campo Variável estão disponíveis todas as variáveis definidas pelo usuário, cujos valores podem ser visualizados dinamicamente, em tempo de execução. O número de agentes de cada tipo também pode ser visualizado através do campo Agentes. Logo abaixo aparece o tempo de execução (número de ciclos já executados pelo simulador).

Os botões oferecem as seguintes funcionalidades:

- Botão *Iniciar/Parar*: inicia ou para a simulação do sistema;
- Botão *Passo*: executa a simulação passo a passo. É necessário pressionar o botão para cada novo passo de simulação;
- Botão *Redistribuir*: reinicia a simulação do sistema, reinicializando todas as variáveis do sistema;
- Botão *Sair*: fecha esta caixa de diálogo;
- Velocidade: barra de rolagem utilizada para aumentar ou diminuir a velocidade da simulação.

5.1.4 Menu Ajuda

Contém informações sobre a utilização do sistema, caso o usuário tenha alguma dúvida. Ainda não implementado.

FIGURA 5.17 – Menu Ajuda

5.2 Funcionalidades do Sistema

O protótipo desenvolvido, que foi apresentado na seção anterior, apresenta um funcionamento simples e consistente para a construção de modelos de sistemas multiagentes reativos. Didaticamente, pode-se definir um roteiro para a construção e simulação de modelos multiagentes no sistema Simula, através de três etapas:

5.2.1 Definição dos Agentes

A definição dos agentes compreende a construção das classes de agentes através da caixa de diálogo apresentada na fig. 5.5, e seus comportamentos (fig. 5.8). Nesta etapa devem-se observar os seguintes aspectos:

- Algoritmo Evolutivo: se a simulação utilizar o algoritmo evolutivo, devem ser observadas as características que definem o comportamento evolucionário dos agentes, através das variáveis Tempo Máximo de Vida, Tempo Mínimo para Fertilização, Tempo entre Fertilizações e a quantidade de Energia Inicial deste tipo de Agente, definidas no capítulo 4;
- Comportamentos: os comportamentos para as classes de agentes estão fortemente ligados às características do modelo que será simulado. Caso a simulação parta de comportamentos aleatórios, não é necessário definir um conjunto de comportamentos para cada classe de agentes.

Adicionalmente, pode ser necessário a criação de variáveis auxiliares, através da janela apresentada na fig. 5.7, para a formulação de procedimentos e comportamentos específicos, dependentes do modelo a ser simulado.

5.2.2 Definição de uma Instância de Simulação

A definição do ambiente pode ser dividida nas seguintes tarefas:

- Tamanho do ambiente: a dimensão do ambiente é de importância crucial para os modelos de SMAR. Define-se o tamanho do ambiente (fig. 5.12) através das variáveis *Número de Linhas e Colunas* da grade bidimensional que irá representar o ambiente de simulação;
- Posicionamento dos agentes: se for necessário para o modelo um posicionamento fixo para os agentes, ou para apenas alguns agentes, é necessário definir para cada agente a sua coordenada cartesiana no sistema (fig 5.13). Os demais agentes terão sua posição definida aleatoriamente no início da simulação;
- Arquivos de dados: para cada simulação, é necessário trocar os nomes dos arquivos de dados, para que o sistema possa armazenar os parâmetros e resultados da simulação de forma adequada (fig. 5.14);
- Opcionalmente, pode-se definir um critério de parada para a simulação. Este critério, que é dependente do modelo adotado, pode ser redefinido para cada nova simulação. Se o critério de parada não for definido, a simulação só será finalizada através de um comando do usuário.

5.2.3 Execução da Simulação

Após a definição de todos os parâmetros, pode-se iniciar a simulação (fig. 5.15). Uma simulação pode ser iniciada e interrompida pelo usuário a qualquer momento. Se uma simulação for reiniciada a partir de uma pausa, os arquivos de dados serão mantidos. Se a simulação for parada, os arquivos de dados serão reinicializados a partir do passo zero, perdendo toda a informação armazenada. Para manter dados acerca de diversas simulações, é necessário trocar os nomes dos arquivos de dados (fig. 5.14).

6 Estudo de Caso: o problema dos Robôs Mineradores

O problema dos Robôs Mineradores foi desenvolvido como um modelo para a funcionalidade emergente de um Sistema Multiagente Reativo em [STE 90]. O problema possui as seguintes características: dado um certo ambiente, desconhecido, deve-se encontrar depósitos de minerais e estes devem ser levados a uma base central, também situada neste ambiente. Não existe nenhuma informação sobre as características do ambiente.

Este problema pode ser facilmente modelado para um sistema multiagente reativo, através da representação do ambiente por uma grade bidimensional, onde a posição dos agentes robôs, do depósito de minerais e da base podem ser representadas por coordenadas cartesianas deste ambiente. Os agentes mineral e base são fixos no ambiente, ou seja, suas posições não mudam no decorrer da simulação, enquanto que os agentes robôs são caracterizados por agentes reativos, que movimentam-se livremente pelo ambiente, procurando o mineral e trazendo este à base.

A fig. 6.1 apresenta uma representação genérica dos agentes envolvidos no problema dos Robôs Mineradores.

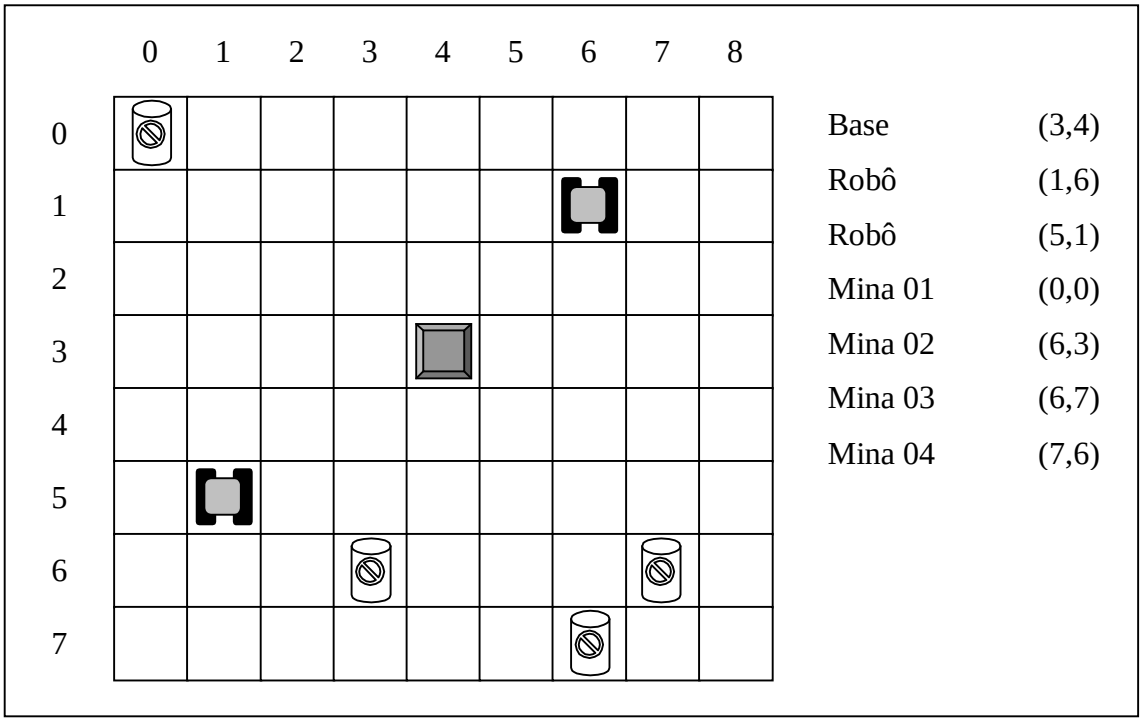


FIGURA 6.1 – Problema dos Robôs Mineradores

O problema dos Robôs Mineradores tornou-se um problema clássico na Inteligência Artificial Distribuída, sendo ainda largamente estudado. Diversos métodos e algoritmos podem ser encontrados na literatura, além de inúmeras variações do mesmo tema central. Dentre estes podemos citar: utilização de marcas [DRO 93], arquitetura de subordinação [BRO 86] e *Fungus Eater* [PFE 96].

O sistema multiagente desenvolvido foi baseado no modelo apresentado em [DRO 93]. A modelagem deste problema por [DRO 93] foi concebida como uma forma para a apresentação e resolução coletiva de problemas.

6.1 Características do Problema

O modelo desenvolvido possui as mesmas características apresentadas em [DRO 93]. O modelo é formado por um ambiente bidimensional, definido por uma grade (100x100), onde cada agente só pode ocupar um espaço desta grade. Deste modo, a posição do agente é fornecida pelas coordenadas cartesianas do ambiente, tal como visto na fig. 6.1 da seção anterior. Existem três tipos de agentes envolvidos no modelo:

- Base: situado na região central do ambiente, a base é utilizada como armazém dos minerais coletados pelos robôs. A base é fixa, não podendo deslocar-se no decorrer da simulação.
- Mineral: existem três minas espalhadas pelo ambiente, a uma distância fixa de 40 casas da Base. Cada mina possui 100 unidades de mineral. Assim como a Base, a mina também não pode deslocar-se de forma autônoma, permanecendo na mesma posição durante a simulação.
- Robôs: são considerados os agentes ativos da simulação. Movimentam-se por todo o ambiente em busca do mineral. Quando o encontram, devem retornar à base para descarregar a sua carga. Um robô só pode carregar uma unidade de mineral de cada vez.

Os Robôs possuem dois sensores de percepção. O primeiro é utilizado para encontrar as minas com os minerais. Este sensor possui um alcance limitado, definido como uma circunferência de raio dois (duas células da matriz). O segundo sensor é utilizado para encontrar a base, após encontrar o mineral. A base emite um sinal que pode ser percebido pelo agente até uma distância de raio 40.

O número de agentes Robôs envolvidos no sistema pode ser variado, mas imutável durante a simulação, enquanto que os demais agentes possuem um número fixo (três agentes Mina e um agente Base).

6.2 Modelagem dos Agentes Robôs *Petit Poucet*

Para efeitos de comparação de desempenho, é reproduzido na fig. 6.2 o gráfico obtido pelas simulações dos Robôs Mineradores, denominados *Petit Poucet* em [DRO 93], para um ambiente bastante similar ao apresentado neste trabalho.

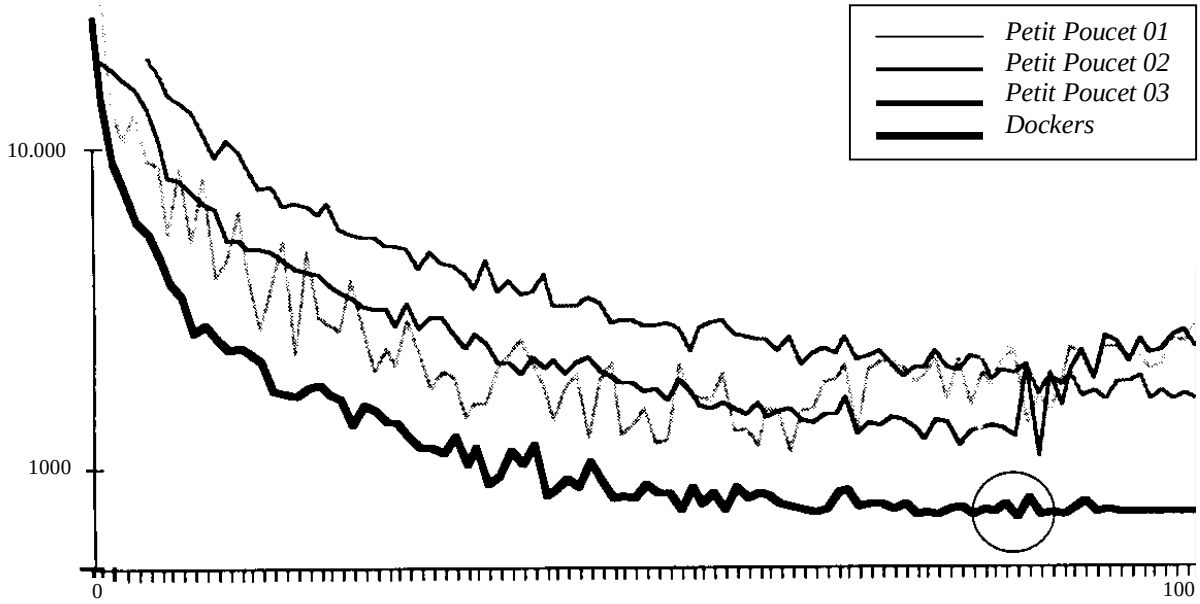


FIGURA 6.2 – Número de Robôs Mineradores x Tempo para Coleta de Minerais [DRO 93]

No gráfico da fig. 6.2, são apresentados quatro conjuntos de simulações, para quatro tipos de agentes específicos. As simulações apresentadas em [DRO 93] utilizaram agentes imutáveis, ou seja, os seus comportamentos básicos não mudavam durante o tempo de simulação. A definição dos comportamentos dos agentes em [DRO 93] pode ser visualizada nas tabelas 6.1 a 6.4. A prioridade das regras apresentadas está em ordem crescente, em relação ao seu número de identificação.

TABELA 6.1 – Comportamento dos Agentes Robôs *Petit Poucet 01*

Núm.	Condição	Ação
1	Descarregado E Não Percebe Mineral E Não Percebe Pista	Movimento Aleatório
2	Descarregado E Atinge Mineral	Carrega Mineral
3	Carregado E Não Atinge Base	Retorna à Base E Deixa Pista
4	Carregado E Atinge Base	Descarrega Mineral
5	Descarregado E Percebe Mineral	Vai em Direção ao Mineral
6	Descarregado E Percebe Pista	Segue Pista em Direção ao Mineral

O comportamento global representado pelo conjunto de regras da tab. 6.1 pode ser descrito da seguinte forma: o Robô deve percorrer aleatoriamente o ambiente enquanto não encontrar o Mineral. Após encontrar o Mineral, ele deve retornar à base, deixando uma série de pistas (representadas por um agente estático especial), que forma uma trilha entre a Mina e a Base. Desta forma, qualquer outro Robô que encontrar acidentalmente esta trilha poderá ir diretamente até a Mina, diminuindo o tempo de busca necessário para a coleta de todos os Minerais. É importante observar que as pistas deixadas pelos agentes estão sempre ativas, mesmo quando não há mais Minerais na Mina para serem coletados. Isto faz com que os agentes Robôs sejam levados para uma

região desfavorável, aumentando o tempo necessário para a coleta de todos os minerais. Devido a esta última característica, foi apresentado em [DRO 93] um melhoramento destas regras, tal como pode ser observado na tab. 6.2.

TABELA 6.2 – Comportamento dos Agentes Robôs *Petit Poucet 02*

Núm.	Condição	Ação
1	Descarregado E Não Percebe Mineral E Não Percebe Pista	Movimento Aleatório
2	Descarregado E Atinge Mineral	Carrega Mineral
3	Carregado E Não Atinge Base	Retorna à Base E Deixa Pista
4	Carregado E Atinge Base	Descarrega Mineral
5	Descarregado E Percebe Mineral	Vai em Direção ao Mineral
6	Descarregado E Percebe Pista	Segue Pista em Direção ao Mineral E Remove Pista

A única regra modificada foi a 6. Neste novo conjunto de regras, se o agente Robô encontrar uma pista, ele segue esta pista em direção ao mineral, removendo-a do ambiente de simulação. Enquanto uma Mina contiver Minerais, a trilha é realimentada pelos Robôs. Desta forma, quando todos os Minerais de uma determinada Mina forem coletados, a trilha torna-se obsoleta e é rapidamente eliminada pelos próprios Robôs que a estavam utilizando.

A terceira geração de robôs utilizada em [DRO 93] foi modificada tendo em vista um pequeno problema, que foi observado nas simulações do grupo 02. Quando poucos agentes estavam coletando Minerais de uma determinada Mina, a trilha poderia ser perdida mesmo quando ela ainda era necessária, devido à ação de outros agentes que encontravam a mesma. Para evitar este problema, foi apresentado um terceiro grupo de agentes, cujas regras de comportamento podem ser visualizadas na tab. 6.3.

TABELA 6.3 – Comportamento dos Agentes Robôs *Petit Poucet 03*

Núm.	Condição	Ação
1	Descarregado E Não Percebe Mineral E Não Percebe Pista	Movimento Aleatório
2	Descarregado E Atinge Mineral	Carrega Mineral
3	Carregado E Não Atinge Base	Retorna à Base E Deixa Duas Pistas
4	Carregado E Atinge Base	Descarrega Mineral
5	Descarregado E Percebe Mineral	Vai em Direção ao Mineral
6	Descarregado E Percebe Pista	Segue Pista em Direção ao Mineral E Remove Pista

O comportamento global dos agentes é o mesmo apresentado pelo grupo 02. A única regra modificada foi a 3. Quando um agente carrega o Mineral e começa o seu caminho de retorno à base, o Robô deixa duas pistas no ambiente. Este comportamento aumenta as chances dos demais Robôs de encontrar a pista e, conseqüentemente, a Mina. Quando a Mina estiver vazia a pista obsoleta é rapidamente destruída pela Robôs, da mesma forma apresentada pelo grupo 02.

A quarta geração de agentes, também conhecida como agentes estivadores *(dockers)* foi baseada no comportamento observado pelos trabalhadores das docas. Os agentes estivadores trabalham de forma cooperativa, formando uma fila que inicia-se na mina e vai até a base. O primeiro agente, que encontra-se junto à Mina, carrega um Mineral e o repassa ao seu agente vizinho. O vizinho pega o Mineral e o repassa ao seu vizinho mais próximo e assim sucessivamente, até o último Robô, que descarrega o Mineral na Base. Desta forma, o tempo de coleta de todos os Minerais reduz-se consideravelmente, pois o caminho percorrido pelos agentes é mínimo. A fig. 6.3 representa um esquema dos agentes estivadores. A tab. 6.4 apresenta o conjunto de regras de comportamento para este grupo de agentes Robôs.

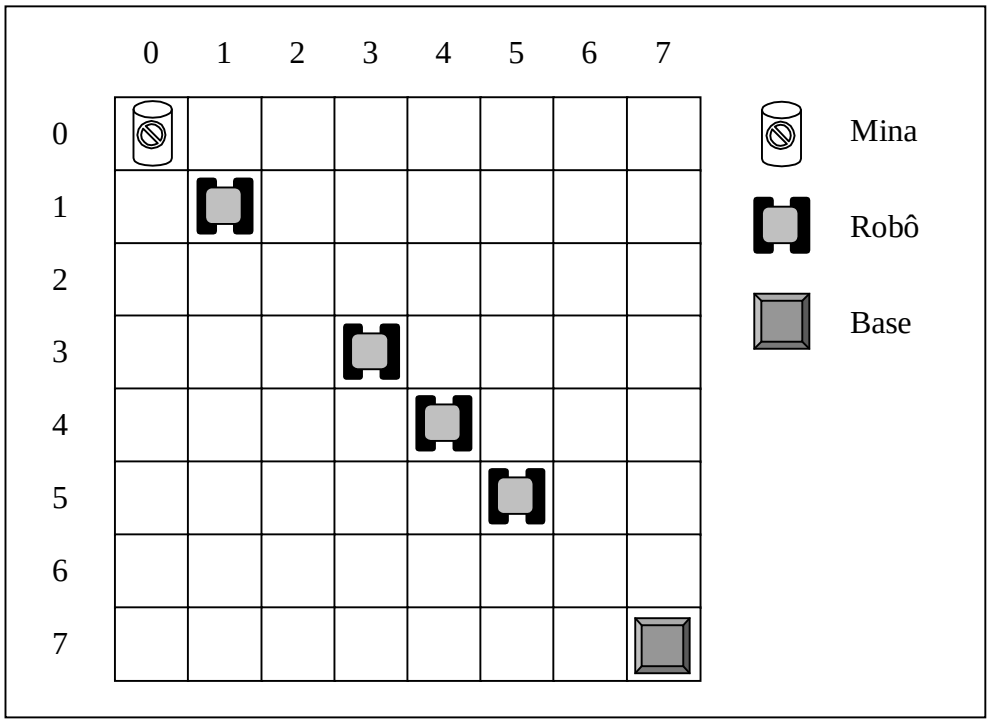


FIGURA 6.3 – Modelo Esquemático dos Robôs Estivadores

TABELA 6.4 – Comportamento dos Agentes Robôs *Dockers*

Núm.	Condição	Ação
1	Descarregado E Não Percebe Mineral E Não Percebe Pista	Movimento Aleatório
2	Descarregado E Atinge Mineral	Carrega Mineral
3	Carregado E Não Atinge Base	Retorna à Base E Deixa Duas Pistas
4	Carregado E Atinge Base	Descarrega Mineral
5	Descarregado E Percebe Mineral	Vai em Direção ao Mineral
6	Descarregado E Percebe Pista	Segue Pista em Direção ao Mineral E Remove Pista
7	Descarregado E Atinge Robô Carregado	Captura Mineral do Robô

A diferença entre os Robôs do grupo *Petit Poucet* 03 e do grupo *dockers* está na adição da regra número 7. Esta regra é responsável pelo comportamento de *fila de carga* que ocorre entre o mineral e a base. Quando um robô descarregado encontra uma pista, ele segue a trilha até encontrar a Mina ou até encontrar um agente carregado. Neste último caso, o Robô descarregado captura o Mineral do Robô carregado e inicia sua caminhada para a base (regra 03). O Robô cuja carga foi capturada torna-se descarregado e retorna para a Mina, seguindo a pista, para carregar um novo Mineral (regra 06). Inicialmente a *fila de carga* contém apenas um Robô (o agente que descobriu a Mina), mas a medida que o tempo passa, os demais agentes poderão se juntar à fila, formando uma cadeia entre a Mina e a Base.

6.3 Modelagem dos Agentes Robôs no Ambiente Simula++

Existem dois comportamentos que foram pré-definidos para todos os Robôs:

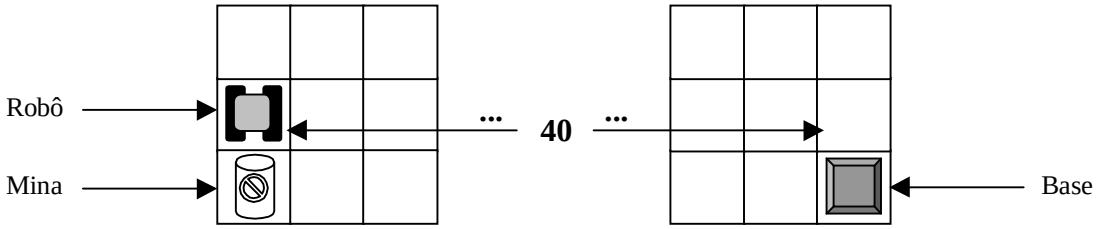
- se o Robô está descarregado e encontra uma mina, ele carrega uma unidade de mineral, diminuindo de 01 a quantidade de minerais da mina;
- se o Robô está carregado e encontra a Base, ele descarrega a unidade de mineral.

Os Robôs iniciam a simulação com um conjunto aleatório de regras de comportamento (cromossomo). Todos os Robôs possuem um tempo de vida máximo de 200 passos de simulação. Ao final deste período, o agente é removido do ambiente, sendo substituído por outro (o número de robôs permanece fixo, ou seja o Tempo de Maturidade Sexual é igual a 200), cujas regras de comportamento são construídas através dos operadores genéticos apresentados no capítulo 4.

O cálculo do *Fitness* (grau de aptidão do agente ao ambiente) é medido através das seguintes regras:

- todo o agente criado é inicializado com *fitness* igual a zero;
- quando um agente encontra uma mina e carrega um mineral, seu grau de *fitness* é aumentado em um ponto;
- quando um agente descarrega um mineral na Base, seu grau de *fitness* é aumentado em dois pontos.

Considerando o tempo de vida dos agentes (200) e a distância entre a Base e as Minas (40), podemos calcular o grau de *fitness* máximo igual a 9. Este valor só pode ser alcançado em uma situação, demonstrada na fig. 6.4.



1. Robô encontra a Mina em x = 0 passos.	TempoVida = x	Fitness = 0+1 = 1
2. Dirige-se imediatamente para a Base.	TempoVida = x + 40	Fitness = 1+2 = 3
3. Volta imediatamente para a Mina.	TempoVida = x + 80	Fitness = 3+1 = 4
4. Dirige-se imediatamente para a Base.	TempoVida = x + 120	Fitness = 4+2 = 6
5. Volta imediatamente para a Mina.	TempoVida = x + 160	Fitness = 6+1 = 7
6. Dirige-se imediatamente para a Base.	TempoVida = x + 200	Fitness = 7+2 = 9

FIGURA 6.4 – Fitness Máximo

Sempre que um agente atinge 200 passos de simulação, ele é substituído por outro agente, denominado agente Filho. As regras de comportamento do agente Filho são formadas pela aplicação dos operadores genéticos definidos no capítulo 4, utilizando o cromossomo do agente de maior grau de *fitness*, calculado pela fórmula apresentada anteriormente, e o cromossomo do agente que está sendo substituído. Desta forma, obtêm-se um bom grau de variabilidade e, ao mesmo tempo, são privilegiados os agentes com maior grau de *fitness*.

6.4 Simulação e Análise dos Resultados

O objetivo da simulação é analisar a eficiência dos Robôs Mineradores em coletar todos os 300 minerais espalhados pelos três agentes Mina. Uma síntese com as características do modelo para o problema dos Robôs Mineradores pode ser visualizada na tabela 6.5. Para estas simulações, foi escolhido o tipo de *crossover* interno de um ponto, pois este apresenta uma maior variabilidade na construção das regras do que o *crossover* externo. A taxa de mutação de 0.02 foi escolhida por ser esta uma taxa usual para os algoritmos evolutivos [DAV 91].

TABELA 6.5 – Características do Modelo dos Robôs Mineradores

Parâmetro	Valor
Número de Agentes Mina	03 agentes com 100 unidades de Mineral cada
Número de Agentes Base	01
Número de Agentes Robôs	Variável (1..100)
Energia Inicial Agente Robô	200
Tipo de Recombinação	Interno de um Ponto
Taxa de Mutação	0.2
Critério de Parada	Número de Minerais igual a zero ou 10000 passos de simulação

A simulação ocorre numa grade 100x100, onde a base encontra-se no centro e os três agentes Mina estão espalhados a uma distância de 40 unidades a partir da base. A posição inicial dos agentes é feita de forma aleatória. O número de agentes Robôs é fixo durante a simulação. A energia dos Agentes Mina e do Agente Robô é infinita, ou seja, estes agentes nunca são eliminados pelo tempo de simulação. No entanto, os agentes Mina são eliminados da simulação se o seu número de minerais chegar à zero.

O número de agentes Robôs foi variado de 1 até 100 unidades. Para cada situação (número de agentes Robôs), foram realizadas 3 simulações, totalizando 300 execuções. Para cada simulação, foram armazenados os seguintes parâmetros:

- identificador do Robô e seu valor de *fitness*, para o agente Robô com maior grau de *fitness*, a cada dez passos de execução;
- número de minerais em cada Mina e somatório do número de minerais, a cada dez passos de execução;
- identificador e Cromossomo de todos os Robôs criados pela simulação, a cada nova geração (200 passos de simulação).

A importância da variação do número de agentes para os fenômenos coletivos, tais como os Sistemas Multiagentes, está diretamente relacionada com o conceito de fenômeno emergente, já discutido na seção 2.1, e em [DRO 93]. A base conceitual dos Sistemas Multiagentes Reativos (seção 3.1.2) e dos comportamentos emergentes como um todo, está na interação e relação entre os agentes e o ambiente que os cerca. Logo, o número de agentes ativos no ambiente influencia a descoberta de novos comportamentos e novas relações entre os agentes envolvidos. No caso específico do problema dos Robôs Mineradores, o número de agentes Robôs influencia na velocidade da coleta dos minerais e no tempo necessário para a descoberta dos agentes Minas.

O gráfico da fig. 6.5 representa o tempo necessário para coletar todos os minerais pelo número de agentes Robôs no ambiente. Cada ponto do gráfico foi plotado pela média de três simulações realizadas com o mesmo número de Robôs. É importante salientar que os dados representados pela fig. 6.5 foram obtidos em simulações onde o algoritmo evolutivo estava sempre presente e as regras eram sempre reinicializadas randomicamente para cada nova simulação. Este grupo de simulações foi denominado Grupo E.

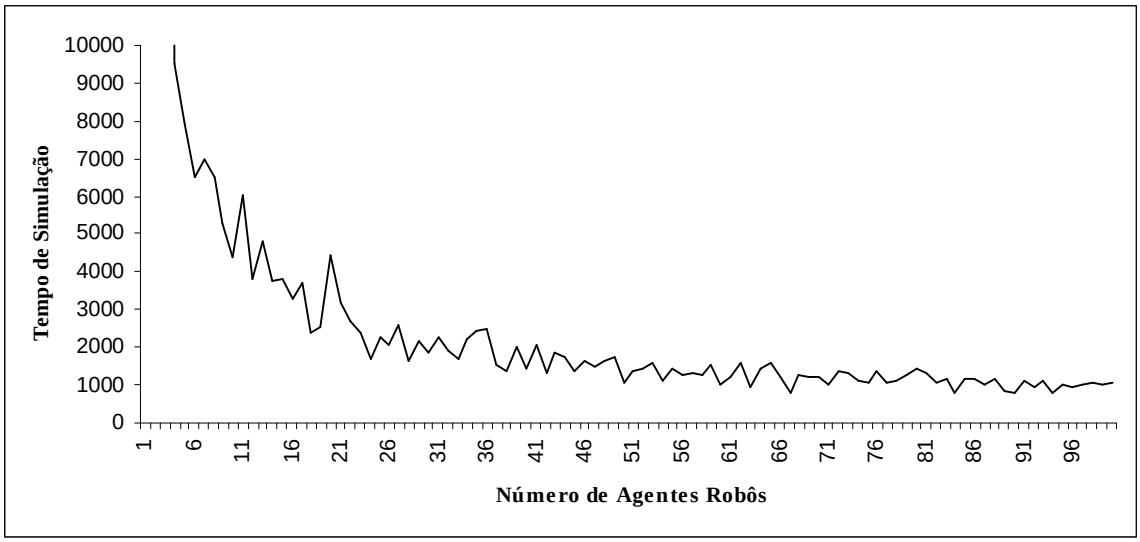


FIGURA 6.5 – Número de Robôs Mineradores Grupo E x Tempo para Coleta de Minerais

A média de tempo para coleta dos minerais, assim como os tempos de simulação máximo e mínimo, podem ser observados na tabela 6.6.

TABELA 6.6 – Simulações dos Robôs Mineradores Grupo E

Parâmetro	Valor
Média para Coleta de Todos os Minerais	2323.7 passos de execução
Tempo Máximo para Coleta dos Minerais	10000 passos de execução (simulações com 1, 2, 3 e 4 agentes)
Tempo Mínimo para Coleta dos Minerais	770 passos de execução (simulação com 95 agentes)

Os resultados obtidos por [DRO 93] mostram o desempenho dos Robôs Mineradores frente a um ambiente desconhecido, de acordo com conjuntos de regras de comportamento pré-definidos. A tab. 6.7 mostra o resultado das simulações dos quatro grupos de agentes de [DRO 93] em comparação com as simulações realizadas no ambiente SIMULA++.

TABELA 6.7 – Performance dos Robôs Mineradores

Parâmetro	Grupo E	*Petit Poucet 1*	*Petit Poucet 2*	*Petit Poucet 3*	*Dockers*
Média de Tempo para Coleta de Todos Minerais	2303,7	3351	5315	3519	1746
Tempo Mínimo para Coleta de Todos Minerais	770	1113	1607	1075	695
Tempo Máximo para Coleta de Todos Minerais	10000	10000	10000	10000	10000
Número de Agentes para Tempo Mínimo	95	64	98	87	84
Número de Agentes para Tempo Máximo	1-4	1-4	1-12	1-7	1-3

A partir da análise do gráfico apresentado na fig. 6.5, foram listadas as regras da tab. 6.8, obtidas pelo algoritmo evolutivo. Estas regras, denominadas de Grupo S, foram geradas pelos agentes de maior grau de *fitness* entre todos os agentes participantes de todas as 300 simulações realizadas. É importante salientar que estas regras foram compiladas pela análise dos agentes com maior grau de *fitness*. Ou seja, nem todas as regras mostravam-se por completo em todos os agentes analisados, assim como algumas regras geradas foram removidas por se mostrarem inócuas.

TABELA 6.8 – Comportamento dos Agentes Robôs de maior Grau de Fitness (Grupo S)

Núm.	Condição	Ação
1	Descarregado E Não Percebe Mineral E Não Percebe Pista E Não Percebe Agente	Movimento Aleatório
2	Descarregado E Não Percebe Mineral E Não Percebe Pista E Percebe Agente	Segue Agente
3	Descarregado E Não Percebe Mineral E Não Percebe Pista E Atinge Agente	Foge de Agente
4	Descarregado E Atinge Mineral	Carrega Mineral
5	Carregado E Não Atinge Base	Retorna à Base E Deixa Pista
6	Carregado E Atinge Base	Descarrega Mineral
7	Descarregado E Percebe Mineral	Vai em Direção ao Mineral
8	Descarregado E Percebe Pista	Segue Pista em Direção ao Mineral

Percebe-se uma semelhança entre estes comportamentos e os apresentados na tab. 6.1 (*Petit Poucet 01*). As maiores diferenças encontram-se no comportamento dos robôs quando estes não estão nas cercanias das Minas. Nestas situações, os robôs, de acordo com a regra 2, tendem a aglutinar-se, formando pequenos grupos de busca. Estes grupos movem-se randomicamente pelo ambiente. Esse agrupamento diminui o espaço de busca em todo o sistema, pois os agentes ficam menos espalhados. Esta situação foi resolvida pelo algoritmo evolutivo através da regra 3, que não permite uma grande aproximação dos agentes, o que restringiria ainda mais o espaço de busca. Uma vantagem interessante neste tipo de estratégia é a maior possibilidade de que todos os agentes participem ativamente do recolhimento dos minerais, após a Mina ter sido descoberta por algum dos Robôs.

Foram realizadas simulações semelhantes às anteriores, utilizando o mesmo ambiente apresentado anteriormente e as regras descobertas pelo algoritmo evolutivo. O gráfico da fig. 6.6 representa o tempo necessário para coletar todos os minerais pelo número de agentes Robôs no ambiente. Cada ponto do gráfico foi plotado pela média de três simulações realizadas com o mesmo número de Robôs. As regras dos robôs não

eram modificadas durante o processo, ou seja, não foi aplicado o algoritmo evolutivo durante as simulações realizadas.

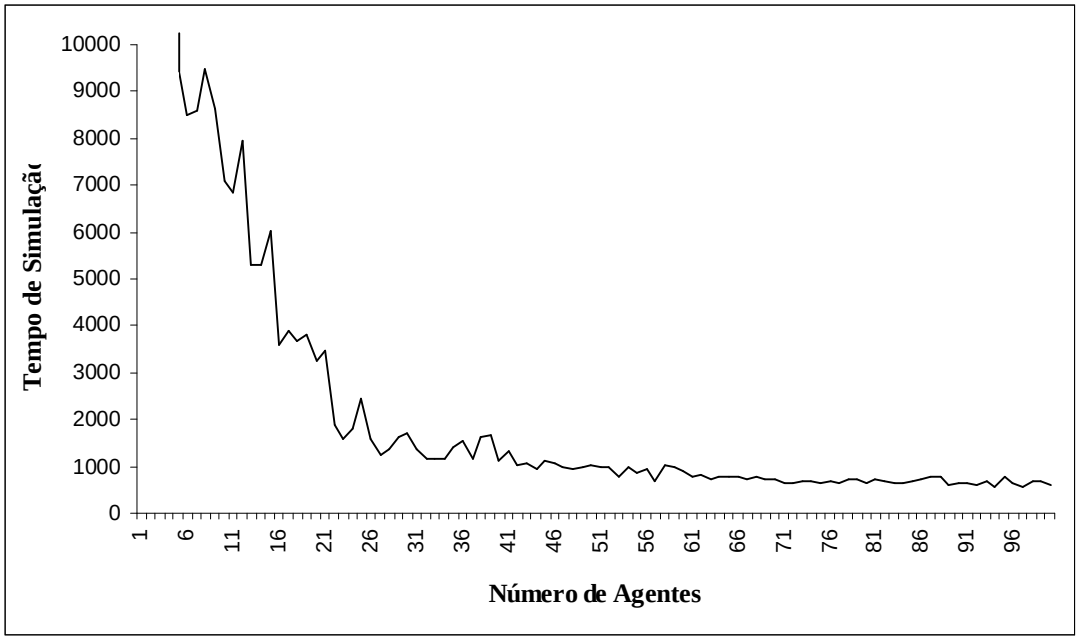


FIGURA 6.6 – Número de Robôs Mineradores Grupo S x Tempo para Coleta de Minerais

Para uma melhor comparação entre os resultados obtidos, a fig. 6.7 apresenta um gráfico onde são apresentados ambos os resultados da fig. 6.5 e 6.6.

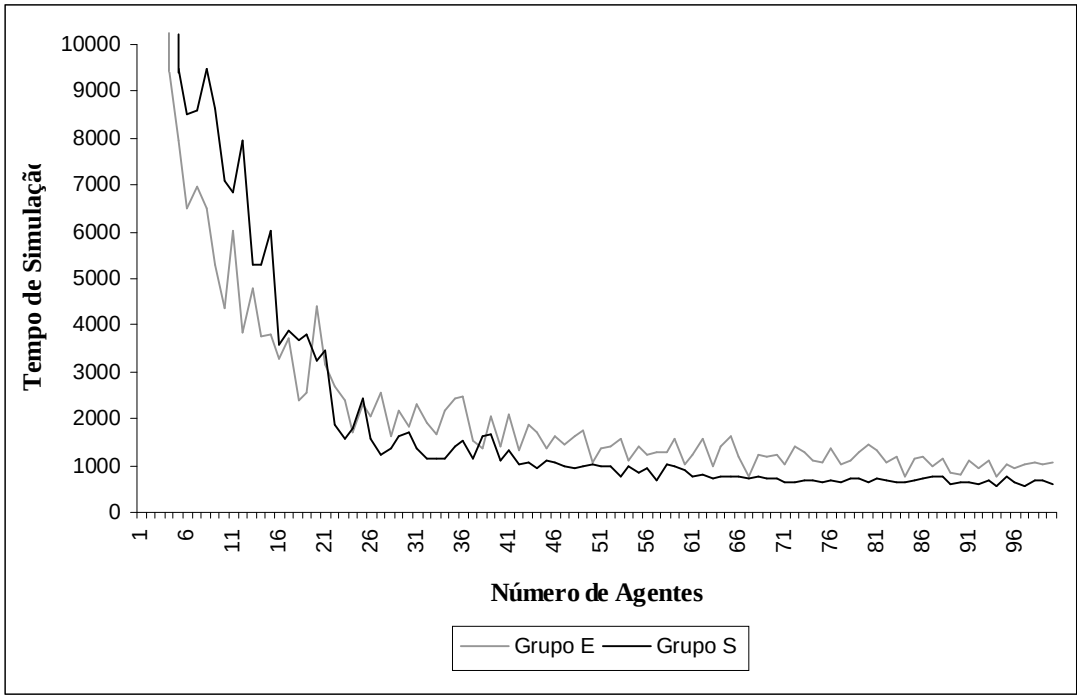


FIGURA 6.7 – Comparação entre dados do Grupo E e o Grupo S

A média de tempo para coleta dos minerais pelo Grupo S, assim como os tempos de simulação máximo e mínimo, podem ser observados na tabela 6.9

TABELA 6.9 – Simulações dos Robôs Mineradores Grupo S

Parâmetro	Valor
Média para Coleta dos Minerais	2216,9 passos de execução
Tempo Máximo para Coleta dos Minerais	10000 passos de execução (simulações com 1, 2, 3, 4, 5 e 6 agentes)
Tempo Mínimo para Coleta dos Minerais	540 passos de execução (simulação com 94)

Ao analisarmos o gráfico da fig. 6.7, pode-se perceber que o conjunto de regras descoberto pelo Algoritmo Evolutivo (Grupo S), apresenta uma performance pobre para um pequeno número de agentes. Este fato pode explicado pelas características das regras 1 e 2 apresentadas na tab. 6.8, que acabam por restringir o espaço de busca global. No entanto, para um maior número de agentes, estas regras apresentaram uma performance superior às demais, devido ao maior número de agentes que participam efetivamente da busca dos minerais.

Um outro fator que contribuiu para o aumento de performance só pode ser percebido visualmente durante as simulações. Após o esgotamento de uma Mina, os agentes que estavam percorrendo a trilha de pistas acabam aglutinando-se na região onde, outrora, havia minerais para serem carregados. Esta situação é bastante comum neste tipo de problema. Com este grande número de agentes reunidos na mesma região, a probabilidade de choques entre os agentes aumenta consideravelmente. Nestes casos, a regra 3 acaba por ajudar no maior espalhamento dos agentes, num tempo bem menor que o praticado por movimentos randômicos. Quanto mais rápido os agentes deslocarem-se desta região infrutífera, maior a possibilidade destes encontrarem novas Minas ou juntarem-se a filas de coleta já existentes.

A semelhança entre as regras encontradas e as apresentadas em [DRO 93] pode ser explicadas por dois fatos: (1) o conjunto de regras pré-definido, apresentado no Anexo 1, favorece a construção de regras baseada no algoritmo de coleta de minerais apresentado em [DRO 93]. Este algoritmo foi baseado na observação de insetos que vivem em sociedades coletivas, tais como as formigas, que deixam uma trilha de perfume especial (feronômio) que indica para as demais formigas onde encontra-se o alimento; (2) a semelhança entre os ambientes de simulação. Ambientes de simulação parecidos tendem a fornecer as mesmas respostas para problemas semelhantes.

No entanto, a evolução das regras não conseguiu encontrar regras semelhantes às dos Robôs *Dockers* (ver. Tab. 6.4). Esta característica decorre da função do *fitness* proposta para este problema, apresentada no início deste capítulo, que apenas recompensa os robôs no momento do encontro do Mineral e seu posterior descarregamento na base. Não há nenhuma punição ou recompensa decorrente do número de passos necessários para que um agente busque o Mineral. Esta condição seria necessária para que o algoritmo evolutivo buscasse meios de diminuir o espaço percorrido pelo agente entre a Mina e a Base.

Considerando o gráfico da fig. 6.7, percebe-se que há uma queda constante no tempo necessário para a coleta de todos os minerais, de acordo com o número de Robôs Mineradores envolvidos. Apesar desta característica ser bastante comum em simulações

semelhantes, existe um fenômeno importante que não ocorreu durante os testes realizados. Quando o número de agentes Robôs aumenta consideravelmente, o tráfego intenso aumenta a probabilidade de choques entre agentes. Quanto maior o número de choques, mais tempo é desperdiçado na coleta de minerais. Este fato, responsável pela queda de performance dos agentes *Petit Poucet*, quando estes trabalham em grupos de 80 ou mais elementos, pode ser melhor visualizado na fig. 6.8.

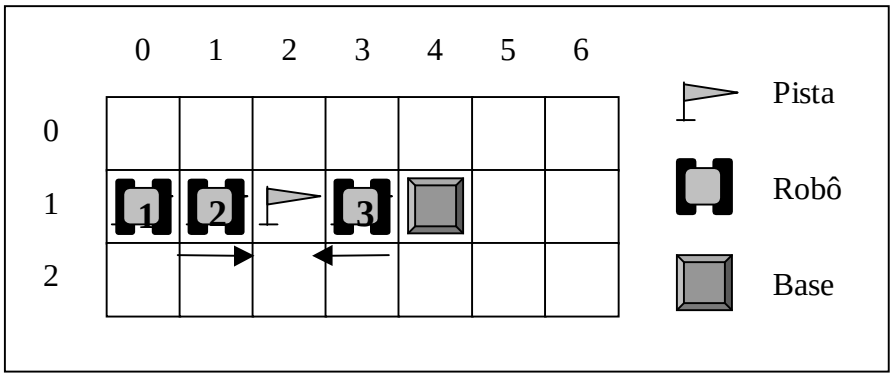


FIGURA 6.8 – Problema do Tráfego de Robôs

No caso apresentado na fig. 6.8, os agentes 1 e 2 estão retornando da Mina, carregados de Mineral, para descarregar sua carga na Base. O agente 3 acaba de descarregar a sua carga e irá iniciar seu retorno à Mina, seguindo a pista que encontra-se no ambiente. Este movimento irá produzir um choque entre os agentes 2 e 3 e, posteriormente, entre os agentes 1 e 3. A cada novo choque, os agentes devem desviar-se do seu obstáculo, aumentando o número de passos necessários para a completa coleta dos Minerais.

O ambiente Simula++ implementa uma forma diferente de trabalho com as pistas. Todo o agente que encontra uma pista, deve segui-la pela sua tangente, ou seja, ao lado da mesma. Desta forma, os agentes que retornam à Base seguem o caminho mais curto (uma linha reta entre a Mina e a Base) e os que seguem a pista até a Mina, traçam um caminho paralelo, evitando o choque entre os Robôs. Esta situação pode ser visualizada na fig 6.9.

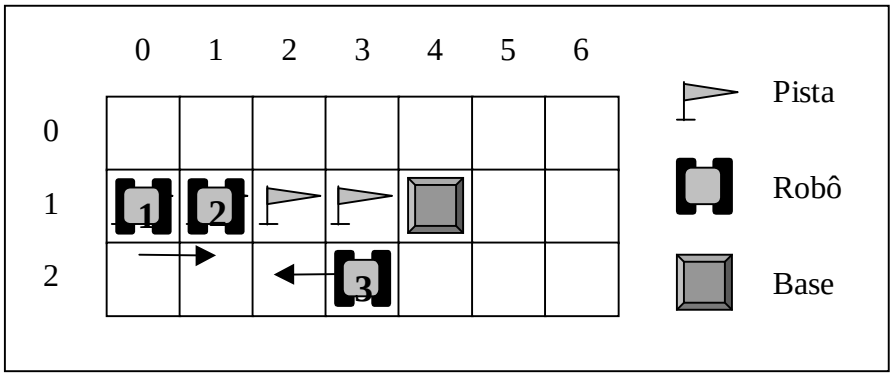


FIGURA 6.9 – O Problema do Tráfego de Robôs no Ambiente Simula++

Considerando os resultados obtidos e a crescente queda no tempo de simulação necessário para a coleta dos Minerais, podemos analisar a simulação sobre um outro

aspecto. Obviamente, quanto maior for o número de agentes envolvidos no processo, a probabilidade de encontrar-se resultados melhores aumenta. No entanto, esta solução pode ser contraproducente se o aumento do número de agentes representar um custo muito alto no sistema. Tal situação pode ocorrer em ambientes reais, com robôs de produção, ou mesmo em sistemas de computação, onde um novo agente pode representar um custo extra em termos de processamento. Nestes casos, a melhor solução é a que representa o melhor custo/benefício entre o número de agentes e o tempo total de processamento.

Para analisarmos este problema, foi plotado o gráfico da fig. 6.10, baseado nas simulações do Grupo S, que representa a quantidade de energia consumida pelos Robôs, calculada como *Número_Robôs * Tempo_Simulação*, pelo número de Robôs presente em cada simulação, até o recolhimento de todos os minerais pelos Robôs.

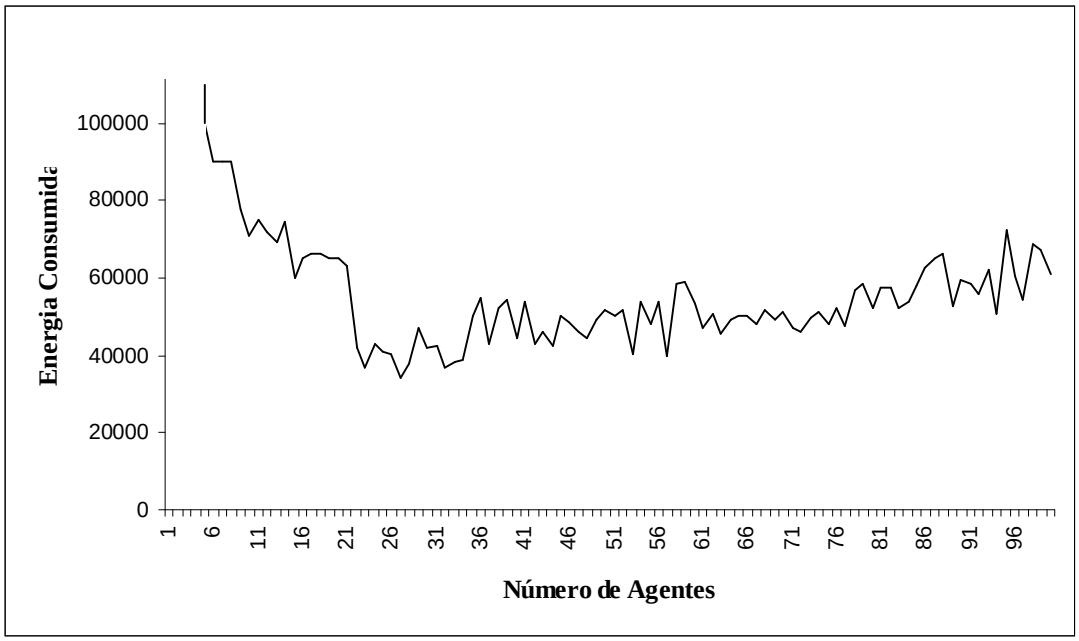


FIGURA 6.10 – Energia Consumida pelos Robôs do Grupo S x Número de Robôs Mineradores

Analisando o gráfico da fig. 6.10, percebe-se que, apesar do tempo de simulação decrescer de acordo com o aumento do número de agentes, a energia utilizada por estes para a coleta dos Minerais também aumenta consideravelmente. Os resultados mais promissores, considerando unicamente a energia consumida, encontram-se nas simulações entre 20 e 40 agentes. É importante salientar que a escolha de uma configuração com menor número de agentes irá, em geral, diminuir a energia consumida e aumentar o tempo total de simulação. Em aplicações onde o tempo é um fator relevante, pode-se definir um tempo máximo de simulação e trabalhar com diferentes números de agentes, buscando uma adequação ao problema específico. A tab. 6.10 apresenta uma comparação entre os resultados obtidos de acordo com as duas análise possíveis: tempo de simulação e energia consumida, para o Grupo S.

TABELA 6.10 – Tempo de Simulação *vs* Energia Consumida pelo Grupo S

Parâmetro	Valor
Média do Tempo de Simulação	2.216,9 passos de execução
Média da Energia Consumida	54.404,47
Tempo Mínimo para Coleta dos Minerais	540 passos de execução (simulação com 94)
Energia Mínima Consumida	34.020 (simulação com 27)
Tempo de Coleta para Simulação 27	1260 passos de execução
Energia Consumida para Simulação 94	50.760

7 Considerações Finais

Os Sistemas Multiagentes (SMA), que formam uma das áreas mais interessantes da Inteligência Artificial Distribuída, concentram-se no estudo de agentes autônomos em um universo multiagente. A principal característica dos SMA está na independência dos agentes através dos seus comportamentos individuais autônomos, ou seja, os próprios agentes decidem sua ações de acordo com seus objetivos e sua percepção do ambiente que os cerca.

Logo, a definição de como as ações serão definidas e implementadas torna-se de fundamental importância para o bom desempenho do sistema. Em SMAR, usualmente, o comportamento de cada agente é definido de forma estática pelo programador, que deve construir as ações dos agentes de acordo com um conjunto de dados de entrada (percepção do ambiente). A idéia principal em um sistema multiagente é que um comportamento global inteligente pode ser alcançado a partir do comportamento individual dos agentes e da interação entre eles. Em um SMAR não é necessário que cada agente seja individualmente inteligente para alcançar um comportamento global inteligente.

O ponto fundamental deste trabalho envolveu os agentes reativos, concentrando-se no problema e na definição dos comportamentos individuais de cada agente. A partir dos estudos realizados acerca da tecnologia dos SMAR, propomos uma nova abordagem na construção dos comportamentos dos agentes, através dos preceitos da Vida Artificial.

Os objetivos da Vida Artificial envolvem a construção de sistemas que sigam as mesmas leis biológicas naturais, analisadas pela teoria evolucionária. A partir do estudo dos aspectos básicos da teoria da Vida Artificial, concentramos nossa atenção nas suas aplicações e modelos existentes, principalmente nos Sistemas de Mundos Virtuais. Estes Sistemas, por sua grande semelhança aos SMA, foram utilizados como a base para a construção de um modelo para evolução dos comportamentos individuais de cada agente.

A principal característica da Vida Artificial está na evolução, ou seja, na capacidade de uma população de indivíduos de se adaptarem ao ambiente em que vivem. Esta capacidade é, usualmente, alcançada através da utilização de algoritmos genéticos ou algoritmos evolutivos. Estes algoritmos foram estudados para uma melhor compreensão dos sistemas de Vida Artificial e sua posterior utilização em sistemas multiagentes reativos.

A partir dos estudos realizados em Sistemas Multiagentes, Vida Artificial e Algoritmos Evolutivos, foi possível atingir o objetivo deste trabalho, que era propor um ambiente para o desenvolvimento de um sistema de evolução dos comportamentos individuais de agentes reativos. Os objetivos para a construção deste ambiente, foram:

- análise do comportamento de sistemas multiagentes reativos que apresentem características de adaptabilidade e evolução;
- facilitar a descoberta de novas regras para sistemas já existentes, através dos preceitos da seleção natural.

Para a implementação do ambiente, foi estudado o ambiente Simula [FRO 97], que foi desenvolvido objetivando o desenvolvimento de aplicações em sistemas multiagentes reativos. Cada grupo de agentes, no ambiente Simula, tinha suas regras de comportamento definidas através de um conjunto de regras pré-definidas, que poderiam ser agrupadas através de conetivos lógicos.

A partir do estudo realizado sobre o ambiente Simula, foi desenvolvido o Simula++. Apesar deste novo ambiente conter as mesmas características gerais para a definição das aplicações, a forma como as simulações desenvolvem-se foi mudada por completo. O Simula++ utiliza um algoritmo evolutivo, que pode ser configurado pelo usuário, para modificar, através dos preceitos da seleção natural, o conjunto de regras que definem o comportamento individual de cada agente.

Como principais vantagens do ambiente Simula++, podemos citar:

- facilidade no uso: devido a interface gráfica, desenvolvida já a partir do ambiente Simula, o usuário consegue modelar sua aplicação facilmente, não sendo necessário nenhum tipo de programação baseada em uma linguagem específica, pois o conjunto de regras pré-definido foi construído de forma a facilitar a compreensão do usuário final, utilizando-se verbos e substantivos da língua portuguesa. Além disso, a definição de como o algoritmo evolutivo funcionará é realizada através de poucas configurações, que podem ser facilmente alteradas pelo usuário através de caixas de diálogo específicas;
- saída de dados: além da saída gráfica, onde o usuário pode visualizar o resultado de sua simulação em tempo real, o ambiente Simula++ fornece uma série de dados, que são gravados em arquivos temporários, para posterior análise pelo usuário. Dentre estes arquivos, os dois tipos mais importantes são os que armazenam a definição das regras de cada agente gerado no ambiente e o número de agentes de cada tipo a cada dez passos de simulação;
- mudança de paradigma: a qualquer momento, o usuário pode definir que a sua aplicação não mais necessita a utilização do algoritmo evolutivo, retornando à simulação com regras comportamentais estáticas. Estas regras podem ser as mesmas que foram descobertas pelo algoritmo evolutivo.

O estudo de caso realizado no ambiente Simula++ foi utilizado para avaliar a performance deste em relação a um problema já clássico nos SMAR, além de prover um teste adequado para a capacidade do simulador em encontrar novas regras para problemas já conhecidos.

A descoberta de novas regras para um problema pode auxiliar ao desenvolvedor em repensar seu sistema através de caminhos pouco usuais. O principal benefício desta metodologia provém do fato de que a análise e utilização de regras pouco ortodoxas pode levar a um estado de melhor entendimento do problema em questão, o que certamente contribuirá para o aumento da performance deste.

Uma característica importante foi percebida durante as simulações realizadas. Usualmente, um SMA possui uma série de agentes com seus próprios objetivos e regras. De acordo com o comportamento individual de cada agente, espera-se obter um

comportamento global emergente inteligente. Ao utilizarmos um algoritmo evolutivo para a modificação dos comportamentos individuais de cada agente, a definição do cálculo do seu grau de adaptação é de fundamental importância para o bom desempenho do mesmo.

No entanto, existe uma distinção clara entre o grau de adaptação do agente individual e do sistema como um todo. Um conjunto de regras de comportamento que sejam consideradas boas para um agente pode levar a um péssimo comportamento global emergente e um bom comportamento emergente pode ser formado pela interação entre agentes cujos comportamentos individuais podem ser considerados ruins.

Este problema poderia ser resolvido com a utilização de dois graus de *fitness*: (1) o primeiro, global, que mediria a performance de todo o sistema; (2) e o segundo, local, que mediria a performance de cada agente individualmente, de acordo com seus próprios critérios. Um sistema que utilizasse duas funções de avaliação necessitaria, provavelmente, de uma forma de medir a influência de cada agente na definição do *fitness* global e um mecanismo que aumentasse ou diminuísse a probabilidade deste agente ser escolhido para a reprodução, de acordo com o seu *fitness* local e sua influência no *fitness* global.

Após a finalização do protótipo do sistema Simula++ e posterior coleta de resultados, pode-se perceber alguns aspectos que podem ser citados como possíveis melhoramentos e trabalhos futuros, identificados à seguir:

- possibilidade do usuário trocar a função de *fitness* de cada agente, que hoje está baseada unicamente na sua sobrevivência. Uma função de avaliação mais complexa aumentaria o campo de aplicações possíveis de serem implementadas no ambiente Simula++;
- a definição e implementação dos conceitos de *fitness* local e global, que pudessem ser construídos pelo usuário. Estas duas funções de avaliação, discutidas anteriormente, poderiam aumentar a performance do sistema, além de prover um valioso método para a avaliação de conjuntos de regras específicas;
- geração de gráficos e estatísticas através dos dados já fornecidos pelo Simula++. Atualmente, estes gráficos são gerados por programas externos, através dos dados coletados nos arquivos de simulação;
- implementação de outras variantes dos algoritmos evolutivos, para uma comparação entre os diversos métodos existentes na literatura.

Anexo 1 Lista das Regras Pré-Definidas no Ambiente Simula++

- movimento_direcionado(direção): efetua um deslocamento de uma posição em uma determinada direção, como visto na fig. A.1.

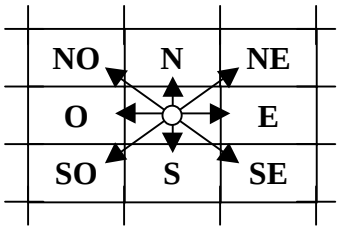


FIGURA A.1 – Movimento Direcionado

Parâmetro: direção para à qual o agente deve movimentar-se (N, NE, E, SE, S, SO, O ou NO).

- movimento_randômico(): realiza um movimento aleatório, para uma das oito direções possíveis, como visto na regra anterior. Esta regra não possui parâmetros.

- deixa_pista(): esta regra faz com que o agente, ao deslocar-se, deixe uma marca (pista), que pode ser seguida pelos demais agentes. A pista é representada por uma pequena bandeira desenhada no ambiente de simulação, como visto na fig. A.2. Esta regra não possui parâmetros.

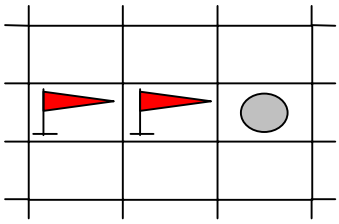


FIGURA A.2 – Deixa Pista

- remove_pista(): esta regra faz com que o agente, estando na mesma posição da bandeira (pista) deixada por outro agente, ou por ele mesmo, a remova do ambiente. Esta regra não possui parâmetros.

- segue_pista(): o agente, ao executar esta regra, seguirá a próxima bandeira (pista) que este perceber, como visto na fig. A.3. O agente segue a pista na direção oposta ao deslocamento do agente que a deixou. Esta regra não possui parâmetros.

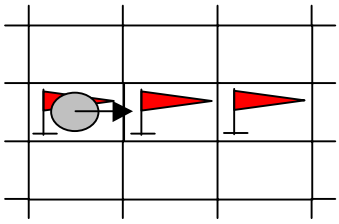


FIGURA A.3 – Segue Pista

- segue_maior_gradiente(agente): para o Simula++, gradiente é a medida da força de um sinal emitido por um determinado tipo de agente. Este sinal pode ser percebido por todos os agentes. Esta regra faz com que o agente calcule qual o sinal mais forte que ele pode perceber e desloque-se para aquela direção. Este sinal é percebido por todos os agentes, independentemente do seu valor de Área de Percepção.

  Parâmetro: tipo de agente que emite o sinal.

- segue_menor_gradiente(agente): esta regra é complementar à anterior. Ela faz com que o agente calcule qual o sinal mais forte que ele pode perceber e desloque-se para a direção oposta ao do sinal. Da mesma forma que a regra anterior, este sinal é percebido por todos os agentes, independentemente do seu valor de Área de Percepção.

  Parâmetro: tipo de agente que emite o sinal.

- segue_agente(agente): esta regra faz com que um agente A possa perseguir um determinado tipo de agente (B), caso o perceba em suas imediações. Isto é realizado através do deslocamento do agente A em direção ao agente B. O agente A só pode perceber um agente B se este estiver dentro da Área de Percepção definida para o agente A.

  Parâmetro: tipo de agente que deve ser perseguido.

- foge_de_agente(agente): esta regra faz com que um agente A possa fugir de um determinado tipo de agente (B), caso o perceba em suas imediações. Isto é realizado através do deslocamento do agente A na direção oposta ao agente B. Da mesma forma que a regra anterior, o agente A só pode perceber um agente B se este estiver dentro da Área de Percepção definida para o agente A.

  Parâmetro: tipo de agente que deve ser evitado.

- mata_agente(agente): esta regra permite que um determinado agente A possa matar (eliminar da simulação) um determinado tipo de agente (B), se estes estiverem em posições contíguas.

  Parâmetro: tipo de agente que deve ser eliminado.

- morte(): esta regra elimina o próprio agente da simulação

- reproduz(int n): esta regra faz com que sejam criados *n* novos agentes do mesmo tipo do agente que invocou a regra. A posição inicial destes novos agentes é escolhida aleatoriamente.

  Parâmetro: número inteiro de agentes que devem ser criados.

- transforma(agente): esta regra transforma uma determinado agente do tipo A num agente do tipo B.

  Parâmetro: tipo de agente(B) no qual o agente A(que invocou a regra) deve ser transformado.

- percebe_agente(agente): esta regra faz com que um determinado agente A visualize a sua cercania, buscando encontrar um determinado tipo de agente B. A visualização leva em conta a área de percepção definida para o tipo de agente A. A fig. A.4 mostra um exemplo de visualização para um agente com área de percepção igual a um.

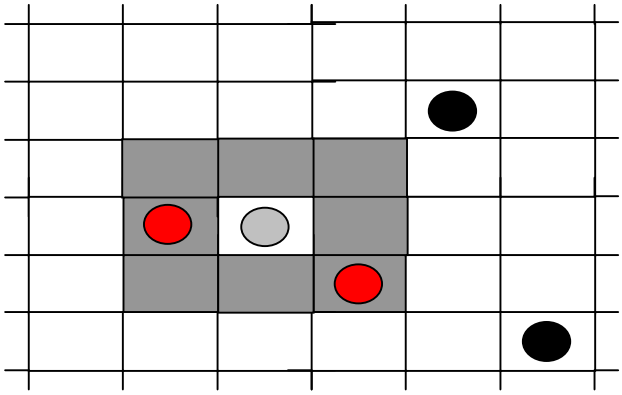


FIGURA A.4 – Percebe Agente

Neste caso, o agente central perceberia apenas os agentes representados pelas elipses em cinza escuro, e não perceberia os agentes pintados de preto.

  Parâmetro: tipo de agente que deve ser percebido.

- atinge_agente(agente): esta regra indica se um determinado agente A está ao lado de um determinado tipo de agente B.

  Parâmetro: tipo de agente que deve ser percebido.

- percebe_pista(): esta regra faz com que um determinado agente A visualize a sua cercania, buscando encontrar bandeiras (pistas), deixadas por outros agentes. A visualização leva em conta a área de percepção definida para o tipo de agente A.

- atinge_pista(agente): esta regra indica se o agente A está ao lado de uma pista, que foi deixada por um determinado tipo de agente B.

  Parâmetro: tipo de agente que deixou a marca.

- atinge_tempo_de_vida(): esta regra retorna verdadeiro se o agente A já atingiu seu tempo máximo de vida. Esta regra não possui parâmetros.

- tempo_de_fertilidade(): esta regra retorna verdadeiro se o agente A já atingiu seu tempo de fertilidade, ou seja, se ele já está apto a gerar novos descendentes. Esta regra não possui parâmetros.

- tipo_sexo(sexo, taxa de nascimento): esta regra indica o sexo do agente (macho ou fêmea) e a taxa de nascimento deste tipo de agente.

  Parâmetros: sexo e a percentagem da taxa de nascimento.

- taxa_sucesso(taxa): retorna um valor lógico (Verdadeiro ou Falso), na proporção da taxa de sucesso passada como parâmetro.

  Parâmetro: percentual que será comparado com a variável de *sucesso* interna do agente.

Bibliografia

[ALV 97] ALVARES, L.O.; SICHMAN, J. Introdução aos Sistemas Multiagentes. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 16.; CONGRESSO DA SBC, 17., 1997, Brasília. **Anais...** Brasília: SBC, 1997. p.1-38.

[BRO 86] BROOKS, R. A. A Robust Layered Control for a Mobile Robot. **IEEE Journal of Robotics and Automation**, New York, v. 2, n. 1, Mar. 1986.

[BRO 91] BROOKS, R. A. Intelligence Without Reason. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE (IJICAI), 12., 1991, Sydney. **Proceedings...** Austrália: [s.n.], 1991.

[DAR 87] DARWIN, C. **A origem das Espécies.** Rio de Janeiro: Tecnoprint, 1987.

[DAV 91] DAVIS, L. **Handbook of Genetic Algorithms.** New York: Van Nostrand Reinhold, 1991.

[DEM 95] DEMAZEAU, Y. From interactions to collective behaviour agent-based system. In: EUROPEAN CONFERENCE ON COGNITIVE SCIENCE, 1., 1995, St. Malo. **Proceedings...** France: [s.n], 1995.

[DRO 93] DROGOUL, A. **De la Simulation Multi-Agents à la Résolution Collective de Problèmes.** Paris, France: Université Paris VI, 1993. Thèse de Doctorat.

[EMM 94] EMMECHE, C. **The Garden in the Machine – The Emerging Science of Artificial Life.** New Jersey : Princeton University Press, 1994.

[FER 91] FERBER, J.; GASSER, L. Intelligence artificielle distribuée. In: INTERNATIONAL WORKSHOP ON EXPERT SYSTEMS & THEIR APPLICATIONS, 10., 1991, Avignon. **Cours n. 9.** France: [s.n], 1991.

[FRO 97] FROZZA, R. **SIMULA – Ambiente para Desenvolvimento de Sistemas Multiagentes Reativos.** Porto Alegre: CPGCC da UFRGS, 1997. Dissertação de mestrado.

[GAS 92] GASSER, L. Boundaries, identity and aggregation: plurality issues in multiagent systems. In: WERNER, E.; DEMAZEAU, Y. (Eds.). **Decentralized AI 3.** Amsterdam: North-Holland, 1992.

[HOL 75] HOLLAND, J.H. **Adaptation in Natural and Artificial Systems.** Massachusetts : MIT Press, 1992.

[KOZ 98] KOZA, J. **Genetic Programmin:** on the programming of computers by means of natural selection. Massachusetts: MIT Press, 1998.

[LAN 84] LANGTON, C.G. Self-reproduction in cellular automata. **Physica D,** [S.l], v. 10, p.135-144, 1984.

[LAN 89] LANGTON, C.G. Artificial Life. In: LANGTON, C.G.(Ed.), **Artificial Life.** [S.l]: Addison-Wesley, p.1-47, 1989.

[LAT 98] LATTAUD, C.; CUENCA, C. In: HEUDIN, J.C.(Ed.)**Virtual World.** Berlin: Springler-Verlag, 1998, p.218-228.

[LEV 92] LEVY, S. **Artificial Life:** The Quest for a New Creation.[S.l]: Random House, 1992.

[MIN 88] MINSKY, Marvin. **The Society of Mind.** New York: A Touchstone book, 1988.

[NEU 66] NEUMANN, J. von. **Theory of Self-Reproducing Automata**. Illinois : University of Illinois Press, 1966.

[PFE 96] PFEIFER, R. Building "Fungus Eaters": Design Principles of Autonomous Agentes, In: INTERNATIONAL CONFERENCE ON SIMULATION OF ADAPTIVE BEHAVIOR, 4., 1986. Proceedings... Cambridge: MIT, 1996. p. 3-12.

[RAY 94] RAY, TS. An evolutionary approach to synthetic biology, zen and the art of creating life. In: LANGTON, C.G (Ed.) **Artificial Life 1**.[S.l]: Addison-Wesley ,1994.

[REY 87] REYNOLDS, C. W. Flocks, Herds and Schools: A Distributed Behaviorial Model. In: SIGGRAPH, 1987. **Proceedings...** [S.l.:s.n], 1987. p. 25-34.

[RIC 93] RICH, E.; KNIGHT, K. **Inteligência Artificial**. São Paulo: Makron Books, 1993.

[ROC 97] ROCHA, L. **Evolutionary Systems and Artificial Life.** [S.l.]: Departament of Systems Science, State University of New York at Binghamton, 1997.

[SIC 95] SICHMAN, J. **Du raisonnement social chez les agents:** une approche fondée sur la théorie de la dépendance. Grenoble, France: INPG, 1995.Thèse de Doctorat.

[SPA 91] SPAFFORD, E. H. Computer Viruses – A form of Artificial Life ? In: LANGTON, C.G(Ed.) **Artificial Life II – Sfi Studeis in the Sciences of Complexity**. [S.l]: Addison-Wesley, 1991. V. 12.

[STE 94] STEELS, L. The Artificial life roots of artificial intelligence. **Artificial Life Journal**, Cambridge, v.1, n.1, p. 75-110, 1994.

[WRI 32] WRIGHT, S. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In: INTERNATIONAL CONGRESS OF GENETICS, 6. 1932. **Proceedings...** [S.l.:s.n.],1932. v.1, p.356-366.