REDES EPISTÊMICAS AUTO-ORGANIZÁVEIS: UM MODELO CONEXIONISTA PARA A APRENDIZAGEM EM REDES SOCIAIS

Celso Niskier

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Luis Alfredo Vidal de Carvalho

Rio de Janeiro

Setembro de 2010

REDES EPISTÊMICAS AUTO-ORGANIZÁVEIS: UM MODELO CONEXIONISTA PARA A APRENDIZAGEM EM REDES SOCIAIS

Celso Niskier

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____
Prof. Luís Alfredo Vidal de Carvalho, D.Sc.

_____
Prof. Antonio Alberto Fernandes de Oliveira, D.Sc.

_____
Prof. Alexandre de Assis Bento Lima, D.Sc.

_____
Profª. Célia Martins Cortez, D.Sc.

_____
Prof. Luerbio Farias, D.Sc.

_____
Profª. Inês de Castro Dutra, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2010

Celso, Niskier

Redes epistêmicas auto-organizáveis: um modelo conexionista para a aprendizagem em redes sociais. / Celso Niskier. – Rio de Janeiro, 2010.

VIII, 244 f.; il. ; 29,7 cm

Orientador: Luis Alfredo Vidal de Carvalho

Tese (doutorado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2010.

Referências Bibliográficas: p. 113–120.

1. Auto-organização. 2. Inteligência Artificial Distribuída. 3. Modelagem Computacional. 4. Redes Sociais. 5. Aprendizagem Social. 6. Lei de Potência. 7. Multi-agentes. I. Carvalho, Luís Alfredo Vidal de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Aos meus pais, Ruth e Arnaldo, pelo estimulo permanente à minha curiosidade.

À minha esposa, Andréa, pelo amor que nos une há mais de 30 anos.

Às minhas filhas, Giovanna e Gabriela, pelo sentido que dão à minha vida.

Agradecimentos

“Nenhum homem é uma ilha”.
John Donne

Agradeço aos mestres Carlos José Pereira de Lucena e Tom Maibaum, que me guiaram no início da minha caminhada como pesquisador de Ciência da Computação, e aos quais dedico sincera gratidão.

Agradeço ao meu orientador, Luis Alfredo Vidal de Carvalho, pelo fundamental estímulo para que eu retornasse às atividades de pesquisa, e pelo fértil ambiente de floração de ideias que criou em torno de si.

Agradeço à Helio Bezerra, pelas inúmeras conversas sobre alguns dos temas do presente trabalho, e por me escutar na minha incessante busca por autoconhecimento.

Agradeço aos amigos, mestres e colaboradores do Centro Universitário Carioca (UniCarioca), uma semente que germinou há 20 anos e que hoje produz maravilhosos frutos, pela compreensão com as minhas eventuais ausências.

Agradeço aos meus brilhantes alunos de iniciação científica, na UniCarioca, Fábio Oshiro, Leandro Freire e Heitor Peralles, pelas importantes e significativas contribuições que deram ao presente trabalho.

Agradeço à Secretária Geral da UniCarioca, Gisele Amaral, companheira de trabalho há mais de 15 anos, pela inestimável ajuda na editoração do presente texto.

Agradeço ao meu pai, Arnaldo Niskier, pela ajuda na revisão do texto. É um luxo ter um imortal da Academia Brasileira de Letras revendo o texto, e é um luxo ainda maior tê-lo como pai e, principalmente, como amigo (e torcedor).

Agradeço, por fim, à minha querida avó Paulina Dain Buchmann (*in memoriam*), que repetia sempre: “Celsinho, você é um número!”. Se até hoje não descobri que número era esse, não deixei, no entanto, de procurar desde então uma verdade científica tão bela quanto o amor incondicional que ela me dedicou.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

REDES EPISTÊMICAS AUTO-ORGANIZÁVEIS: UM MODELO CONEXIONISTA PARA A APRENDIZAGEM EM REDES SOCIAIS

Celso Niskier

Setembro/2010

Orientador: Luís Alfredo Vidal de Carvalho

Programa: Engenharia de Sistemas e Computação

O rápido crescimento das redes sociais baseadas na Web tem redefinido o espaço de interação social e motivado o surgimento de interessantes pesquisas, usufruindo da riqueza de dados que podem ser registrados a partir das múltiplas interações entre seus usuários. Apesar do recente crescimento das pesquisas nessa área, os mecanismos fundamentais que governam a dinâmica e evolução das redes sociais, ainda não são totalmente conhecidos.

O objetivo principal do presente trabalho é a proposta de um modelo computacional e a construção de um ambiente de simulação para descrever e observar como múltiplos agentes interagem em uma rede social, gerando, comunicando e revisando suas crenças e comportamentos, a partir dos seus diferentes “pontos de vista”. Para ilustrar a aplicabilidade do modelo, são realizadas diversas simulações computacionais. Em particular, é modelado e simulado um subconjunto da rede social Orkut, apresentando-se os resultados obtidos.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

SELF-ORGANIZING EPISTEMIC NETWORKS: A CONEXIONIST MODEL OF LEARNING IN SOCIAL NETS

Celso Niskier

September / 2010

Advisor: Luís Alfredo Vidal de Carvalho

Department: Computing and Systems Engineering

The rapid development of web-based social networks has redefined the space of social interactions and inspired the recent surge of new research in the area, making use of the ready available data provided by the internet. Although promising, the research field still lacks well defined mechanisms for the understanding of the dynamics and evolution of real-world social networks.

The main goal of this work is to propose a new computational model and to construct a simulation environment to describe and observe how different agents interact, generating, communicating and revising their beliefs and behaviors, from their different points of view. To illustrate its applicability a number of simulations is provided. In particular, it is modeled and simulated a subset of the web-based social network Orkut, presenting the results thus obtained.

Índice

1. Introdução 1

1.1. Motivação 1

1.2. Objetivos e Contribuições 3

1.3. Estrutura da Tese 6

2. Trabalhos Relacionados 8

2.1. Complexidade e Auto-organização 8

2.2. Modelagem Baseada em Agentes Inteligentes 11

2.3. Representação Distribuída do Conhecimento 14

2.4. Aprendizagem Social 16

2.5. Sistemas Imunológicos 22

2.6. Construção Coletiva do Conhecimento Científico 24

3. Redes Sociais 27

3.1. A Tradição Analítica 27

3.2. Grafos Randômicos 29

3.3. O Mundo é Pequeno? 32

3.4. A Formação de *Clusters* 34

3.5. Propagação de Doenças e Contágio Social 39

4. Modelo 43

4.1. Definições Básicas 43

4.2. Premissas do Modelo 50

4.3. Elementos Principais do Modelo 54

4.4. Dinâmica do Modelo 56

4.4.1. Dinâmica individual de um agente epistêmico 56

4.4.2. Dinâmica coletiva da rede epistêmica 58

5. Simulação 65

6. Estudos de Caso 72

6.1. Auto-organização de uma rede epistêmica 72

6.2. Propagação de Crenças 80

6.3. Formação de *Clusters* Concorrentes 85

6.4. Formação de Consenso 92

6.5. A Rede Social Orkut 97

7. Conclusões 109

Referências Bibliográficas 113

Anexo A – Documentação Técnica do Simulador IndraNet 1.2 121

Anexo A.1 – Diagramas de sequência e de classes/objetos 122

Anexo A.2 – Programas 125

Figuras

Figura 3.1 - Pontes de Konigsberg (adaptada de [10]) 29
Figura 3.2 – Exemplo de Emergência de uma Rede em uma Festa Social (adaptada de [10]) 30
Figura 3.3 - Modelo de Laços Fracos (*Weak Ties*), inspirada por Granovetter [34] 33
Figura 3.4 - Modelo Watts-Strogatz [76] 34
Figura 3.5 – (a) Rede com Lei de Potência (*scale-free*)
(b) Rede com distribuição normal (adaptada de [74]) 36
Figura 3.6 – Redes randômicas X redes *scale-free* (adaptada de [10]) 37
Figura 3.7 – Modelos de propagação de doenças X modelos de propagação de crenças [73] 41
Figura 4.1 – Funcionalidades de um Agente Epistêmico 45
Figura 4.2 – Funcionalidades de uma Rede Epistêmica com Três Agentes Epistêmicos 46
Figura 4.3 – Agentes e suas crenças 47
Figura 4.4 – Agentes e seus conhecimentos individuais 48
Figura 4.5 – Agentes e o conhecimento coletivo da rede 48
Figura 4.6 – Agentes e o conhecimento comum da rede 49
Figura 4.7 – Rede Neuronal de Aprendizado por retro-propagação que representa um agente epistêmico 54
Figura 4.8 – Rede Neuronal Genérica de Aprendizado Competitivo que representa a rede epistêmica com três agentes epistêmicos (epistrons) 55
Figura 4.9 – Foco e escopo de um Agente Epistêmico (Epistron) 56
Figura 4.10 – Diagrama esquemático de 3 agentes epistêmicos (epistrons) com diferentes focos e escopos 57
Figura 4.11 – Diagrama de Interesses de três usuários do Orkut 58
Figura 4.12a – Fase de Geração 59
Figura 4.12b – Fase de Comunicação 59
Figura 4.12c – Fase de Comparação 59
Figura 4.12d – Fase de Revisão (Aprendizado Individual) 60
Figura 4.12e – Fase de Atualização dos Ws (Aprendizado Coletivo) 60

Figura 5.1 – Tela de configuração dos parâmetros dos agentes epistêmicos no simulador IndraNet 66
Figura 5.2 – Tela de configuração dos parâmetros da rede epistêmica no simulador IndraNet 68
Figura 5.3 – Tela de configuração dos parâmetros de visualização da rede epistêmica no simulador IndraNet 69
Figura 6.1 – Situação inicial do Exemplo 1.(a) 73
Figura 6.2 – Exemplo 1.(a) após 13 etapas 73
Figura 6.3 – Exemplo 1.(a) após 101 etapas 74
Figura 6.4 – Exemplo 1.(a) após 500 etapas 75
Figura 6.5 – Exemplo 1.(a) após 1.000 etapas 75
Figura 6.6 – Tela inicial do Exemplo 1.(b) 77
Figura 6.7 – Exemplo 1.(b) após 751 etapas 77
Figura 6.8 – Exemplo 1.(b) após 1501 etapas 78
Figura 6.9 – Exemplo 1.(b) após 3.063 etapas (Lei de Potência) 78
Figura 6.10 – Gráfico de Distribuição Média dos Somatórios de Pesos dos Agentes 79
Figura 6.11 – Tela inicial do Exemplo 2 81
Figura 6.12 – Exemplo 2 após 10 etapas 81
Figura 6.13 – Exemplo 2 após 16 etapas 82
Figura 6.14 – Exemplo 2 após 22 etapas 82
Figura 6.15 – Exemplo 2 após 26 etapas 83
Figura 6.16 – Exemplo 2 após 32 etapas 83
Figura 6.17 – Exemplo 2 após 100 etapas 84
Figura 6.18 – Crenças inicias dos dois agentes no Exemplo 3 86
Figura 6.19 – Tela de início do Exemplo 3 86
Figura 6.20 – Exemplo 3 após 10 etapas 87
Figura 6.21 – Exemplo 3 após 19 etapas 87
Figura 6.22 – Exemplo 3 após 36 etapas 88
Figura 6.23 – Exemplo 2 após 57 etapas 88
Figura 6.24 – Exemplo 3 após 91 etapas 89
Figura 6.25 – Exemplo 3 após 112 etapas 89
Figura 6.26 – Exemplo 3 após 176 etapas 90

Figura 6.27 – Exemplo 3 após 231 etapas 90
Figura 6.28 – Exemplo 3 após 301 etapas (formação de *clusters* concorrentes) 91
Figura 6.29 – Configuração inicial do Exemplo 4 93
Figura 6.30 – Tela inicial do Exemplo 4 93
Figura 6.31 – Exemplo 4 após 66 etapas 94
Figura 6.32 – Exemplo 4 após 90 etapas 94
Figura 6.33 – Exemplo 4 após 133 etapas 95
Figura 6.34 – Exemplo 4 após 273 etapas 95
Figura 6.35 – Exemplo 4 após 329 etapas 96
Figura 6.36 – Exemplo 4 após 495 etapas (formação de consenso) 96
Figura 6.37 – Perfil de um usuário do Orkut (com permissão) 98
Figura 6.38 – Grafo do Exemplo do Orkut 101
Figura 6.39 – Subconjunto de Usuários e seus interesses no Orkut 102
Figura 6.40 – Crenças Modeladas do Exemplo 5 103
Figura 6.41 – Tela inicial do Exemplo 5 103
Figura 6.42 – Exemplo 5 após 6 etapas 104
Figura 6.43 – Exemplo 5 após 52 etapas 104
Figura 6.44 – Exemplo 5 após 321 etapas 105
Figura 6.45 – Exemplo 5 após 338 etapas 105
Figura 6.46 – Exemplo 5 após 353 etapas 106
Figura 6.47 – Exemplo 5 após 379 etapas 106
Figura 6.48 – Exemplo 5 após 490 etapas 107
Figura 6.49 – Exemplo 5 após 785 etapas 107
Figura 6.50 – Exemplo 5 após 1001 etapas 108
Figura 6.51 – Situação final do Exemplo 5 com gráfico de distribuição (Lei de Potência) 108
Figura 7.1 – Tabela de Aplicações do Modelo Proposto 112

1. Introdução

1.1. Motivação

O rápido crescimento das redes sociais baseadas na Web tem redefinido o espaço de interação social e motivado o surgimento de interessantes pesquisas, usufruindo da riqueza de dados que podem ser registrados a partir das múltiplas interações dos seus usuários. A Web permite que as pessoas participem e interajam entre si através das comunidades virtuais, incluindo fóruns, grupos de discussão e redes sociais, tais como o Orkut[1], o Facebook[2] e o Twitter[3]. Todo esse conteúdo gerado pelos usuários oferece aos cientistas da computação uma oportunidade única para estudar a chamada "inteligência coletiva" nas comunidades on-line. Apesar do recente crescimento das pesquisas nessa área, os mecanismos fundamentais que governam a dinâmica e a evolução das redes sociais ainda não são completamente conhecidos.

Demonstrando o interesse crescente no estudo da inteligência coletiva na *Web*, há poucos meses, dirigido por pesquisadores do *Rensselaer Polytechnic Institute* de Nova Iorque, foi fundado o *Social and Cognitive Networks Academic Research Center* (SCNARC), com financiamento do Departamento de Defesa norte-americano, dedicado a pesquisar a dinâmica e a evolução dos comportamentos nas redes sociais virtuais [61]. O Centro de Pesquisas organiza-se em cinco projetos, focando a ciência das redes sociais virtuais e suas potenciais aplicações nas áreas industriais e militares. Uma das potenciais aplicações militares é na área de Inteligência, por meio da criação de mecanismos de identificação e prevenção de ameaças terroristas, a partir da análise das trocas de mensagens nas redes sociais da *Web*.

1. www.orkut.com
2. www.facebook.com.br
3. www.twitter.com

Outro fator que comprova o crescimento recente da área é o surgimento de edições especiais de tradicionais periódicos científicos, tais como a revista *Science* (edição especial sobre *Complex Systems and Networks* [65]), o *ACM Transactions on Intelligent Systems and Technology* (edição especial em "Computational Models of Collective Intelligence in the Social Web", prevista para abril/maio de 2011 [60]), o *IEEE Intelligent Systems* (edição especial em "Social Learning", publicada em julho/agosto de 2010 [64]) e o *Journal of Emerging Technologies in Web Intelligence* (edição especial em *Learning and the Social Web* [63]), dedicados às questões relacionadas à descoberta, análise e modelagem do comportamento social humano, em especial a emergência da chamada "inteligência coletiva".

Por todos esses motivos, acredita-se que o estudo da dinâmica das redes sociais seja relevante para o avanço do conhecimento científico e para o futuro desenho de novos métodos, técnicas e ferramentas que possam fazer uso do enorme potencial da inteligência coletiva na *Web* social (a chamada *Web* 2.0).

1.2. Objetivos e Contribuições

O objetivo principal do presente trabalho é a proposta de um modelo computacional de abordagem conexionista e a construção de um ambiente de simulação para descrever e observar como múltiplos agentes interagem em uma rede social, gerando, comunicando e revisando suas crenças e comportamentos, a partir dos seus diferentes "pontos de vista" sobre um mesmo problema, com aplicação ao fenômeno da aprendizagem social e da emergência da inteligência coletiva na *Web* social.

Entre os objetivos específicos, pode-se destacar:

(a) A revisão dos trabalhos relacionados ao tema em questão, apontando suas contribuições e limitações, e as questões em aberto hoje existentes na área escolhida;

(b) A proposta de um modelo conexionista, baseado em redes neuronais, em dois níveis (nível do agente individual e nível do comportamento da rede), com características simples, porém capaz de expressar uma ampla gama de fenômenos emergentes da área de redes sociais;

(c) A construção de um ambiente de simulação computacional através do qual se pode observar a dinâmica do modelo, a partir de diversos parâmetros e condições iniciais;

(d) A exemplificação do modelo através de estudos de caso selecionados a partir da literatura corrente;

(e) A demonstração da aplicabilidade do modelo para a representação da dinâmica de uma rede social virtual, como o Orkut.

Entre as contribuições esperadas do presente trabalho, consideram-se as seguintes:

(a) A formalização de conceitos das teorias sociológicas de Gabriel Tarde e da Teoria Social da Aprendizagem de Albert Bandura em um modelo computacional, que pode ser simulado em computadores;

(b) A proposta de um modelo simples, com poucos parâmetros, porém capaz de modelar uma ampla gama de fenômenos emergentes em redes sociais, possivelmente generalizando outros modelos de aprendizagem social propostos na literatura;

(c) A criação de um ambiente de simulação computacional desenvolvido na linguagem Java, que possibilita a experimentação ***in silico*** de uma gama variada de situações possíveis de aprendizagem em redes sociais, e a verificação do surgimento de propriedades emergentes;

(d) A modelagem de um subconjunto da rede social virtual Orkut, propiciando a observação da sua dinâmica sob o aspecto da aprendizagem social, com possíveis aplicações futuras na construção de modelos preditivos para o comportamento de usuários e a evolução de suas crenças na *Web* social.

Este trabalho amplia e aprofunda pesquisas anteriormente realizadas pelo autor, que focavam o uso de múltiplas visões na especificação de sistemas de software [49, 50, 51], na medida em que parte do mesmo pressuposto de que diferentes agentes constroem seus conhecimentos de forma paralela e distribuída, a partir de diferentes "pontos de vista" sobre o mesmo problema. Tal abordagem evoluiu desde então, constituindo-se hoje em uma sub-área de pesquisa dentro do campo maior da Engenharia de Software [24].

O modelo proposto possui interessantes aplicações e pode ser instanciado em várias áreas onde sistemas distribuídos inteligentes se façam presentes, como, por exemplo, nos sistemas imunológicos e na construção coletiva do conhecimento científico. Nestes, por se tratarem de exemplos de sistemas complexos (distribuídos e não-lineares), importantes propriedades emergentes

como a auto-organização levam à formação de agrupamentos (*clusters*) de agentes com crenças comuns, além da possibilidade de se observar fenômenos como a bifurcação, com a consequente emergência de novos padrões estruturais e funcionais nos sistemas analisados.

1.3. Estrutura da Tese

A presente tese estrutura-se em mais 6 capítulos, além desta Introdução.

No capítulo 2, procurou-se rever trabalhos existentes nas áreas relacionadas aos temas da presente tese. Em especial, foi feita uma revisão da literatura e dos principais conceitos nas áreas de Complexidade, Auto-organização, Inteligência Artificial Distribuída e Modelos Conexionistas. Relacionam-se também os principais trabalhos e modelos na área de Aprendizagem Social, aprofundando-se na origem do conceito de aprendizagem por imitação, desde o trabalho pioneiro de Gabriel Tarde e as indagações de Michael Polanyi, passando pelas ideias de Henri Atlan de auto-organização dos sistemas vivos até a criação mais recente da Teoria da Aprendizagem social do psicólogo canadense Albert Bandura. Finaliza-se apresentando-se, de forma breve, duas áreas em que ocorre o fenômeno da aprendizagem social, e que contribuíram como inspiração para o modelo proposto: Sistemas Imunológicos e Sociologia da Ciência.

No capitulo 3 é feita uma apresentação dos principais trabalhos na área de Redes Sociais, que servirão como referência para a discussão de alguns dos resultados obtidos com a simulação do modelo de redes epistêmicas.

No capítulo 4 é apresentado o modelo proposto de redes epistêmicas, considerando-se as premissas adotadas na sua definição, e as decisões sobre a melhor forma de representação de agentes e redes epistêmicas. Encerra-se o capítulo 4 com a apresentação do algoritmo que formaliza a dinâmica do modelo conexionista proposto.

No capítulo 5 é descrita a simulação computacional do modelo proposto, justificando-se as decisões de projeto adotadas e ilustrando as funcionalidades da implementação realizada.

No capítulo 6 apresentam-se os exemplos que ilustram a aplicabilidade do modelo proposto. Em especial, descreve-se como as redes epistêmicas auto-organizam-se a partir de configurações aleatórias, como as crenças são propagadas em uma rede epistêmica, como são formados *clusters* de agentes epistêmicos a partir de diferentes parâmetros e condições iniciais e como se chega a um consenso em uma rede epistêmica. Ao final, é apresentada uma instanciação do modelo para representação de uma rede social real, no caso a rede social Orkut. São discutidos os critérios usados na definição dos parâmetros, e apresentados os resultados encontrados. É também demonstrado, por simulação, um resultado importante: as redes epistêmicas se auto-organizam segundo uma Lei de Potência, constituindo-se em um caso especial de rede *scale-free*.

No capítulo 7, por fim, são apresentadas as conclusões e discutidas as possibilidades de futuros trabalhos na área. Em particular, apresenta-se um quadro de aplicações possíveis das redes epistêmicas em várias áreas de estudo. Encerra-se o texto com a necessária discussão das limitações do presente trabalho, e sugestões de como podem ser superadas no futuro.

No Anexo A apresenta-se a documentação técnica do simulador IndraNet (versão 1.2), especialmente desenvolvido para realizar os experimentos do presente trabalho.

2. Trabalhos Relacionados

A presente revisão da literatura tem a intenção de assinalar os trabalhos mais relevantes para o desenvolvimento da tese, e que serviram de inspiração para a proposta do modelo conexionista para a aprendizagem em redes sociais. Partindo dessa premissa, organizou-se o capítulo 2 como uma revisão dos conceitos de complexidade e auto-organização, modelagem com base em agentes inteligentes, modelos conexionistas para representação do conhecimento e aprendizagem social, deixando-se para o próximo capítulo uma revisão mais detalhada dos principais conceitos e temas das redes sociais.

2.1. Complexidade e Auto-organização

O estudo de sistemas complexos tem sido reconhecido como um novo campo científico (Bar-Yam [13]), combinando e integrando várias disciplinas, desde a Física até a Antropologia. A motivação desse campo é a busca de propriedades universais para sistemas complexos. Um sistema complexo é um sistema constituído por um grande número de partes, interagentes e interconectadas. Para identificar as propriedades de um sistema complexo deve-se descrever não só a propriedade de suas partes individualmente, mas também a forma como as partes interagem, o que torna a tarefa bem mais difícil do que os sistemas ditos lineares. Exemplos de sistemas complexos são universidades, organizações de negócios, famílias, o corpo humano, entre outros.

Sistemas complexos podem ser descritos a partir das seguintes características [13]:

(a) Elementos (e seus números);
(b) Interações de elementos (e suas intensidades);
(c) Formação e operação (e sua dinâmica);
(d) Diversidade/variabilidade;

(e) Ambientes (e suas exigências);
(f) Atividades (e seus objetivos).

Uma importante questão dos sistemas complexos é a emergência de comportamentos coletivos a partir do comportamento individual das suas partes. A predição da emergência de comportamentos coletivos complexos a partir de comportamentos simples das partes de um sistema complexo é um campo interessante de pesquisa, com várias aplicações.

Em síntese, um sistema complexo é um sistema formado por vários componentes, cujo comportamento é emergente, isto é, o comportamento do sistema não pode ser inferido a partir do comportamento dos seus componentes. A quantidade de informação necessária para descrever o comportamento de um sistema é a medida da sua complexidade.

Para o biólogo argelino radicado na França, Henri Atlan [2], o conceito de sistema auto-organizador foi proposto como uma forma de conceber os organismos vivos sob a forma de máquinas cibernéticas, com propriedades específicas de adaptação ao ambiente. De uma forma geral, pode-se conceber a evolução de sistemas organizados, ou o fenômeno de auto-organização, como um processo de aumento de complexidade, simultaneamente estrutural e funcional, resultante de uma sucessão de “desorganizações resgatadas”, acompanhadas, em todas as ocasiões, pelo restabelecimento num nível de variedade maior e de redundância mais baixa. A organização seria, portanto, um processo ininterrupto de desorganização-reorganização, provocado por ruídos externos que atuariam como fatores de organização. É importante destacar, nesse ponto, que Atlan não considera o ruído como algo positivo ou negativo, apenas um fator que contribui para a evolução de um sistema através da sua auto-organização.

Aplicando o princípio da complexidade pelo ruído a uma teoria da aprendizagem, Atlan chega a algumas propriedades do que se pode considerar como um sistema auto-organizador de aprendizagem não-dirigida.

Diferentemente da aprendizagem dirigida, na qual um professor diz o que é preciso aprender, na aprendizagem não-dirigida um sistema é colocado em um ambiente que é novo para ele, e passa a criar os padrões que em seguida ele mesmo passará a reconhecer [2]. Atlan considera, nesses casos, que esse processo de aprendizagem pode ser compreendido como uma criação de padrões por diminuição da redundância. Assim, o que aumenta na aprendizagem é, conforme esse princípio, a diferenciação do sistema, a especificidade dos padrões apreendidos, e o que diminui é a redundância total do sistema, o caráter indiferenciado. Explicando melhor, através da aprendizagem não-dirigida, ou auto-organizada, o sistema adquire especificidade no reconhecimento de certos padrões, tornando-se mais diferenciado. Veremos, nos estudos de caso, como o modelo de redes epistêmicas auto-organizadas atende aos princípios identificados por Atlan, baseados na visão de construção de ordem a partir do caos, através do ruído.

Henri Atlan aplicou seus conceitos de auto-organização na descrição de sistemas biológicos, em particular na modelagem de sistemas imunológicos [3], que podem ser compreendidos como um exemplo interessante de aprendizagem social em nível celular. Além disso, procurou ampliar suas ideias para entender o funcionamento do psiquismo humano, tendo proposto uma conceituação da consciência e do desejo humano como processos de auto-organização [2]. Para Atlan, somos “máquinas de fazer sentido”, movidos pelo processo de auto-organização comum a todos os sistemas vivos. Pode-se dizer, portanto, que o processo de viver é um processo de contínua criação de ordem a partir do caos, através de sucessivas desorganizações e reorganizações.

2.2. Modelagem Baseada em Agentes Inteligentes

Agentes são entidades computacionais autônomas, que percebem o seu ambiente através de sensores e agem no ambiente através de efetores [77, 78]. Sendo computacionais, existem fisicamente na forma de programas, que se executam em dispositivos computacionais. Sendo autônomos, têm controle sobre o seu comportamento, sem intervenção humana ou de outros sistemas. Agentes buscam realizar tarefas ou concretizar objetivos, muitas vezes conflituosos com outros agentes.

A Inteligência Artificial Distribuída (IAD) é a disciplina que envolve o estudo, a construção e a aplicação de sistemas multi-agentes, entendidos como sistemas nos quais vários agentes inteligentes interagem, buscando atingir um conjunto de objetivos ou realizar determinadas tarefas. O objetivo de longo prazo das pesquisas em IAD é o desenvolvimento de mecanismos e métodos que permitam que agentes interajam tão bem quanto seres humanos, a partir de uma compreensão dos princípios que regem o seu funcionamento e a sua construção.

Várias aplicações industriais e comerciais de IAD encontram-se descritas na literatura [78], entre as quais podemos citar o comércio eletrônico, o monitoramento em tempo-real de redes de telecomunicações, a modelagem, simulação e otimização de sistemas de transporte de longo alcance, a gestão da informação em ambientes de rede, o controle de tráfego aéreo, a análise de processos de negócios entre empresas e o entretenimento interativo, com jogos baseados em realidade virtual, entre outros. Em particular, interessa aos pesquisadores investigar os aspectos sociais da inteligência e a simulação de fenômenos sociais complexos, tais como a evolução de papéis, normas e estruturas organizacionais, nas quais os agentes representam membros de sociedades naturais.

Os sistemas multi-agentes podem ser classificados a partir dos atributos dos agentes, das suas interações ou do ambiente em que atuam. As aplicações de

sistemas multi-agentes listadas anteriormente têm em comum algumas características, a saber:

(a) Distribuição - Os ambientes têm uma inerente distribuição dos dados e informações a serem processadas, considerando-se diferentes localizações geográficas (distribuição espacial), diferentes momentos (distribuição temporal), organização em *clusters* com linguagens distintas (distribuição semântica) e funcionalidades distintas (distribuição funcional).

(b) Complexidade - As aplicações são inerentemente complexas, no sentido de que são muito grandes para serem resolvidas por um único sistema centralizado, dadas as limitações de hardware e software que podem existir.

A solução para problemas dessa natureza tem sido a distribuição do processo entre múltiplas entidades capazes de coordenação inteligente. A IAD busca, portanto, prover tecnologias e metodologias para a solução elegante desses problemas.

Algumas questões em aberto na pesquisa em IAD, ainda segundo Weiss [77] são as seguintes:

(a) Como os agentes decompõem problemas em sub-problemas;
(b) Como os agentes se comunicam;
(c) Como os agentes raciocinam sobre planos, ações e conhecimentos de outros agentes, de forma a interagir propriamente;
(d) Como os agentes raciocinam sobre o processo de solução de problemas;
(e) Como os agentes conciliam diferentes pontos de vista e resolvem conflitos;
(f) Como projetar e desenvolver sistemas multi-agentes e suas plataformas;
(g) Como equilibrar computação local e coordenação entre agentes computacionais;

(h) Como evitar comportamentos inadequados do sistema;
(i) Como os agentes negociam entre si;
(j) Como os agentes formam equipes e se organizam durante o processo de solução;
(k) Como descrever formalmente os sistemas multi-agentes;
(l) Como implementar “processos inteligentes”, tais como resolução de problemas, planejamento, decisão e aprendizado em sistemas multiagentes.

Novas pesquisas são necessárias para desenvolver a base e as técnicas para representação de sociedades de agentes computacionais autônomos, que atuam em ambientes abertos, por períodos indefinidos. Essas pesquisas necessitarão confiar na habilidade dos agentes para adquirir e usar representações de outros agentes, através de processos de negociação, cooperação, coordenação e aprendizado multi-agentes.

2.3. Representação Distribuída do Conhecimento

Segundo Russel & Norvig [58], as primeiras discussões sobre a representação do conhecimento em Inteligência Artificial (IA) tendiam a se concentrar na "representação de problemas", e não na "representação do conhecimento". Na década de 70 e 80, a pesquisa em IA enfatizava o desenvolvimento de "sistemas especialistas", também chamados de "sistemas baseados em conhecimento", que podiam, dado o conhecimento de um domínio apropriado, equiparar ou superar o desempenho de especialistas humanos em tarefas específicas.

Com o passar do tempo, os pesquisadores de IA ficaram cada vez mais interessados em formalismos para representação do conhecimento que poderiam simplificar a criação de novos sistemas especialistas, incorporando propriedades que envolvessem raciocínio temporal, mudanças, ações e eventos. As várias lógicas não-clássicas propostas na literatura atestam a importância dessa vertente de pesquisa, entre elas as lógicas temporal, dinâmica, deôntica e epistêmica, entre outras.

Alternativamente aos proponentes dos modelos declarativos de representação do conhecimento, os defensores do chamado paradigma conexionista consideram que a informação pode ser armazenada de forma distribuída, e é processada paralelamente por um conjunto de elementos computacionais simples, denominados neurônios. Estes, distribuídos no espaço e ligados através de conexões, denominadas sinapses, trocam sinais inibitórios ou excitatórios, competindo ou cooperando entre si. No modelo conexionista, o conhecimento estaria armazenado nas conexões entre os neurônios, e o comportamento inteligente emergiria da atuação simultânea desta coletividade, sem a necessidade de elementos centralizadores. Denomina-se "neurocomputação" o estudo dos processos computacionais realizáveis por sistemas dinâmicos que obedecem ao paradigma conexionista (McClelland [44], Vidal de Carvalho [69]).

No chamado paradigma conexionista, acredita-se que a inteligência de um sistema emerge a partir da interação de um grande número de unidades simples de processamento [44]. Para os adeptos desta abordagem, a modelagem simbólica tradicional, na qual o conhecimento é representado de forma centralizada e sequencial, falha em capturar a interatividade na natureza do processamento de informações.

A abordagem PDP (*Parallel Distributed Processing*) seria, para seus defensores, um paradigma mais eficiente e natural para a representação de fenômenos que ocorram de forma descentralizada e simultânea [44].

A literatura referente às aplicações dos modelos conexionistas é ampla, e uma revisão completa foge ao escopo do presente trabalho. É importante ressaltar, no entanto, a inexistência de propostas de modelos conexionistas para os fenômenos da aprendizagem social, o que sem dúvida chama a atenção, já que nos parece uma escolha natural, considerando-se que a aprendizagem ocorre em ambientes em que diferentes agentes (unidades de processamento) atuam de forma simultânea, a partir das suas múltiplas e distribuídas representações.

Finalmente, é importante destacar que o conceito de aprendizagem competitiva [44] é um bom candidato para representar a situação em que diversos agentes “competem” pela atenção de seus pares, e foi uma das inspirações na escolha do modelo conexionista para representar a aprendizagem em redes sociais.

2.4. Aprendizagem Social

Diferentemente de Émile Durkheim, que fundou sua escola de pensamento sociológico focada na análise de grandes representações coletivas, como as religiosas, por exemplo, os fatos sociais para o pensador francês Gabriel Tarde não eram encarados como “coisas” passíveis de observação externa, mas como resultantes dinâmicas e transitórias de relações de forças baseadas em desejos e crenças [68]. Tarde privilegia, na sua análise, as engrenagens infinitesimais que compõem o real, investigando as diferenças que se formam e deformam entre múltiplos agentes, interagindo dinamicamente. Para ele, operam-se no mundo social os constantes jogos de diferença e de semelhança:

> “*A verdade é que a diferença vai diferindo, que a mudança vai mudando e que, tendo como fim em si mesmas, a mudança e a diferença atestam seu caráter necessário e absoluto (...). Se olharmos o mundo social, o único que nos é conhecido por dentro, vemos os agentes, os homens, bem mais diferenciados, mais caracterizados individualmente, mais ricos em variações contínuas, do que o mecanismo governamental, os sistemas de leis ou de crenças, os próprios dicionários e as gramáticas, mantidos pela participação dos agentes*” [68].

A compreensão do jogo complexo que envolve diferenças e semelhanças se dá a partir de dois conceitos fundamentais para Tarde: a *imitação* e a *invenção*. Ambos os conceitos se mostram presentes no Universo de Tarde, dos níveis físico-químicos, passando pelo nível vital até o nível social.

Para Tarde, o conceito de indivíduo é formado por um composto singular de fluxos diversos de desejos e crenças que o atravessam, e que produzem constante autodiferenciação. Desejos e crenças são, portanto, a base de toda a teoria social tardeana, as “quantidades psicológicas irredutíveis” [68] de todo o mundo vivo, as micro-unidades de comunicação entre os seres:

> *"A meu ver, os dois estados da alma, ou melhor, as duas forças da alma chamadas crença e desejo, das quais derivam a afirmação e a vontade, apresentam esse caráter eminente e distintivo. Através da universalidade de sua presença em todo fenômeno psicológico do homem ou do animal; através da homogeneidade de sua natureza de uma ponta a outra de sua imensa escala – indo desde a menor inclinação a crer e a desejar até à certeza e à paixão; através, enfim, de sua mútua penetração e de outros traços de semelhança não menos impressionantes, a crença e o desejo realizam no eu, em relação às sensações, precisamente o papel exterior do espaço e do tempo em relação aos elementos materiais* [68]".

É importante destacar, como percebem alguns pesquisadores da obra de Tarde [32], que a afirmação da crença e do desejo como quantidades psicológicas presentes em todos os indivíduos leva-nos inevitavelmente a um *psicomorfismo universal*, tornando-os potencialmente comparáveis em suas diferenças. Agentes são, assim, singularidades aproximáveis, na medida em que suas crenças e desejos podem ser transmitidos uns aos outros. Nesse ponto é importante afirmar que, para Tarde [68], as crenças e desejos se assemelhariam por força da influência de uns agentes sobre os outros. Quando dotados de crenças e desejos semelhantes, os seres se assemelham. Ainda segundo os mesmos autores, com referência a Tarde [32], um agente singular (um átomo, uma pessoa ou um animal) pode ser dotado de tamanha quantidade de crença e desejo em algo a ponto de provocar uma espécie de "força magnética" em outros agentes, cujas crenças e desejos ainda não tomaram forma determinada. Pela força da crença e do desejo de alguns agentes, o que seria um sistema de múltiplas singularidades poderá evoluir para estados transitórios de semelhanças adquiridas.

Para Tarde, as ideias ou as opiniões [66] não são as do seu "autor", pois não são propriamente "inventadas"; caberia dizer que são "descobertas", simplesmente trazidas à luz. Para ele, é como se as opiniões já estivessem presentes, ocultas, prontas para serem reveladas (tornadas objetivas). O indivíduo de Tarde *encontra* uma ideia. O cérebro seria apenas o local, o elo

consciente da grande cadeia da Opinião imanente a todas as coisas, e, portanto, a todo homem. Podemos dizer, assim, que os agentes em uma sociedade já possuem em si o potencial de todas as opiniões possíveis.

A sociedade, para Tarde, é "*uma coleção de seres na medida em que estão se imitando entre si*" [67]. A imitação, compulsória ou espontânea, eletiva ou inconsciente, transforma a descoberta individual num fato social. A opinião, a ideia ou o desejo de um torna-se progressivamente a opinião, a ideia ou o desejo de um grande número. O futuro normal de uma inovação, assim, é a sua propagação. É interessante notar que, na definição que faz Tarde para a sua sociedade, a aprendizagem social é parte do jogo constitutivo da própria sociedade. Só se pode falar em sociedade quando se considera a existência de imitação entre seus membros.

Mas sobre o que repousa o fenômeno da imitação, para Tarde? De forma radical, ele propõe que tal fenômeno repousa sobre a atividade da *sugestão*, que nada mais é do que uma forma de "hipnotismo" [68]. Para Tarde, de forma impressionante, o estado social é um estado hipnótico, no qual agentes influenciam e são influenciados por outros agentes. De nada importa a ilusão da autonomia e do livre-arbítrio, se somos mecanismos autômatos de geração e reprodução de ideias. Se a nova ideia ou opinião vai ao encontro de uma opinião existente, já partilhada por grande número de agentes, ela será reforçada. Caso seja contrária à opinião existente, representará uma inovação, e poderá expandir-se progressivamente até tornar-se ela própria uma opinião dominante, até ser ela também substituída por nova opinião ou ideia. Como veremos, esse conceito tardeano está no cerne do modelo que pretendemos propor.

Julgamos ainda interessante descrever em Tarde a preocupação com um novo conceito de Estatística, entendida como o instrumento *per se* de análise dos fatos sociais. Para Tarde, tão importante quanto conhecer a opinião de uma sociedade é conhecer a sua *opinião pública*, ou as opiniões que são tornadas públicas, sem esquecer, porém, das crenças e desejos que representam as aspirações coletivas dos seus agentes em determinado momento, e que muitas

vezes não são capturados em um processo de sufrágio universal. Conhecer, hoje em dia, o que pensam, por exemplo, os membros de uma rede social virtual não seria apenas realizar uma pesquisa de opinião, mas entender as suas crenças e desejos, e como eles se articulam dinamicamente na formação de uma “opinião coletiva”.

Em resumo, vê-se em Tarde alguns conceitos interessantes para o trabalho em tela. Em primeiro lugar, a consideração das crenças como um elemento universal em vários níveis de sistemas. Em segundo lugar, o conceito de imitação como fenômeno definidor de uma sociedade. Em terceiro lugar, a dinâmica social em que agentes se assemelham a partir de suas crenças, se aproximando pelas suas semelhanças e se afastando pelas suas diferenças. Em quarto lugar, o conceito de sugestão como base para a imitação, sugerindo o potencial de influência que um agente pode exercer sobre outro na propagação de suas crenças.

A Teoria Social da Aprendizagem, desenvolvida pelo psicólogo canadense radicado nos EUA, Albert Bandura [5, 6] também propõe que indivíduos aprendem através de um processo de observação e imitação de outros indivíduos, a que ele chamou de “modelação”. Para Bandura, o processo de aprendizagem social ocorre em quatro etapas: *atenção*, *retenção*, *reprodução motora* e *motivação*. Através da atenção, indivíduos observam características do comportamento a ser modelado. Em seguida, a informação é retida na memória de longo prazo. O observador deve ser capaz de reproduzir de forma motora o aprendizado observado (como, por exemplo, durante uma aula de esqui). Finalmente, na fase motivacional ou de reforço o observador recebe reforços positivos para o comportamento modelado.

As influências do ambiente foram profundamente pesquisadas por Bandura e seus seguidores, como no caso da influência da televisão em crianças e na violência doméstica. Hoje em dia, vários defensores da Teoria Social da Aprendizagem indicam que o crime é um produto do aprendizado de valores e comportamentos agressivos, em alguns casos observados dentro da própria família.

Entre os críticos da Teoria Social da Aprendizagem de Bandura estão os que acreditam que as diferenças individuais devem ser consideradas, e que a influência dos genes, do cérebro e das características individuais de aprendizagem tornam a resposta do individuo difícil de ser prevista. Acreditam, eles, que os comportamentos individuais não são somente aprendidos, mas também parcialmente herdados.

A Teoria Social da Aprendizagem evoluiu recentemente para ser chamada de Teoria Social Cognitiva, distanciando-se do seu início com ênfase mais comportamental. Hoje em dia, os seguidores de Bandura dão valor também aos aspectos cognitivos da aprendizagem, considerando que o funcionamento humano é produto de uma inter-relação dinâmica entre influências pessoais, comportamentais e ambientais [6].

De forma paralela aos estudos sociológicos e psicológicos, alguns cientistas da área de zoologia também têm investigado o comportamento dos animais sob o ponto de vista das estratégias de aprendizagem social [29, 43], identificando aqueles que melhor se adaptam à sobrevivência e evolução das espécies, curiosamente, observaram que a estratégia de aprendizagem por observação e imitação de comportamentos é amplamente difundida no meio animal.

A área de aprendizagem social também vem recebendo destaque recente na literatura econômica, a partir do estudo pioneiro de Banerjee [7]. Vários modelos matemáticos foram propostos na literatura recente [17, 28, 31, 33], principalmente utilizando métodos bayesianos como formalismo para análise estatística. O uso de modelos computacionais é uma alternativa aos métodos estatísticos, como demonstra o presente trabalho.

Outro trabalho recente na literatura é o de Cecille Roth [56], que propõe um modelo de rede de afiliação em que agentes e conceitos co-evoluem. O modelo de redes epistêmicas, como proposto no presente trabalho, combina a evolução de agentes e de suas crenças em um tipo único de estrutura. Outro trabalho identificado na literatura é o de Ang e Zaphiris [1], no qual a

abordagem baseada em agentes é aplicada para a simulação de redes sociais “on-line”.

Ao citar as principais áreas e fontes de interesse para o presente trabalho, nossa intenção foi, portanto, formular as bases para o desenvolvimento da pesquisa, extraindo dos vários campos brevemente descritos as intuições e conhecimentos necessários para fundamentar o trabalho a ser apresentado.

Em síntese, pode-se compreender os sistemas de aprendizagem social como sistemas em que os agentes se auto-organizam a partir do “ruído” (Atlan [2]) produzido pelas diferenças entre suas crenças (Tarde [68]), em um processo dinâmico de aprendizagem por observação e imitação (Bandura [5]).

2.5. Sistemas Imunológicos

Os sistemas imunológicos são um exemplo de sistemas complexos, nos quais diferentes partes interagem produzindo comportamentos coletivos emergentes. A motivação para o estudo dos sistemas imunológicos, no âmbito deste trabalho, é a possível modelagem do seu funcionamento através de redes que vamos descrever mais adiante.

Um sistema imunológico é uma escolha esplêndida para quem procura um sistema autônomo distribuído que sirva como base para a compreensão de conceitos organizacionais. Os autores Kovács & Ueno [41] argumentam que o sistema imunológico é um sistema epistêmico, capaz de perceber padrões, entender contextos, escolher diferentes tipos de ações, aprender e memorizar fatos. Da mesma forma, Atlan & Cohen [3] propõem que o sistema imunológico seja considerado um sistema cognitivo, que se auto-organiza e que produz sentido.

Na tentativa de responder, entre outras questões, como o sistema imunológico atinge a estabilidade antes da chegada de um antígeno, Niels Jerne [40] desenvolveu a teoria da rede funcional, ou teoria da rede idiotípica. Para Jerne, a teoria da seleção clonal olhava para os linfócitos sensíveis a antígenos como células independentes, e sua proposta incorporava a teoria da seleção clonal numa abordagem de rede mais abrangente.

Na formulação de sua rede, Jerne chamou de epítopos os determinantes gênicos carregados pelos antígenos. Os epítopos são pedaços da molécula do antígeno que possuem um padrão que pode ser reconhecido com alta precisão pelos padrões complementares do parátopo – nome dado aos locais de combinação do anticorpo. Os parátopos e os epítopos são essenciais para que o sistema imunológico realize o reconhecimento dos antígenos. Ao conjunto de epítopos existentes nas regiões variáveis de um conjunto de moléculas de anticorpo chamamos idiotipo; e cada epítopo idiotípico é chamado de idiótopo.

Na rede idiotípica é assumido que os repertórios de parátopos e idiótopos são da mesma ordem de grandeza. Vários estudos, citados em [53], mostram que os anticorpos possuem parátopo e idiótopo, e por isso Jerne considera que no sistema imunológico de um indivíduo qualquer idiótopo pode ser reconhecido por um conjunto de parátopos e que qualquer parátopo pode reconhecer um conjunto de idiótopos. O sistema imunológico é, portanto, uma grande rede complexa de parátopos que reconhecem idiótopos e de idiótopos que são reconhecidos por parátopos.

Trabalhos anteriormente desenvolvidos utilizando outras formas de modelagem podem, e serão, redefinidos utilizando o modelo conexionista de redes epistêmicas aqui apresentado [43, 71].

2.6. Construção Coletiva do Conhecimento Científico

Outro possível domínio de aplicação do modelo a ser proposto é o processo de construção coletiva do conhecimento científico por múltiplos agentes (cientistas), atuando de forma paralela e distribuída.

Segundo a corrente empirista da Epistemologia, o homem só pode ampliar o conhecimento descobrindo a ordem natural das coisas, através da observação. Francis Bacon, em seu "Novum Organum", pretendeu estabelecer uma nova lógica de base empírica, em contraposição à lógica aristotélica. Bacon propôs, em seu método, limpar a mente para a observação dos fatos, eliminando o que ele chamou de "ídolos da tribo, da caverna, do fórum e do teatro", que impediriam a busca da verdade. Seu método consistia essencialmente em observar os fatos, negando qualquer possibilidade de descrever a natureza a partir de silogismos aristotélicos. Uma de suas frases tornou-se famosa: *"A verdade emerge mais rapidamente do erro do que da confusão".* Trata-se, sem dúvida, de uma concepção interessante para o erro como fator de organização da atividade científica, muito semelhante à visão de Atlan, descrita anteriormente.

Popper [55] concebe o conhecimento científico como teorias conjecturadas livremente, sem ser necessário haver a indução lógica, e testadas empiricamente pelos experimentos cruciais que sempre podem refutá-las, como em Bacon são refutadas as hipóteses. O experimento capaz de refutar uma hipótese tornou-se a pedra fundamental da epistemologia contemporânea, na vertente do racionalismo crítico de Popper. Assim, a Ciência avança a partir da geração e refutação de teorias.

Para Thomas Kuhn [42], a ciência normal é feita de acúmulo de fatos em torno de uma teoria prevalente, até que, a partir de uma nova descoberta que não se enquadra no padrão anterior, dá-se um salto para um novo paradigma científico. A mudança de paradigma, conforme propõe Kuhn, ocorre pela destruição do paradigma anterior e pela construção de uma nova "visão de

mundo". Tal fenômeno lembra os ciclos de desorganização-reorganização de Atlan, mencionados anteriormente, que formam a base do processo de auto-organização, construindo um paralelo interessante entre os sistemas imunológicos e a evolução da Ciência, o que se pretende explorar nos exemplos de simulação do modelo a ser proposto.

Para o pensador húngaro Michael Polanyi, as crenças que os seres humanos encampam lhes são repassadas, na maioria das vezes, pela educação recebida. Mais especificamente, aponta ele que são os "exemplos" [54] que transmitem as crenças apreendidas. No caso da Ciência, toda a prática da pesquisa e verificação é transmitida pelo exemplo, e seus padrões são sustentados por "um contínuo relacionamento crítico dentro da comunidade científica" [54]. Segundo Polanyi, a comunidade científica se mantém unida e pacificada "mediante a aceitação conjunta das mesmas crenças fundamentais" [54]. Para ele, a liberdade da Ciência consiste no direito de buscar a exploração dessas crenças, e para isso é necessário certo grau de autogoverno. Além disso, afirma ele, a sociedade livre – da qual uma comunidade científica livre naturalmente faz parte – só pode ser defendida pelo reconhecimento expresso de crenças características encampadas em comum. Uma comunidade, portanto, organiza, disciplina e defende o cultivo de certas crenças, sustentadas pelos seus membros.

Polanyi, ainda no seu livro "A Lógica da Liberdade", defende o ponto de vista de que a Ciência é caracterizada como um empreendimento colaborativo, não sendo conduzida por esforços isolados de seus membros. Segundo ele, "no dia em que todas as comunicações entre os cientistas forem cortadas, a ciência praticamente paralisará". O princípio coordenativo da Ciência aflora assim, segundo Polanyi, com toda a sua natureza simples e óbvia.

> "*Ele consiste no ajuste das atividades de cada cientista aos resultados até então alcançados pelos outros. A cada passo, um cientista seleciona dos resultados obtidos pelos outros aqueles elementos que poderá usar com maior eficiência em seu trabalho, fazendo, dessa forma, a melhor contribuição possível para a Ciência; ele abre, dessa*

forma, o campo para que outros cientistas, por sua vez, façam suas contribuições ótimas – e assim indefinidamente" [54, pag. 71].

Podemos ver aqui, claramente, a lógica proposta por Polanyi da autocoordenação dos cientistas na busca da descoberta, consistindo simplesmente na extensão de um padrão desconhecido por meio de passos individuais, de acordo com a dupla condição de que cada novo passo sugerido possa ser rapidamente julgado quanto à sua correção ou não, e que cada passo novo seja com rapidez levado à atenção de todos os participantes e por eles considerado quando tiverem que dar o próximo passo [54, pag. 73].

Para ilustrar o conceito, apresenta Polanyi, o exemplo de um quebra-cabeças: supomos que nos coubesse montar um quebra-cabeça enorme, que consumiria dias ou semanas para se concluir. Imagine-se também que a tarefa é realmente urgente, pois a descoberta de um segredo importante depende da solução do problema. Uma vez convocada uma equipe de ajudantes, como organizá-los? Não faria sentido, segundo Polanyi, providenciar diversas cópias do problema, a serem entregues aos colaboradores para, ao final de determinado período de tempo, juntar as contribuições. Embora tal método pudesse permitir o uso de um grande número de contribuintes, o resultado não seria compensador. Cada ajudante iria observando a situação na medida em que o progresso fosse sendo feito e proporia a si mesmo novos problemas, de acordo com o último estágio da figura completada por todos os outros. Esse é um exemplo que distingue um processo auto-organizado de um processo centralizado, com uma autoridade central a distribuir tarefas específicas.

Portanto, segundo Polanyi, "desde que exista um propósito definido e estável em cada passo da descoberta científica, e que cada um deles possa ser competentemente julgado quanto à sua conformidade com esse propósito e ao seu sucesso em se aproximar dele, tais passos podem se somar espontaneamente para a mais eficiente busca da ciência" [54, pag. 76].

As comunidades científicas e suas dinâmicas são, portanto, uma das inspirações para o modelo a ser desenvolvido.

3. Redes Sociais

3.1. A Tradição Analítica

Os fenômenos sociais envolvem a interação de um número muito grande (porém finito) de agentes heterogêneos, com comportamentos que se desdobram no tempo, em vários níveis. Para dar uma ideia da complexidade do problema de descrever os fenômenos sociais, se considerarmos uma pequena organização (uma empresa, por exemplo), não temos como descrever seu comportamento no tempo sem antes compreender como se comportam os indivíduos que a compõem, as outras organizações com as quais ela interage, a estrutura regulatória do setor e as interações de todos esses componentes. Como bem observa o pesquisador de redes sociais Duncan J. Watts, cientista-chefe da empresa Google, Inc.[75]:

> "*Em uma analogia com a Física, é como se precisássemos resolver o equivalente à mecânica quântica, à teoria geral da relatividade e ao problema de múltiplos corpos, simultaneamente*".

Nos últimos 50 anos, os sociólogos vêm pesquisando a importância da interação de indivíduos, organizações e mercados na determinação do comportamento social coletivo. As ferramentas da "análise de redes" (*network analysis*) foram criadas como um instrumento quantitativo de pesquisa nesse campo, focando nas relações (arestas das redes sociais) entre indivíduos e/ou organizações (nós da rede social) [27].

A tradição sociológica da análise de redes focou seus estudos em dois objetivos principais:

(a) A identificação das estruturas que emergem da análise empírica das redes sociais do mundo real;

(b) A influência das estruturas em rede na definição das identidades e papéis de indivíduos e organizações sociais.

No primeiro caso, busca-se, através dos estudos empíricos, a compreensão de como os agentes sociais se relacionam, usando para isso uma modelagem formal baseada nas redes. Já no segundo caso, procura-se conhecer como a estrutura de uma rede social influencia o comportamento de indivíduos e organizações, seus papéis e identidades.

Como ferramenta de trabalho, sociólogos da tradição analítica em redes sociais foram buscar na Matemática, mais especificamente na Teoria dos Grafos, a base formal para suas análises.

Como crítica à tradição de análise de redes sociais, pode-se considerar que o estudo das interações sociais foi historicamente limitado aos fenômenos ligados à estrutura das redes, consideradas de forma estática, não se dando a devida importância à questão da dinâmica e evolução das redes sociais. Pouco se considerou, também, a interação dos aspectos cognitivos dos indivíduos e seus comportamentos individuais e coletivos, uma questão sem dúvida relevante para a compreensão das situações em que estes aspectos individuais são determinantes para a emergência de fenômenos coletivos.

3.2. Grafos Randômicos

O grande matemático suíço Leonhard Euler deu origem à chamada Teoria dos Grafos a partir da solução que propôs em 1736 para o “problema das pontes de Konigsberg”, no qual desejava saber se seria possível construir um caminho único por entre as pontes daquela cidade prussiana, de forma que a cidade não precisasse construir uma nova ponte para seus cidadãos. Como podemos ver na Figura 3.1, ele propôs representar as áreas distintas por nós (A, B, C, D) e os caminhos entre as pontes por arestas (de a a g), chegando a uma solução matemática para o grafo resultante [10]. Euler demostrou, de forma rigorosa, que não existia um caminho único por entre todos os nós do grafo. Como resultado adicional, a solução do problema das pontes de Konigsberg provocou o surgimento de uma nova e interessante área da Matemática, que hoje é um dos pilares da ciência das redes, mostrando que pequenas alterações na estrutura dos grafos (ou redes) permitem a emergência de novas e interessantes propriedades. Essa intuição ainda hoje é fundamental na pesquisa de redes complexas.

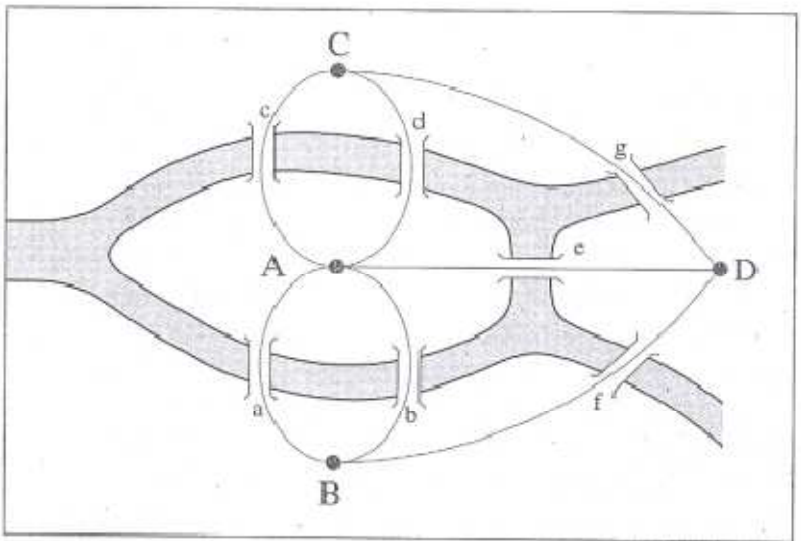


Figura 3.1 - Pontes de Konigsberg (adaptada de [10])

Até a metade do século passado, o objetivo da pesquisa em Teoria dos Grafos era apenas descobrir e catalogar as propriedades de vários tipos de grafos, sem muita preocupação com as eventuais aplicações práticas. Dois séculos

após o trabalho pioneiro de Euler, matemáticos mudaram o foco do estudo exclusivo das propriedades dos grafos para a questão de como se formam e evoluem as redes (grafos) na Natureza.

Um personagem importante nessa mudança de foco foi o profícuo matemático húngaro Paul Erdos, que até sua morte, em 1996, produziu mais de 1.500 trabalhos científicos, alguns dos quais são considerados *groundbreaking*. Em um conjunto de oito artigos, escritos em parceria com outro matemático húngaro, Alfréd Rényi, procurou responder à questão de como as redes se formam, estabelecendo as bases do que hoje se chama a teoria dos grafos randômicos (*random graph theory*) [23]. Em linhas gerais, Erdos propôs que as redes poderiam ser representadas como grafos, e que os nós da rede seriam conectados randomicamente por arcos, dando origem assim a uma estrutura complexa.

No exemplo abaixo (Figura 3.2), extraído de [10], vemos como uma rede social emerge a partir da interação de diferentes grupos (ou *clusters*) de amigos. No início, os grupos estão desconectados, mas a partir da interação durante uma festa, vão sendo criados arcos entre diferentes agentes (pessoas), até a emergência de uma rede (grafo) completamente conectada.

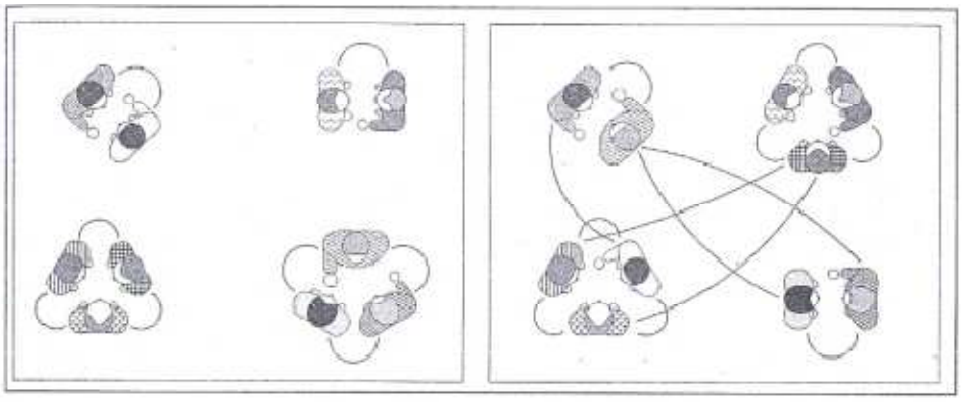


Figura 3.2 – Exemplo de Emergência de uma Rede em uma Festa Social (adaptada de [10])

Em Física, chama-se a esse momento em que um *cluster* único é formado uma "transição de fase", enquanto em Sociologia diz-se que foi formada uma

“comunidade”. Independentemente da terminologia, a emergência de uma nova propriedade da rede a partir da interação entre seus múltiplos componentes captura uma característica fundamental de diversos sistemas complexos existentes na Natureza.

A teoria dos grafos randômicos prevê que, para uma rede suficientemente grande, se criarmos os arcos de forma randômica, praticamente todos os nós terão aproximadamente o mesmo número de arcos. No caso de uma rede social, construiría-se idealmente uma sociedade democrática, na qual as pessoas (nós) teriam na média o mesmo número de relacionamentos. No caso da *Web*, teria-se os sites (nós) sendo “linkados” (arcos) pelo mesmo número de sites, sem preferências ou popularidades. Porém, isso não é o que ocorre nas redes do mundo real, como será visto a seguir.

Em síntese, os grafos randômicos são intuitivamente muito interessantes como modelos formais para as redes sociais, porém possuem limitações na modelagem de redes do mundo real, dada essa característica de distribuição média do grau das redes. Diversas extensões dos grafos randômicos foram propostas recentemente na literatura, como forma de reverter essas limitações [60].

3.3. O Mundo é Pequeno?

Stanley Milgram, professor de psicologia experimental da Universidade de Harvard, desenvolveu em 1967 um experimento para testar a interconectividade das pessoas em uma sociedade [45]. Seu objetivo era encontrar a "distância" entre duas pessoas quaisquer nos Estados Unidos. Em outras palavras: "quantos conhecidos são necessários para conectar dois indivíduos selecionados aleatoriamente?". Através de um interessante desenho experimental, no qual indivíduos selecionados aleatoriamente recebiam e reenviavam cartas, como em uma "corrente da felicidade", ele pode estabelecer que o número de intermediários entre duas pessoas era de 5,5, que mais tarde foi aproximado e ficou conhecido como os "seis graus de separação" [73]. O trabalho demonstra um fato curioso: a despeito do tamanho imenso da sociedade, aparentemente estamos muito perto uns dos outros, separados por apenas 6 conhecidos. Deu-se a essa propriedade das redes sociais (presente também em outros tipos de redes) o nome de "mundo pequeno" (*small world property*), e demonstrou-se que ela obedece à fórmula

$$\mathbf{d} = \log \mathbf{N} / \log \mathbf{k}, \qquad \text{(Equação 3.1)}$$

na qual **d** é o grau médio de separação, **N** é o número de nós da rede e **k** é o número médio de *links* (ou arcos) de um nó qualquer da rede [73].

O trabalho de Mark Granovetter [34] mostrou que, em uma rede social, existem formas diferentes de relacionamento entre amigos e conhecidos. No primeiro caso, formam-se laços fortes, pois os amigos são amigos dos seus amigos. No segundo caso, formam-se laços mais fracos (Figura 3.3). Nesse modelo, diferente dos grafos randômicos de Erdos, nos quais os nós têm em média o mesmo número de arcos, a rede social pode ser vista como uma coleção de grafos completos (*clusters* de amigos), ligados de forma fraca por arcos entre conhecidos de diferentes grupos. Granovetter demonstrou a importância dos

laços fracos na disseminação de rumores e na indicação para oportunidades de trabalho. Intuitivamente, os laços fracos são o nosso contato com o “mundo exterior” ao nosso círculo de amigos, alimentando-nos de novas informações sociais.

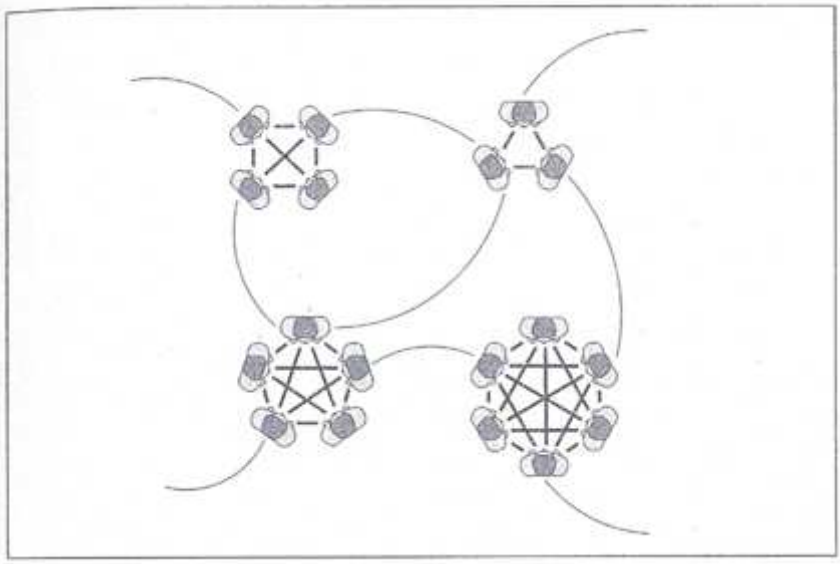


Figura 3.3 - Modelo de Laços Fracos (*Weak Ties*), inspirado por Granovetter [34]

3.4. A Formação de *Clusters*

De maneira a medir o grau de agregação (*clustering*) de uma rede, Watts e Strogatz [76] introduziram uma medida chamada de “coeficiente de agregação” (*clustering coefficient*), que relaciona a quantidade real de arcos que ligam um grupo de amigos à quantidade máxima possível (agregação máxima). De forma a modelar redes com alto grau de agregação (ver Figura 3.4), Watts e Strogatz, em seu trabalho seminal, construíram um modelo a partir de um círculo de nós, conectando cada um com seus vizinhos imediatos e os vizinhos imediatos de seus vizinhos. Para tornar a rede um “mundo pequeno”, basta adicionar alguns *links*, selecionando nós aleatoriamente. São esses os arcos (laços fracos) que diminuem drasticamente a separação média entre os nós da rede.

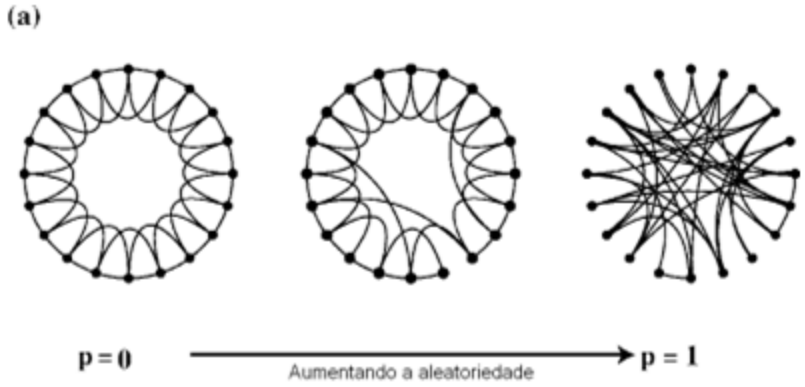


Figura 3.4 - Modelo Watts-Strogatz [76]

O mérito do trabalho de Watts e Strogatz [76] foi criar um modelo simples, ameno ao tratamento matemático-estatístico, que relacionasse o alto valor de *clustering* (agregação) de uma rede ao baixo valor da distância média entre os seus nós. Em termos intuitivos, quanto mais agregada é uma rede, menor é a distância média entre seus nós. No exemplo anterior, da Figura 3.4, a variável **p** é uma medida da aleatoriedade da rede. Inicialmente, trata-se de uma rede de alta agregação. Com a introdução aleatória de novos *links* entre os nós, a agregação diminui, aumentando a distância média entre os nós.

Barabási e coautores [9] estudaram uma rede de coautoria de 70.975 matemáticos, mapeando os seus trabalhos publicados entre 1991 e 1998 e formando uma rede de mais de 200.000 arcos. Comparando o coeficiente de agregação obtido com a hipótese de a rede ser formada aleatoriamente, Barabási encontrou um número 10.000 vezes maior, demonstrando a altíssima agregação existente na rede de coautoria de trabalhos de Matemática. Coautores de trabalhos científicos não seriam escolhidos aleatoriamente, portanto.

Mark Newman [46] também investigou, de forma independente, as redes de colaboração entre cientistas (no caso médicos, físicos e cientistas da computação), e reforçou a partir dos seus resultados empíricos que a ciência é conduzida por grupos densamente agregados de cientistas, conectados eventualmente por laços fracos.

Um desenvolvimento paralelo importante na literatura de redes [8] partiu da observação de que, em redes do mundo real, alguns nós possuem mais "popularidade" (*links* apontados para eles) do que outros. São os chamados *hubs*. Por exemplo, o site do Google (www.google.com) é seguramente mais popular do que o da UniCarioca (www.unicarioca.edu.br). A distribuição do número médio de *links* de um nó (k) dessas redes obedece a uma "lei de potência" (*Power Law*) da forma assintótica abaixo:

$$\mathbf{P(k) \sim k^{-\alpha}} \qquad \text{(Equação 3.2)}$$

Em outras palavras, a probabilidade de um nó escolhido aleatoriamente ter um grau **k** decai com a potência de **k**, na qual o expoente **α** (tipicamente entre $2 < \alpha < 3$) determina a taxa de decaimento. Alguns poucos nós têm alta popularidade, e muitos nós têm baixa popularidade. Uma característica importante das distribuições que obedecem à lei de potência é que, quando plotadas em um uma escala logarítmica dupla, formam uma linha reta com inclinação negativa **α** (Figura 3.5a), contrastando com uma distribuição normal (Figura 3.5b), na qual existe claramente um ponto de corte acima do qual a

probabilidade de existir um nó com grau maior é efetivamente zero. A existência de um ponto de corte implica em uma escala para o grau de uma rede (k), e como as redes que obedecem à lei de potência não possuem esse ponto, são chamadas de “livres de escala” (*scale-free networks*).

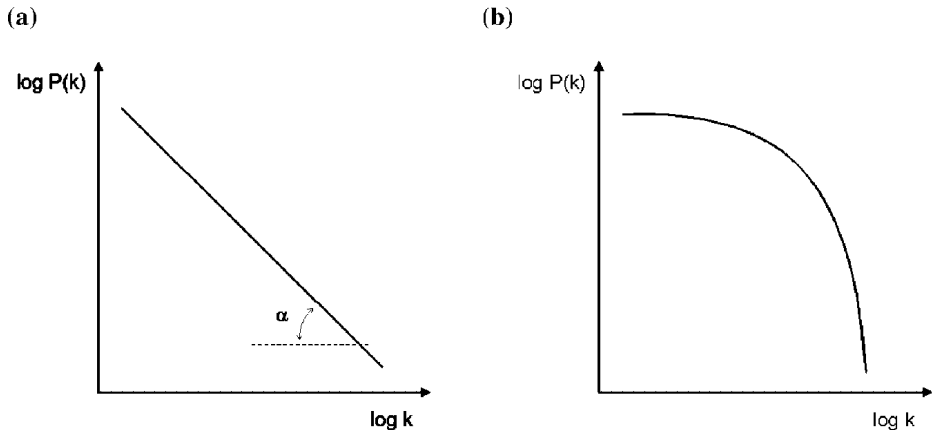


Figura 3.5 – (a) Rede com Lei de Potência (*scale-free*)
(b) Rede com distribuição normal [adaptado de 74]

Barabási e Albert [8] também sugeriram um modelo matemático simples para descrever as suas observações empíricas (no caso de redes de colaboração de atores americanos e de uma parte da rede *World Wide Web*), considerando dois mecanismos: crescimento populacional dos nós e vinculação preferencial dos novos nós (*preferential attachment*). A intuição no caso do primeiro mecanismo é óbvia: redes reais não são estáticas, e crescem na medida em que novos membros se unem a ela. O segundo mecanismo expressa a noção (também intuitiva) de que os novos nós tendem a se conectar preferencialmente com aqueles já bem conectados. Os dois autores definiram a probabilidade $P(k_i)$ de um nó **i** já existente, com k_i arcos, receber um novo arco como

$$\mathbf{P(k_i) = c\ k_i} \qquad \text{(Equação 3.3)}$$

onde **c** é uma constante normalizadora. Barabási e Albert [8] demonstraram que, considerando-se um período longo de tempo, a distribuição do grau **k** de uma rede que exiba vinculação preferencial converge para algo na forma da equação 3.2, com um expoente **α** = 3.

Podemos resumir a comparação entre as redes randômicas e as redes livres de escala na Figura 3.6, adaptada de [10]:

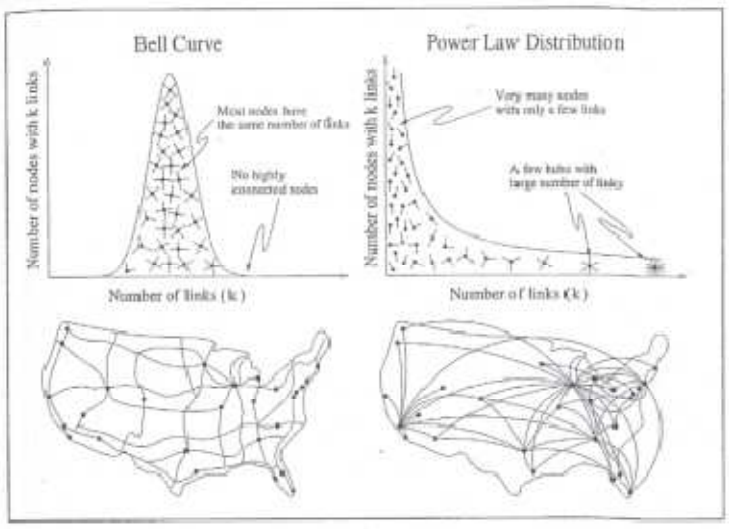


Figura 3.6 – Redes randômicas X redes *scale-free* (adaptada de [10])

Em uma rede randômica, o pico da distribuição do número médio de arcos de cada nó (k) significa que uma ampla maioria de nós tem o mesmo número de arcos (mesma “popularidade”), e nós que se desviam dessa média são raros. Possuem, portanto, uma escala (limite). Na Figura 3.6 (lado esquerdo), Barabási [8] dá o exemplo da rede rodoviária norte-americana, em que cada cidade (nó) no mapa possui um número de rodovias (arcos) que as interligam. Cada cidade, portanto, é servida por um número aproximadamente igual de rodovias. Em comparação (Figura 3.6, lado direito), uma rede livre de escalas descreve melhor o sistema de vias aéreas norte-americano, no qual a maioria das cidades pequenas possui aeroportos pouco movimentados, que são servidos por alguns poucos grandes aeroportos (*hubs*) que concentram a maior parte dos vôos. A verificação da existência de liberdade de escala em grande parte das redes do mundo real é a indicação da presença de importantes princípios organizativos na sua evolução.

Como consequência do desenvolvimento recente dos modelos para análise da estrutura e evolução das redes, diversos trabalhos foram publicados sobre as propriedades de diferentes redes do mundo real, entre os quais podemos citar, a partir de Watts [74], as redes de reações metabólicas, redes de regulação

genética, redes de interação de proteínas, redes neuronais biológicas, redes de alimentos (*food webs*), redes de transporte e redes de colaboração entre *board* de diretores de organizações e de cientistas, além da própria *Web*. Essa lista, que cresce a cada ano, não inclui ainda muitos exemplos de redes sociais devido à dificuldade de registro das interações sociais em grande escala, dificuldade essa que vem sendo superada pela crescente migração das atividades humanas para o mundo *on-line*, o que oferece aos cientistas uma ótima oportunidade para futuras pesquisas.

Sem dúvida, a compreensão da estrutura e evolução das redes do mundo real é uma questão legítima para a pesquisa científica, uma vez que ajuda a esclarecer a relação entre o comportamento individual (micro) e o comportamento coletivo (macro), um dos mecanismos fundamentais do funcionamento social. No entanto, como ponderado por Watts [74], “a relação entre a estrutura da rede e as suas consequências dinâmicas não é nada trivial, e depende do contexto em que os processos dinâmicos são investigados”. Assim, a identificação de propriedades universais para a dinâmica das redes do mundo real é uma “área de estudo desafiadora” [74], e deve partir de exemplos simples que nos trarão algum *insight*.

3.5. Propagação de Doenças e Contágio Social

Um dos exemplos da dinâmica das redes complexas é a propagação de doenças, que é muito semelhante à propagação de ideias. Muitas epidemias humanas são propagadas através do contato entre seres humanos, ou entre seres humanos e outros animais. Natural, portanto, que modelos epidemiológicos sejam candidatos óbvios a uma das aplicações mais naturais da ciência das redes sociais. No entanto, segundo Watts [74, 75], os modelos matemáticos clássicos da epidemiologia foram formulados em termos estatísticos, baseados em contatos aleatórios entre um conjunto de indivíduos, com pouca ênfase na estrutura implícita das redes existentes entre seres humanos. Tais modelos baseiam-se no conceito de "limiar crítico" (*critical threshold*) e de "taxa de propagação" (*spreading rate*). Em resumo, consideram que cada indivíduo tem uma diferente resistência a um vírus, ou a uma ideia (no caso de difusão de ideias e crenças). Assumindo essas diferenças individuais, modelos de difusão atribuem um *threshold* a cada indivíduo, quantificando as chances dele adotar uma determinada inovação. Cada inovação (ou vírus) possui uma bem definida "taxa de propagação", representando as chances de que ela seja adotada (ou contraído, no caso do vírus), uma vez que um indivíduo tenha contato com ela (ou ele). Dessa forma, nos modelos clássicos, para que possamos estimar se uma doença ou inovação será propagada precisamos apenas conhecer sua taxa de propagação e seu limiar crítico.

Pastor-Satorras & Vespignani [52] investigaram a propagação de doenças em redes randômicas exibindo comportamento de lei de potência (*scale-free networks*), que possuem, portanto, diversos *hubs*, e descobriram, para surpresa de todos, que em desacordo com as previsões do modelo de limiar crítico, em redes reais livres de escala a alta virulência não garante a propagação de um vírus (ou de uma ideia), nem a baixa virulência impede sua propagação. A fonte de tal comportamento inesperado é a característica topológica das redes livres de escala, que são dominadas por nós com alta concentração de arcos (*hubs*). Esses, devido às suas características, possuem uma alta possibilidade de serem infectados, e uma vez infectados transmitem o

vírus para um número muito grande de outros nós. *Hubs* altamente conectados, portanto, oferecem um meio único e privilegiado através do qual os vírus (ou ideias) podem persistir e se espalhar pela rede.

Uma classe interessante de modelos análogos ao da difusão de doenças é a de modelos de contágio social, que envolvem a tomada de decisão coletiva. Indivíduos que tomam decisão frequentemente observam a ação ou as decisões de seus vizinhos ou pares. Podemos modelar tal situação imaginando que as decisões são “comunicadas” entre indivíduos, em um modelo análogo ao da difusão de doenças. Watts [73] descreve exemplos do uso dos modelos de contágio social em casos de inovações, tecnologias competitivas, modas culturais, normas sociais, cooperação, desordem social e mercados financeiros.

Um ponto importante, que difere os modelos de contágio social dos modelos de difusão de doenças, é que esses últimos são considerados processos sem memória. A probabilidade de ser infectado por um vírus é a mesma, independentemente da quantidade de exposição anterior ao vírus. No caso dos processos de contágio social, indivíduos que tomam decisões são afetados tanto pelas interações passadas quanto pelas presentes.

Na Figura 3.7, extraída de [73], podemos ver a diferença entre os dois tipos de modelos. A probabilidade acumulada de infecção no modelo de propagação de doenças cresce linearmente, enquanto que no modelo de contágio social a probabilidade de se acreditar em um rumor, por exemplo, pode mudar drasticamente após uma segunda ou terceira exposição ao assunto, formando um gráfico de *threshold* como o da Figura 3.7(b), que representaria um “limiar da crença” do indivíduo.

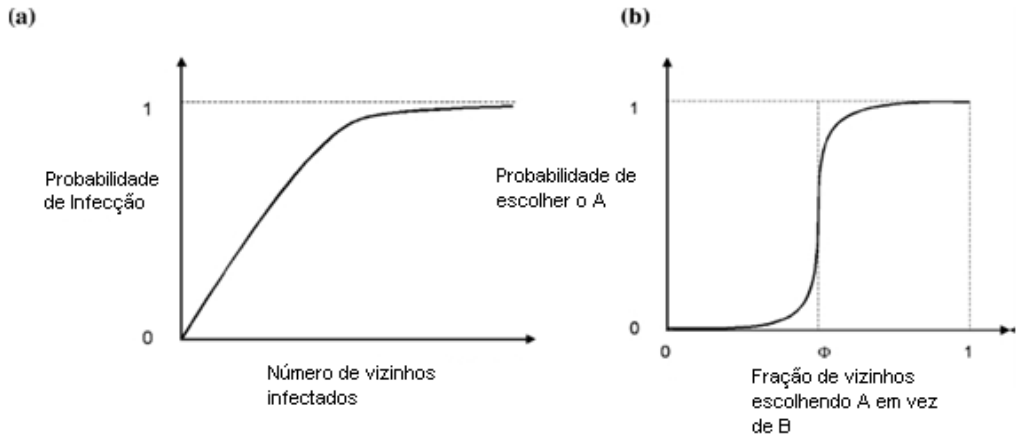


Figura 3.7 – Modelos de propagação de doenças X modelos de propagação de crenças (adaptada de [73])

Considere-se, no modelo a ser proposto, que esse limiar varia para cada indivíduo, e que é função da importância que ele atribui ao indivíduo que transmite a crença.

Esse é um ponto importante a ser ressaltado, mais uma vez: nos modelos atuais, é dada pouca consideração às características cognitivas do indivíduo no processo de propagação de ideias, e é nesse aspecto que o processo poderá ser mais bem modelado caso sejam considerados dois aspectos adicionais, não contemplados na literatura corrente:

(a) O efeito da importância do agente que transmite a ideia (ou crença) para o agente que a recebe, a exemplo do que foi discutido por Tarde (com o conceito de “imitação”) e Bandura (com o conceito de “modelação”);

(b) A aceitação de uma ideia (ou crença) como um processo de aprendizagem (ou “reforço”, como na Teoria Social da Aprendizagem de Bandura).

Nesse ponto, converge-se para a ideia de um modelo que faça uso das ideias de Tarde, Bandura e Atlan (“aprendizagem dirigida em sistemas biológicos”) e

que considere que as ideias e crenças são livremente propagadas entre agentes através de um processo combinado de etapas de aprendizagem individual e aprendizagem coletiva. Como será visto a seguir, no modelo de redes epistêmicas a ser proposto o limiar crítico da Figura 3.7 é um limiar não só de um indivíduo isoladamente, mas também da rede social coletivamente, tornando-o, sob este aspecto, um modelo bem original.

Resta comentar o crescimento recente das pesquisas em *Webs* sociais ou, como denominam alguns pesquisadores, a *Web* 2.0. A disponibilidade de grandes quantidades de dados transacionais nas redes via *Web* abre a possibilidade de pesquisar como diferentes usuários se relacionam, a partir de seus interesses e preferências. Esse é um dos desafios deste trabalho, e o modelo proposto é uma tentativa de formalizar aspectos dinâmicos da aprendizagem social presentes na *Web* 2.0.

4. Modelo

Para a construção do modelo apresentam-se algumas definições básicas seguidas de premissas sobre o processo de geração e comunicação de crenças por múltiplos agentes, ilustrando-se, sempre que possível, com exemplos. Em seguida, apresentam-se os elementos principais do modelo, e encerra-se o capítulo com a descrição da dinâmica do modelo.

4.1. Definições Básicas

Def. 4.1. Par epistêmico

Um **par epistêmico** é uma dupla de vetores <**a**,**c**>, **a** $\in$ $\mathbf{A}^{n}$, **c** $\in$ $\mathbf{C}^{m}$, onde $\mathbf{A}^{n}$ e $\mathbf{C}^{m}$ são chamados de subespaços epistêmicos. Um par epistêmico é capaz de representar qualquer tipo de relação (causal, temporal ou lógica, por exemplo) entre dois elementos **a** e **c**, podendo comportar a expressão declarativa de qualquer forma de conhecimento (daí a escolha do nome "epistêmico).

Por exemplo, pode-se expressar a crença (falsa) de que a Terra é o centro do Universo por meio do seguinte par epistêmico:

<"A Terra é o centro do Universo", V>.

A expressão "Terra é o centro do Universo" é o antecedente **a** e o valor-verdade V é o consequente **c**.

Dois cientistas, por exemplo, podem realizar dois experimentos e a partir de uma mesma observação chegar a duas explicações diferentes:

Experimento 1 = <observação, explicação1>
Experimento 2 = <observação, explicação2>

No exemplo anterior, os experimentos constituem-se como pares epistêmicos, e para um mesmo antecedente pode-se obter dois consequentes distintos.

Em resumo, o conceito de par epistêmico é amplo o suficiente para que o modelo possa representar a expressão de diversos atributos (antecedentes) e seus valores (consequentes) no domínio de aplicação escolhido. Nesse sentido, pode ser visto como uma generalização da modelagem de Axelrod [4] para a representação da influência social em ambientes multiculturais. A definição genérica dos pares epistêmicos também permite a generalização do modelo de formação de consenso de DeGroot [19], que representa um caso especial em que o antecedente tem valor 1, e o consequente assume valores 0 ou 1. No Capítulo 6 será apresentado um exemplo da formação de consenso usando as redes epistêmicas.

Def. 4.2. Espaço epistêmico

Um **espaço epistêmico** $\mathbf{E^{nXm}}$ é um produto cartesiano $\mathbf{A^n}$ X $\mathbf{C^m}$ formado por pares epistêmicos. As **n x m** dimensões referem-se aos atributos representáveis do conhecimento que se deseja expressar.

Pode-se dizer que o espaço epistêmico, portanto, associa a um conjunto de atributos expressáveis um conjunto de valores possíveis, permitindo uma ampla gama de representações.

Def. 4.3. Agente Epistêmico

Um **agente epistêmico** é um construto capaz de gerar, comunicar, revisar e representar pares epistêmicos.

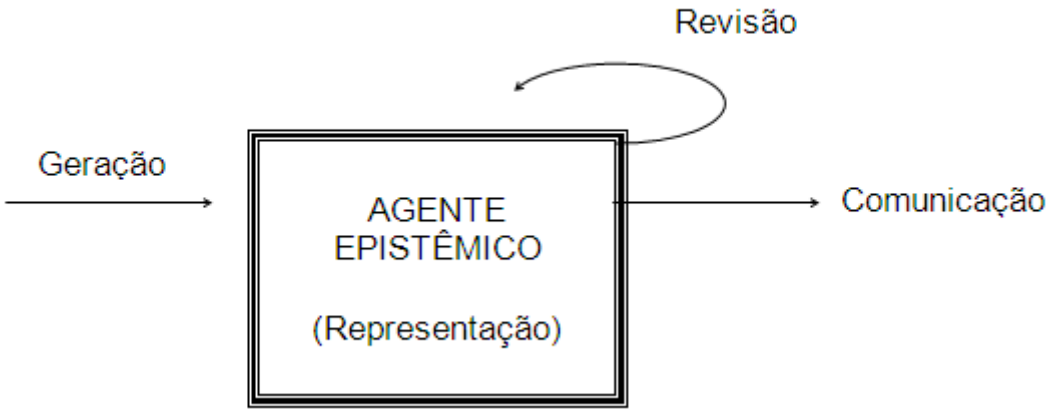


Figura 4.1 – Funcionalidades de um Agente Epistêmico

A geração de pares epistêmicos pode acontecer espontaneamente, no caso em que um agente epistêmico já “nasce” com um conjunto de pares epistêmicos pré-definidos, ou por aprendizagem individual, através da qual ele é treinado para responder sempre da mesma forma. A revisão de pares epistêmicos em um agente epistêmico ocorre por observação e imitação de outros agentes. A comunicação de pares epistêmicos é a forma através da qual um agente epistêmico interage com outros agentes epistêmicos.

Def. 4.4. Rede Epistêmica

Uma **rede epistêmica** é um conjunto de agentes epistêmicos que interagem através da comunicação de pares epistêmicos entre si.

Na figura 4.2, pode-se observar uma rede epistêmica com três agentes:

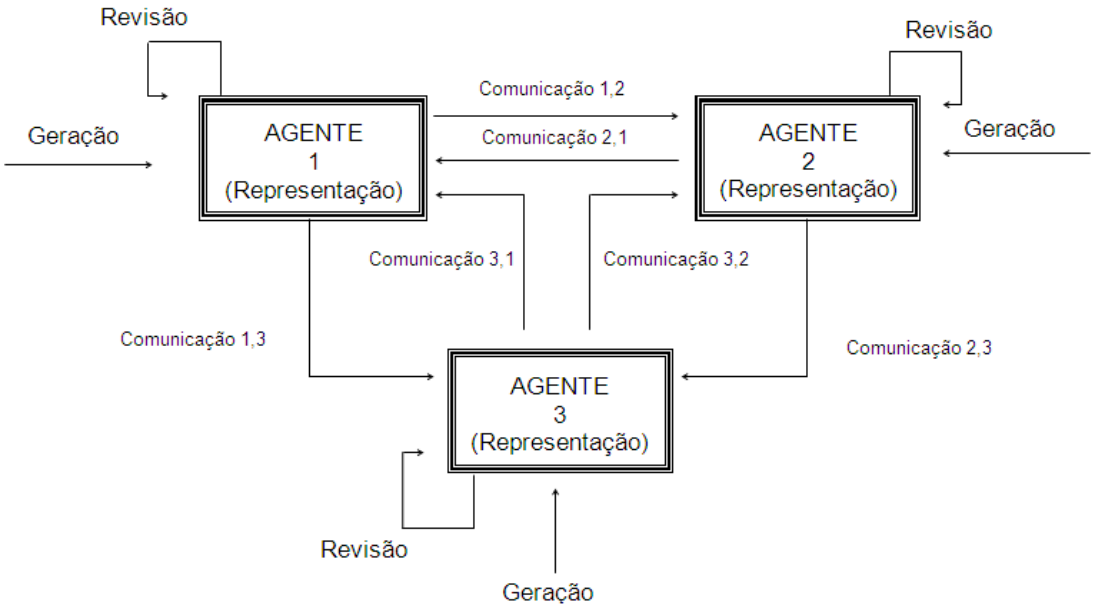


Figura 4.2 – Funcionalidades de uma Rede Epistêmica com Três Agentes Epistêmicos

Como os pares epistêmicos de cada agente podem representar qualquer forma de conhecimento, a rede é chamada de epistêmica, pois organiza-se a partir das várias comunicações e revisões de pares epistêmicos entre seus agentes, constituindo-se em um modelo dinâmico para a aprendizagem social.

Def. 4.5. Crença

A **crença** de um agente epistêmico é formada pelo conjunto de pares epistêmicos representados por este agente em um dado instante de tempo.

Considere-se a seguir uma rede epistêmica com três agentes e as seguintes crenças de cada um:

Figura 4.3 – Agentes e suas crenças

Def. 4.6. Conhecimento Individual

O **conhecimento individual** de um agente epistêmico é formado pelo subconjunto das suas crenças que é comum a, pelo menos, outro agente.

A explicação por trás dessa definição é simples: por mais convicção que um agente tenha em sua crença, o conhecimento é algo objetivo, e necessita de justificação. Na rede epistêmica, a justificação de um conhecimento é a chamada “realidade do outro”. As redes epistêmicas podem ser vistas, portanto, como um modelo de justificação de crenças através da interação de diferentes agentes epistêmicos. Tal definição vai ao encontro da intuição de autores como Martin Buber [16], para quem o indivíduo se define a partir das suas relações com os outros.

No exemplo anterior, o conhecimento individual de cada agente é representado na figura 4.4:

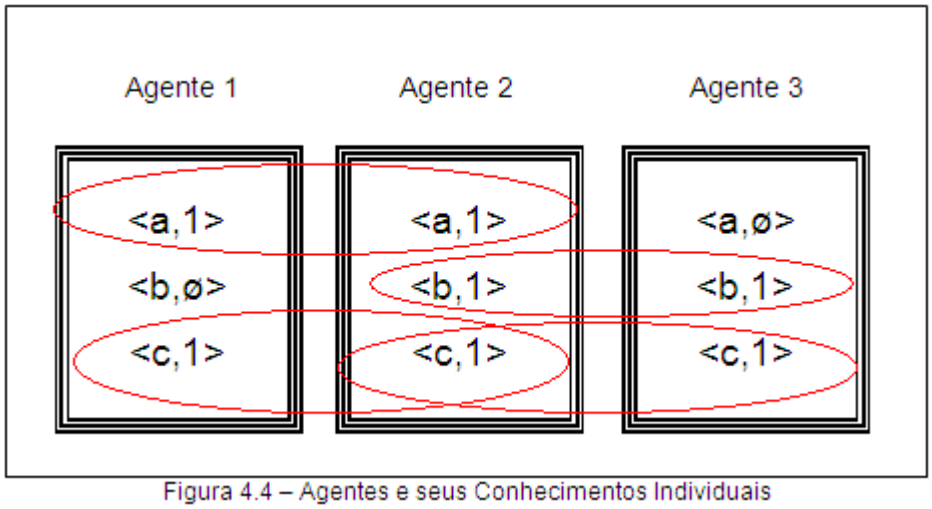


Figura 4.4 – Agentes e seus Conhecimentos Individuais

Def. 4.7. Conhecimento coletivo

O **conhecimento coletivo** em uma rede epistêmica é a união dos conhecimentos individuais de seus agentes.

No exemplo utilizado, o conhecimento coletivo da rede é expresso pelos pares epistêmicos <a,1>, <b,1> e <c,1>, como se observa no diagrama abaixo (Figura 4.5):

Figura 4.5 – Agentes e o conhecimento coletivo da rede

Def. 4.8. Conhecimento comum

O **conhecimento comum** em uma rede epistêmica é a interseção dos conhecimentos individuais de seus agentes.

Na rede do exemplo utilizado, o único par epistêmico que constitui o conhecimento comum da rede epistêmica é o par <c,1>, conforme se observa na Figura 4.6:

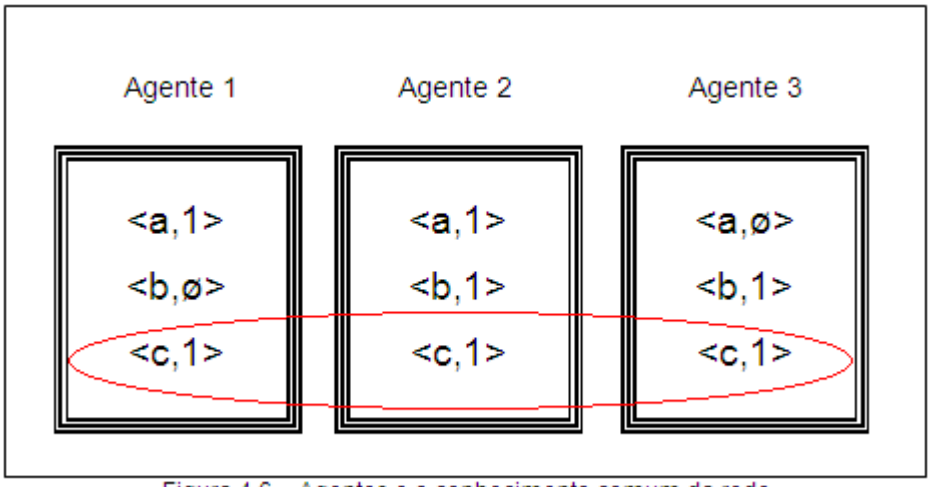


Figura 4.6 – Agentes e o conhecimento comum da rede

4.2. Premissas do Modelo

Apresentam-se, agora, algumas premissas contempladas no modelo proposto:

Premissa 4.1. Agentes epistêmicos constroem o conhecimento de forma paralela e distribuída.

O conhecimento evolui por um processo em que diferentes agentes epistêmicos possuem diferentes características e interesses na geração e representação paralela e distribuída do conhecimento, na forma de pares epistêmicos. Por exemplo, no campo da Sociologia da Ciência, diferentes pesquisadores atuam simultaneamente, com várias teorias convivendo de forma espacialmente distribuída pelos grupos de pesquisa existentes. No sistema imunológico, cada agente tem a competência de identificar um escopo de bactérias, em qualquer parte do corpo.

Premissa 4.2. Agentes epistêmicos possuem uma habilidade particular para geração e representação de pares epistêmicos.

Como consequência dos seus parâmetros, que definem suas habilidades específicas, um agente é capaz de gerar a cada instante um par epistêmico, que pode, por exemplo, ser associado a uma observação e sua hipótese explicativa, no caso do conhecimento científico, ou a um par <parátopo, idiótopo>, no exemplo de um sistema imunológico. Diferentes representações internas permitem que, no modelo, dois agentes distintos possam gerar, para o mesmo antecedente **a**, consequentes **c** diversos (representando diferentes visões do mundo).

Premissa 4.3.. Agentes epistêmicos geram pares epistêmicos de acordo com uma particular distribuição de frequência.

Para garantir a generalidade do modelo proposto, sendo fiel à distribuição e ao paralelismo, temos que admitir que cada agente epistêmico tem a sua

característica temporal na geração de pares epistêmicos. Cada um no seu próprio tempo e todos de forma paralela, essa é a essência do modelo. No exemplo de um sistema imunológico, isso equivale a que um clone de linfócitos possa ser temporalmente independente dos outros no reconhecimento de antígenos (representados como **a**) ameaçadores. Já na instância da geração do conhecimento científico é bastante conhecido o fato de que alguns pesquisadores produzem mais resultados que outros, obedecendo ao seu próprio ritmo de produção de novos conhecimentos científicos.

Premissa 4.4. Agentes epistêmicos geram pares epistêmicos a partir de um ponto (chamado “foco”) de um espaço $\mathbf{A}^{n}$, conforme uma distribuição de probabilidades em torno deste ponto (chamado “escopo”).

O agente percebe a realidade (gera um par epistêmico) a partir de um foco, de acordo com o seu escopo. No exemplo do sistema imunológico, este escopo corresponderia ao limite de reconhecimento dos parátopos pelos idiótopos. Quanto maior o escopo de um linfócito mais antígenos ele conseguiria reconhecer. Este parâmetro fornece a medida de dispersão ou de generalização deste linfócito no reconhecimento de antígenos. Por outro lado, no exemplo da geração de conhecimentos científicos, existem cientistas que atuam em um campo científico (“foco”) com maior amplitude de observação (escopo mais amplo) e outros com menor amplitude de observação (escopo mais estreito).

Premissa 4.5. Agentes epistêmicos comunicam suas crenças (pares epistêmicos) a outros agentes conforme uma distribuição particular de frequência, variável com a evolução do próprio agente na rede epistêmica.

Em determinado momento, durante o ciclo de geração de pares epistêmicos, um agente comunica um par que representa sua crença. No caso do sistema imunológico isso equivale à frequência com que um clone de linfócitos libera no plasma proteínas sinalizadoras da presença do antígeno que este clone é capaz de reconhecer. No exemplo do conhecimento científico, um pesquisador

autoconfiante comunica com mais frequência seus resultados, através de publicações científicas.

Premissa 4.6. Agentes epistêmicos interagem entre si a partir de comunicações seletivas.

As comunicações conforme relatadas na premissa anterior não necessariamente atingem todos os agentes da rede, podendo ser seletivas. No caso do sistema imunológico esta hipótese corresponde à emissão de uma proteína sinalizadora distribuída localmente em um processo inflamatório, ou generalizadamente em um processo infeccioso generalizado. No caso dos pesquisadores científicos, pode haver uma comunicação geral (publicação em revista) ou uma comunicação mais restrita (simpósio).

Premissa 4.7. Agentes epistêmicos atribuem importância variável aos outros agentes epistêmicos da rede, conforme a proximidade de suas crenças.

Pesquisadores com resultados mais relevantes recebem um maior reconhecimento da sua comunidade e atraem novos pesquisadores, formando “escolas de pensamento”. Já no exemplo de sistemas imunológicos, determinados clones de linfócitos, que reconhecem os mesmos antígenos, se agregam, permitindo uma capacidade maior de inibir o antígeno.

Premissa 4.8. Agentes epistêmicos na rede se auto-organizam em função de suas crenças.

Na comunidade científica os pesquisadores aglutinam-se em *clusters* ou grupos de pesquisa, que tendem a manter crenças semelhantes. No sistema imunológico clones que reconhecem o mesmo antígeno reforçam-se mutuamente através de proteínas sinalizadoras. Essa é uma propriedade estrutural (“auto-organização”) que emerge naturalmente no modelo proposto.

Premissa 4.9. Agentes epistêmicos podem surgir ou desaparecer, representando o aprendizado topológico da rede epistêmica.

No caso do sistema imunológico, novos clones são criados por hipermutação, enquanto clones que perdem função são eliminados ao longo da vida do indivíduo. No caso da construção do conhecimento científico surgem novos discípulos de uma escola de pensamento ou morrem pesquisadores antigos. As condições de surgimento e aparecimento de agentes epistêmicos são função da própria dinâmica do modelo, a partir da necessidade de mais flexibilidade de geração e representação do conhecimento.

Após as premissas, apresentam-se os elementos principais do modelo.

4.3. Elementos Principais do Modelo

Na representação do modelo proposto, consideram-se dois níveis: o nível individual do agente e o nível coletivo da rede epistêmica. A rede epistêmica é uma "rede de redes", como veremos a seguir.

No nível individual, cada agente epistêmico é representado por uma *rede neuronal de aprendizado por retro-propagação de erros* [44, 70], capaz de produzir um vetor **c** (de dimensão m) na camada de saída a partir de um vetor **a** (de dimensão n) na camada de entrada. Internamente, as conexões **B** são representadas por pesos variáveis, dando uma medida da habilidade particular para a geração e representação das crenças do agente epistêmico (Figura 4.7).

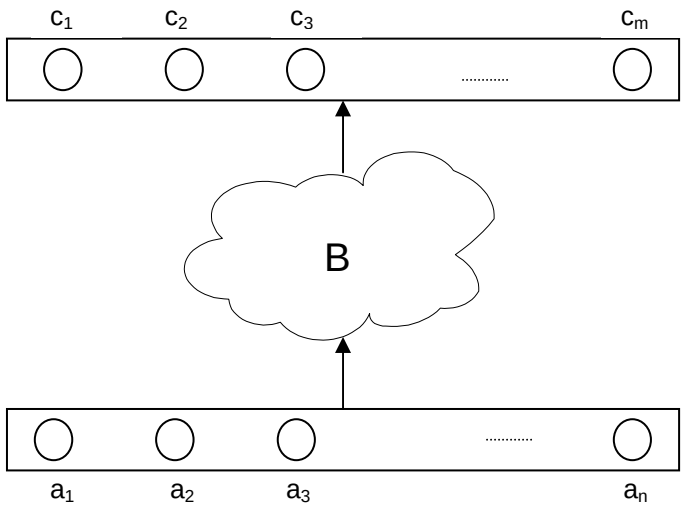


Figura 4.7 – Rede Neuronal de Aprendizado por Retro-propagação que representa um agente epistêmico

No nível coletivo da rede epistêmica, os nós correspondem aos agentes epistêmicos (denominados de *epistrons*, a partir da expressão *epistemic neurons*, e simbolizados como ε_i) e as conexões W_{ij} entre o epistron ε_i e o epistron ε_j representam a proximidade entre suas crenças. Forma-se, assim, uma *rede genérica de aprendizado competitivo* [44, 70], ilustrada na Figura 4.8. As conexões aferentes de um epistron representam a importância que ele tem

para os outros epistrons da rede, enquanto que as eferentes representam a importância que ele dá aos outros epistrons da rede epistêmica. Os W_{ii} dão uma medida da importância que o epistron atribui a si próprio, que, por sua vez, é função da importância que os outros epistrons atribuem a ele, como veremos a seguir.

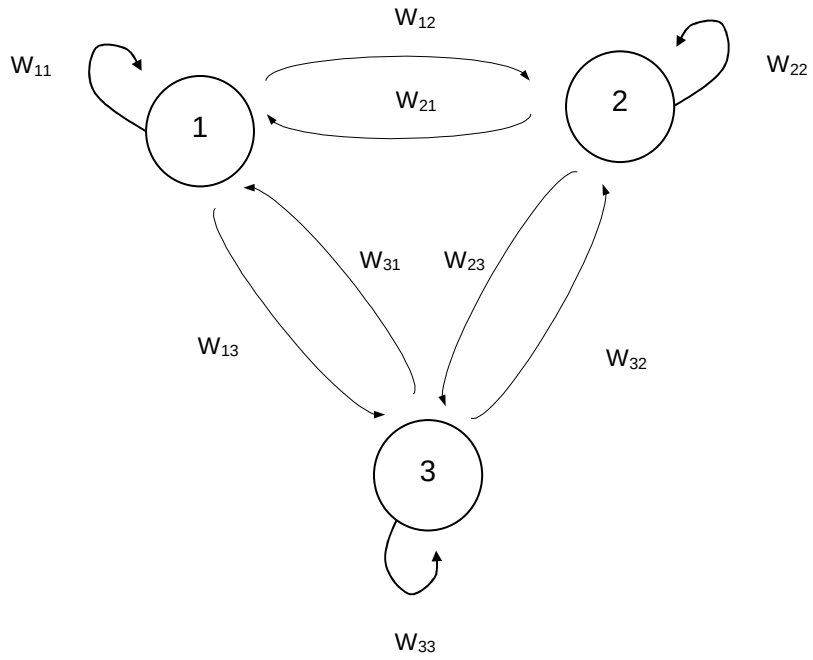


Figura 4.8 – Rede Neuronal Genérica de Aprendizado Competitivo que representa a rede epistêmica com três agentes epistêmicos (epistrons)

O aspecto interessante da representação do modelo em dois níveis de redes neuronais é a possibilidade de combinação de dois tipos de aprendizagem: no primeiro nível (nível do agente), utiliza-se a aprendizagem por *back-propagation*. No nível coletivo (nível da rede), utiliza-se a aprendizagem competitiva (regra de Hebb). Esta escolha de representação dá ao modelo originalidade e flexibilidade para tratar a evolução da rede epistêmica como um processo misto de aprendizagem neuronal.

4.4. Dinâmica do Modelo

O modelo possui uma dinâmica que pode ser descrita em duas etapas: a dinâmica individual de um epistron e a dinâmica coletiva da rede.

4.4.1. Dinâmica individual de um agente epistêmico

Trata-se da geração individual, conforme uma distribuição particular de frequência, de pares epistêmicos, de acordo com o foco e o escopo do epistron (bola n-dimensional de raio ρ, no espaço $\mathbf{A^n}$, em torno de um ponto central $\mathbf{a_f}$ inicialmente determinado para cada epistron) (Figura 4.9).

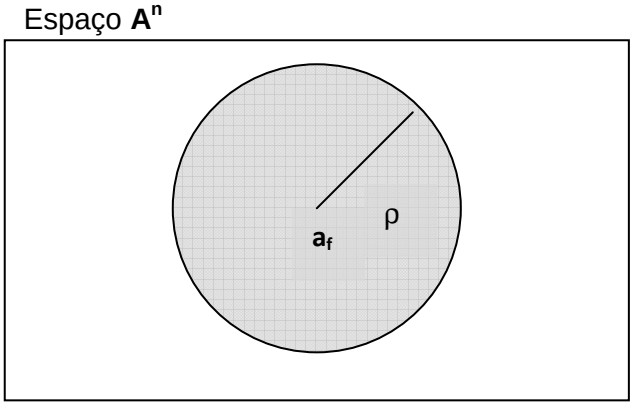


Figura 4.9 – Foco e escopo de um Agente Epistêmico (Epistron)

Dado um centróide inicial $\mathbf{a_{fi}}$ pertencente ao espaço $\mathbf{A^n}$, representando o foco de um certo epistron ε^{i}, sua representação interna (a rede neuronal de retro-propagação da Figura 4.7) gera um vetor $\mathbf{c_{fi}}$. A escolha do próximo vetor $\mathbf{a_i}$ pertencente a $\mathbf{A^n}$ é realizada através do sorteio probabilístico de um novo ponto neste espaço n-dimensional $\mathbf{A^n}$ dentro do limite máximo ρ que é o raio da bola que representa o escopo do agente ε_i. O caráter estocástico desse sorteio representa o ruído natural em sistemas complexos, levando à auto-organização da rede epistêmica.

Por exemplo, diferentes pesquisadores, conforme seus vetores iniciais $\mathbf{a_f}$ (focos) e seus ρ (escopos), transitam de forma aleatória, limitada pelo espaço n-dimensional $\mathbf{A^n}$, representando diferentes experimentos e interesses. Pesquisadores com maior amplitude de observação possuem um ρ maior enquanto que pesquisadores com menor amplitude de observação possuem seus escopos mais limitados, concentrando-se em determinadas "áreas de interesse" mais restritas (Figura 4.10).

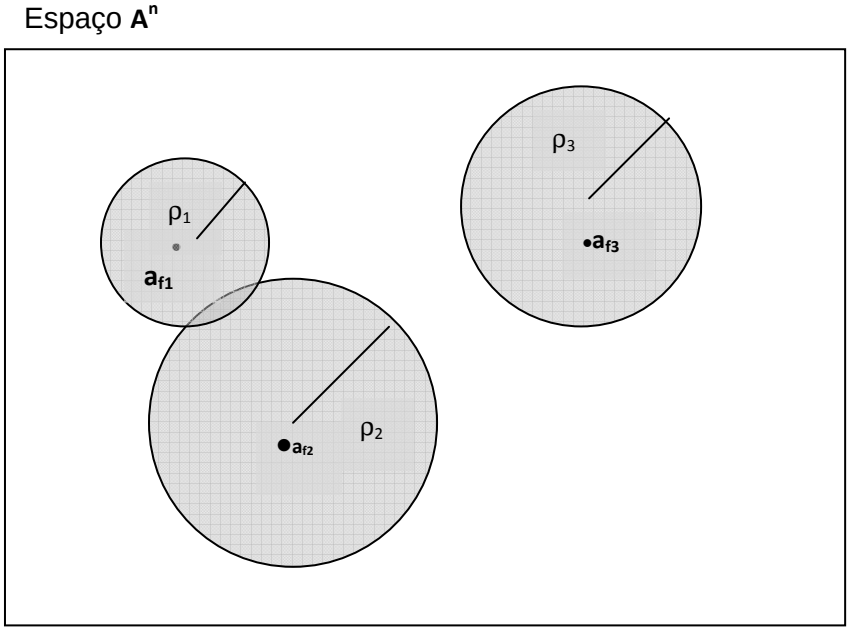


Figura 4.10 – Diagrama esquemático de 3 agentes epistêmicos (epistrons) com diferentes focos e escopos.

Da mesma forma, no sistema imunológico, estes escopos representam a capacidade de generalização no reconhecimento de antígenos, aliado ao processo de mutação do linfócito, que confere ao sistema sua criatividade na produção de novos anticorpos.

Como outro exemplo, podemos considerar os interesses de diferentes usuários de uma rede social virtual, como o Orkut. Digamos que três amigos, **A**dolfo, **B**ernardo e **C**arlos, possuem perfil no Orkut. Adolfo tem um interesse

específico (foco) em futebol, com um escopo estreito ligado apenas aos times do Rio de Janeiro. Já Bernardo, apesar de gostar de futebol também (mesmo foco), tem seu escopo mais amplo, considerando também os times de todo o Brasil. Carlos tem outros interesses, e aprecia música e literatura.

Abaixo, podemos ver a representação esquemática dos três amigos, com focos e escopos distintos:

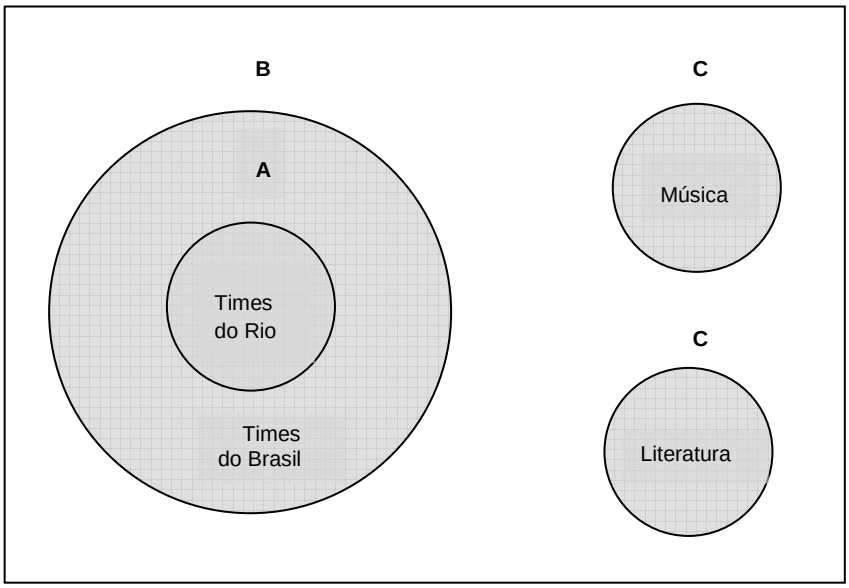


Figura 4.11 – Diagrama de Interesses de três usuários do Orkut

4.4.2. Dinâmica coletiva da rede epistêmica

A todo instante, os epistrons geram (conforme uma distribuição de frequência f_g) e comunicam (conforme uma distribuição de frequência f_c) pares epistêmicos. Cada epistron recebe a comunicação de outro epistron e atualiza suas crenças (aprendizagem individual) em função da importância percebida por ele do epistron que comunica (W_{ji}) e pela distância entre os vetores **c** (consequentes) para um mesmo vetor **a** (antecedente) comunicado (δ_{ij}).

A Figura 4.11 apresenta um diagrama esquemático da dinâmica coletiva do modelo:

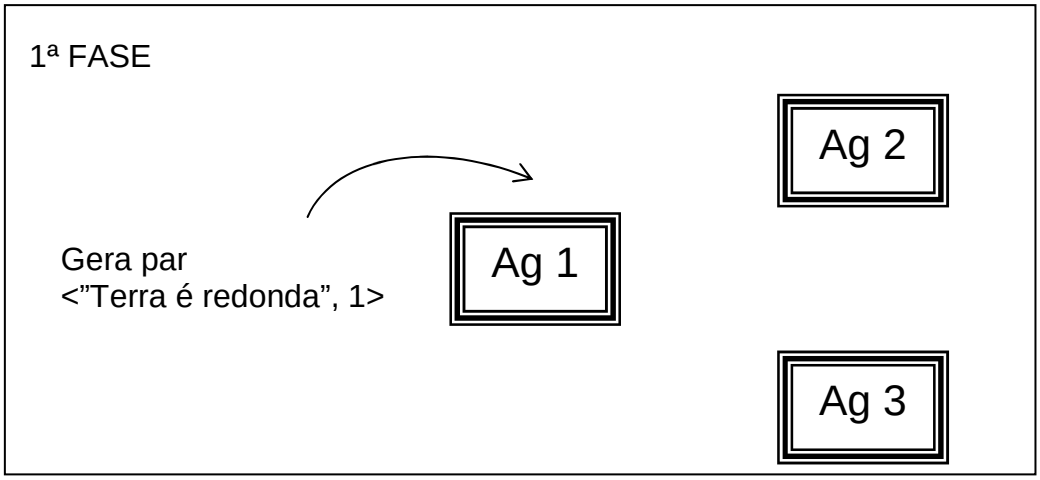


Figura 4.12(a) – Fase de Geração

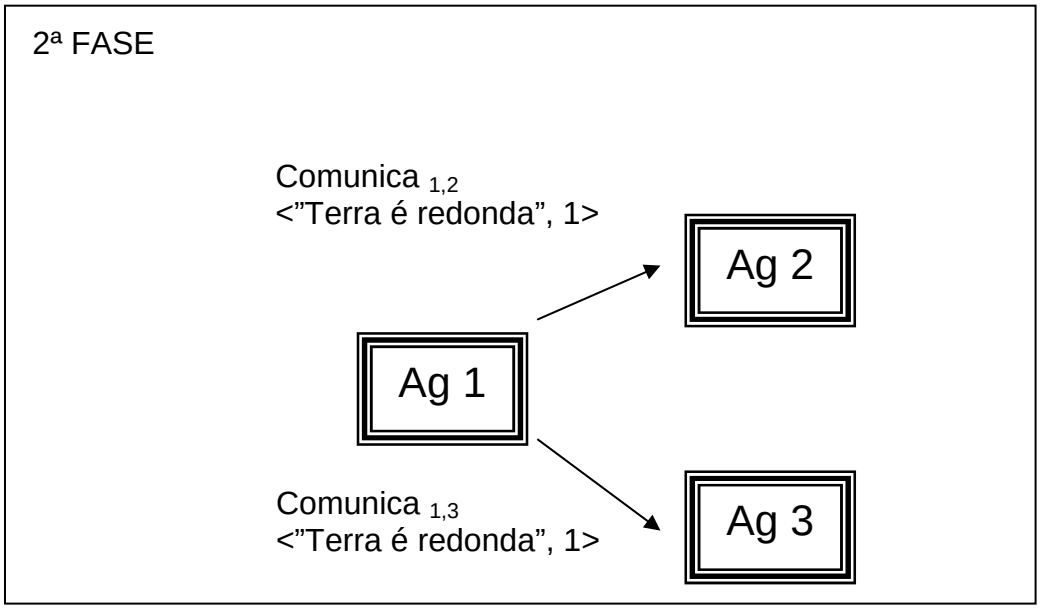


Figura 4.12(b) – Fase de Comunicação

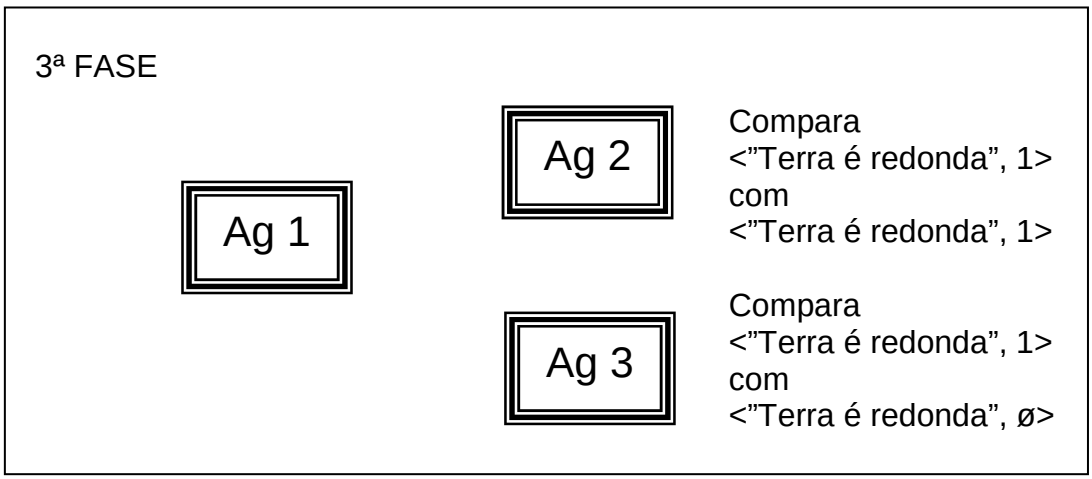


Figura 4.12(c) – Fase de Comparação

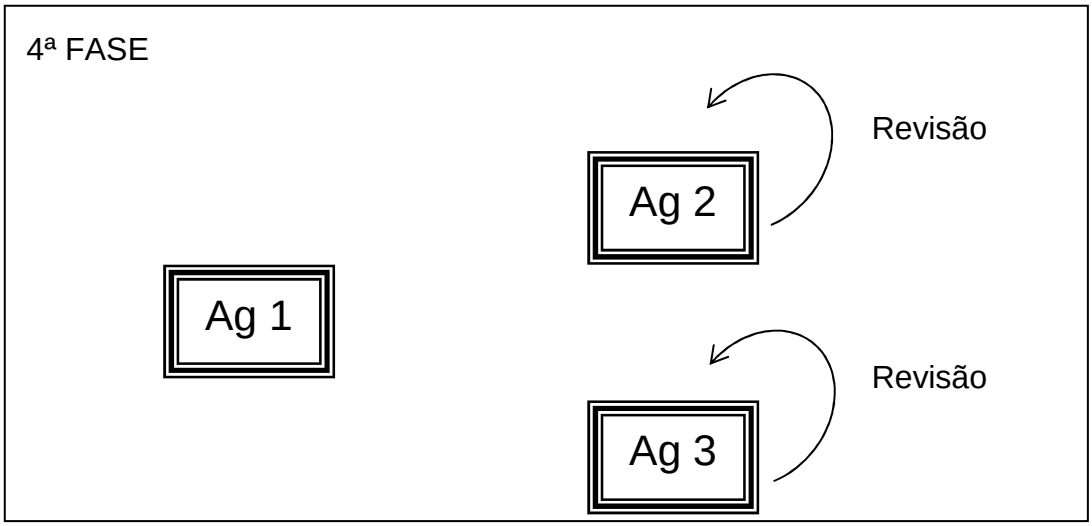


Figura 4.12(d) – Fase de Revisão (Aprendizado Individual)

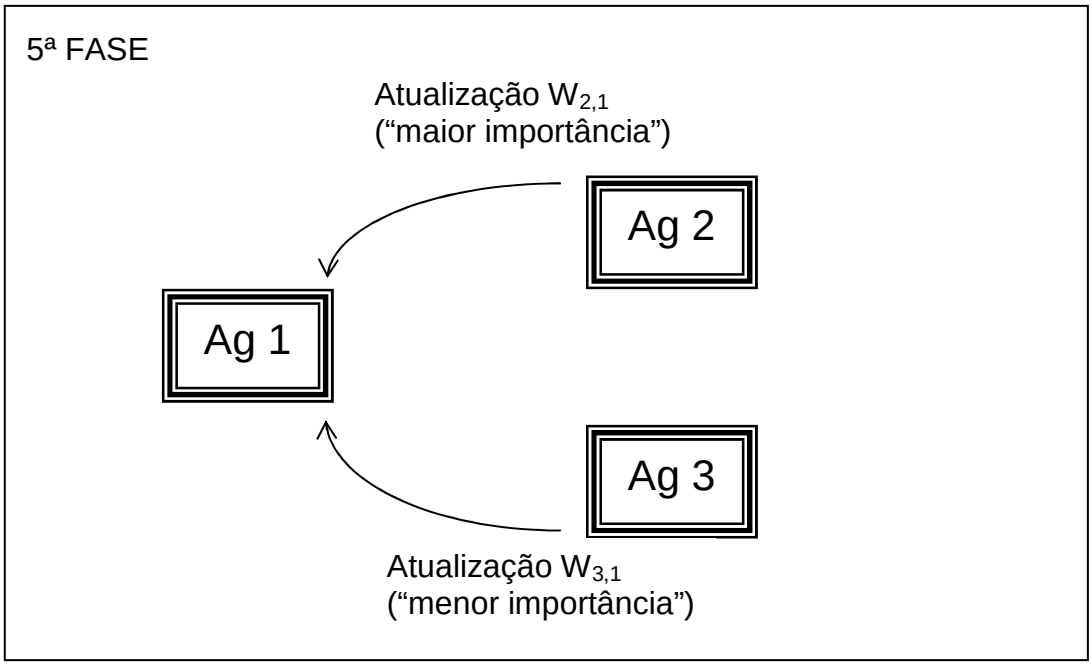


Figura 4.12(e) – Fase de Atualização dos Ws (Aprendizado Coletivo)

A aceitação da comunicação de um par epistêmico $<a_i, c_i>$ de um epistron ε_i para um epistron ε_j qualquer e a revisão das crenças do epistron ε_j é modelada por um processo de aprendizado supervisionado, no qual são aplicadas L etapas de aprendizagem à rede neuronal de aprendizado por retro-propagação que representa as crenças de ε_j. Estas etapas são proporcionais à importância de ε_i para ε_j (W_{ji}) e também à distância entre os vetores $\mathbf{c_i}$ e $\mathbf{c_j}$ ($\delta_{ij} = |\mathbf{c_i} - \mathbf{c_j}|$) para $\mathbf{a_i} = \mathbf{a_j}$ ($L \alpha \delta_{ij} . W_{ji}$). Quanto maior a importância do agente que comunica, maior a intensidade do treinamento.

Em seguida, a conexão W_{ji} é atualizada pela Regra de Hebb, isto é:

$$\Delta W_{ji} = \eta . \frac{1}{1 + \delta_{ij}} . W_{ji} \qquad \text{(Equação 4.1)}$$

sendo η uma constante (taxa de aprendizagem) para toda a rede.

A regra de Hebb formaliza, desta forma, o princípio da rede espistêmica: quanto menor for a distância entre os consequentes (ou seja, quanto mais próximas forem as crenças), maior é a variação W_{ji}. Valores maiores de Ws sinalizam, portanto, a proximidade entre as crenças dos agentes.

Um aprimoramento do modelo permite o surgimento e morte de agentes epistêmicos, conforme uma determinada distribuição de frequência. Isso acrescenta uma variável interessante ao modelo, permitindo a modelagem de fenômenos dinâmicos e evolutivos, como a multiplicação de anticorpos em sistemas imunológicos ou o surgimento de novos pesquisadores científicos, atraídos por grandes grupos de pesquisa. Com essa dinâmica se pode testar as hipóteses do modelo de Barabási e Albert [8], referentes ao crescimento da rede e à propriedade de *preferential attachment*, conforme discutidas anteriormente no capítulo 3.

Pode-se aumentar a frequência de comunicação (f_{ci}) de um epistron ε_i a partir do crescimento de sua *autoconfiança* (W_{ii}), agregando um novo fator à dinâmica da rede. A variação da autoconfiança de um epistron ε_i é proporcional ao somatório dos W_{ji} ($\Delta W_{ii} \alpha V . \Sigma W_{ji}$), onde $V \in [0,1]$ é uma constante que representa o grau de *volubilidade* do epistron. Quando este parâmetro é próximo de zero, a auto-confiança do epistron não depende da importância que os outros epistrons da rede lhe atribuem, enquanto que para valores próximos de um, o epistron é muito volúvel, alterando sua auto-confiança sempre em função da confiança que os outros epistrons da rede atribuem a ele.

Por fim, considera-se também no modelo que possa haver comunicações públicas, destinadas a todos os outros epistrons, e comunicações privadas, destinadas apenas a um subgrupo de epistrons da rede epistêmica. Para modelar essa propriedade, atribui-se a cada epistron ε_i um parâmetro dinâmico θ_i, denominado *limiar da comunicação*, de tal forma que somente para W_{ij} acima deste liminar as comunicações serão realizadas pelo epistron ε_i. O parâmetro θ_i pode ser sorteado, em uma primeira hipótese, junto com o par epistêmico a ser comunicado. Ou, em outra hipótese, pode variar conforme a autoconfiança do próprio epistron. Tal característica permite modelar com mais precisão as redes de pesquisadores científicos, por exemplo, nas suas interações intragrupos e intergrupos de pesquisa. No caso do sistema imunológico, podemos modelar o estímulo que um clone de anticorpos envia a outros clones, em uma reação inflamatória qualquer.

Sintetizando o modelo que foi apresentado anteriormente, um epistron ε_i é definido pelos seguintes parâmetros:

$$\varepsilon_i = \{\mathbf{B}_i, \rho_i, f_{gi}, f_{ci}, \theta_i, \mathbf{a_{fi}}, V_i\},$$

Onde

$\mathbf{B}_i$ = matriz de crenças;

ρ_i = escopo de ε_i;

f_{gi} = distribuição de frequência de geração de ε_i;

f_{ci} = distribuição de frequência de comunicação de ε_i;

θ_i = limiar de comunicação de ε_i;

$\mathbf{a_{fi}}$ = centróide inicial (foco) de ε_i;

V_i = volubilidade de ε_i.

Uma rede epistêmica com N epistrons é definida, portanto, como

$$R(N) = \{\varepsilon, \mathbf{W}, \eta\}$$

Onde

ε = conjunto de N epistrons da rede

W = matriz de importâncias entre epistrons da rede

η= taxa de aprendizagem da rede

O algoritmo que modela a dinâmica da rede epistêmica, para N agentes epistêmicos (epistrons), é descrito a seguir, de uma forma simples, lembrando que é desta simplicidade que se espera a emergência de fenômenos complexos:

ALGORITMO DA REDE EPISTÊMICA

Inicializar os parâmetros de todos os epistrons da rede;
Repetir para M etapas:
    Escolher aleatoriamente um epistron ε_i;
    Segundo a distribuição de frequência f_{gi}, faça:
        Gerar par epistêmico $<\mathbf{a_i}, \mathbf{c_i}>$ a partir do foco $\mathbf{a_{fi}}$;
        Segundo a distribuição de frequência f_{ci} faça:
            Para todos os epistrons ε_j da rede cujos W_{ij} são maiores do que θ_i faça:
                Comunicar par epistêmico $<\mathbf{a_i}, \mathbf{c_i}>$ a ε_j;
                Executar etapa de aprendizagem em ε_j;
                Atualizar W_{ji}.
FIM

Observe-se que o algoritmo acima está representando a rotina de funcionamento da rede epistêmica em caráter macroscópico, para implementação em uma máquina serial. É possível imaginar a implementação também em uma máquina paralela, na qual vários epistrons estariam gerando e comunicando pares epistêmicos simultaneamente. A paralelização da rede epistêmica permitiria a utilização de um número muito maior de epistrons, levando-nos a emergências possivelmente mais interessantes e realistas em relação às redes sociais, ao sistema imune e a outros exemplos de aplicações. Este é um dos propósitos futuros de trabalho, como será discutido no Capítulo 7.

No próximo capítulo apresenta-se a implementação do algoritmo da rede epistêmica em um ambiente de simulação que permite a experimentação computacional do modelo proposto, de modo a verificar a sua validade e aplicabilidade.

5. Simulação

A construção de um ambiente de simulação computacional justifica-se no caso de modelagem de sistemas de grande complexidade, dada a dificuldade de tratamento puramente matemático. Com a simulação, se pode-se observar ***in silico*** a emergência de fenômenos coletivos, a partir das propriedades individuais dos agentes.

Na implementação do algoritmo de redes epistêmicas foi utilizada a linguagem de programação Java, com o ambiente de desenvolvimento IDE Eclipse. A escolha da linguagem deveu-se à liberdade de utilização de um software livre, além dos recursos de concorrência disponíveis.

As principais classes e objetos da implementação são: *RedeEpistemica*, *AgenteEpistemico* e *ParEpistemico*. No Anexo A são apresentados os diagramas de sequência e de classes e objetos utilizados na implementação do simulador.

Na mitologia hindu, a realidade é descrita por uma metáfora, como uma infinita rede de joias, interligadas por fios. Em cada joia pode-se ver o reflexo de todas as demais. Trata-se da Rede de Indra, uma linda imagem para entender como tudo se interliga, em uma realidade holística e conectada [39].

Batizado em referência a essa metáfora, e para realização da simulação, foi construído o simulador de redes epistêmicas IndraNet, já em sua versão 1.2.

O limite máximo de agentes que o simulador permite, desconsiderando a capacidade da máquina, é igual à capacidade de um inteiro. Na prática, executando-se o IndraNet com mais do que mil agentes, o sistema torna-se lento em um computador com 2GB de memória RAM e um processador de dois núcleos com o *clock* de 2GHz. Entretanto, o algoritmo é passível de ser transformado para ser executado em múltiplas máquinas.

Na Figura 5.1, podemos ver as opções de configuração dos parâmetros dos agentes epistêmicos

.

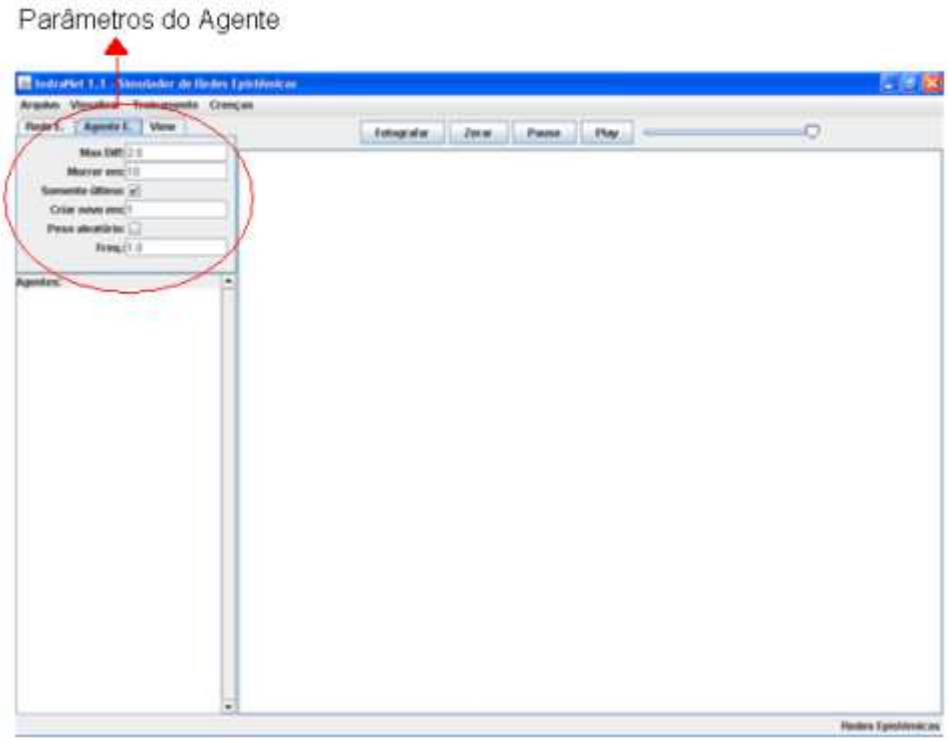


Figura 5.1 – Tela de configuração dos parâmetros dos agentes epistêmicos no simulador IndraNet

São definidos, para os agentes epistêmicos da rede, os seguintes parâmetros configuráveis:

(a) Máxima diferença (*Max Diff*);
(b) Taxa de Mortalidade;
(c) Taxa de Nascimento;
(d) Peso Aleatório;
(e) Frequência .

A máxima diferença (*Max Diff*) para um agente define a distância máxima entre os consequentes de dois agentes, para que ocorra o treinamento interno (aprendizagem individual). É o caso em que, ao receber um par epistêmico comunicado por um agente, o outro agente calcula a diferença entre os

conseqüentes (δ_{ij}), e se a diferença for maior do que o parâmetro de máxima diferença, não haverá a etapa de treinamento. Tal critério serve para criar visualmente o “afastamento relativo” entre agentes da rede epistêmica.

A taxa de mortalidade (“*Morrem em*”) configura o número máximo de etapas da vida de um agente, após o qual ele desaparece da rede epistêmica.

A taxa de nascimento (“*Criar Novo em*”) define em quantas etapas é criado um novo agente epistêmico, permitindo o aprendizado topológico na rede epistêmica.

O *peso aleatório* define se os pesos (W_{ij}s) atribuídos aos agentes epistêmicos são inicialmente distribuídos de forma homogênea ou aleatória.

A *frequência* define a velocidade de geração e comunicação dos agentes da rede.

Na Figura 5.2, podemos ver as opções de configuração dos parâmetros da rede epistêmica.

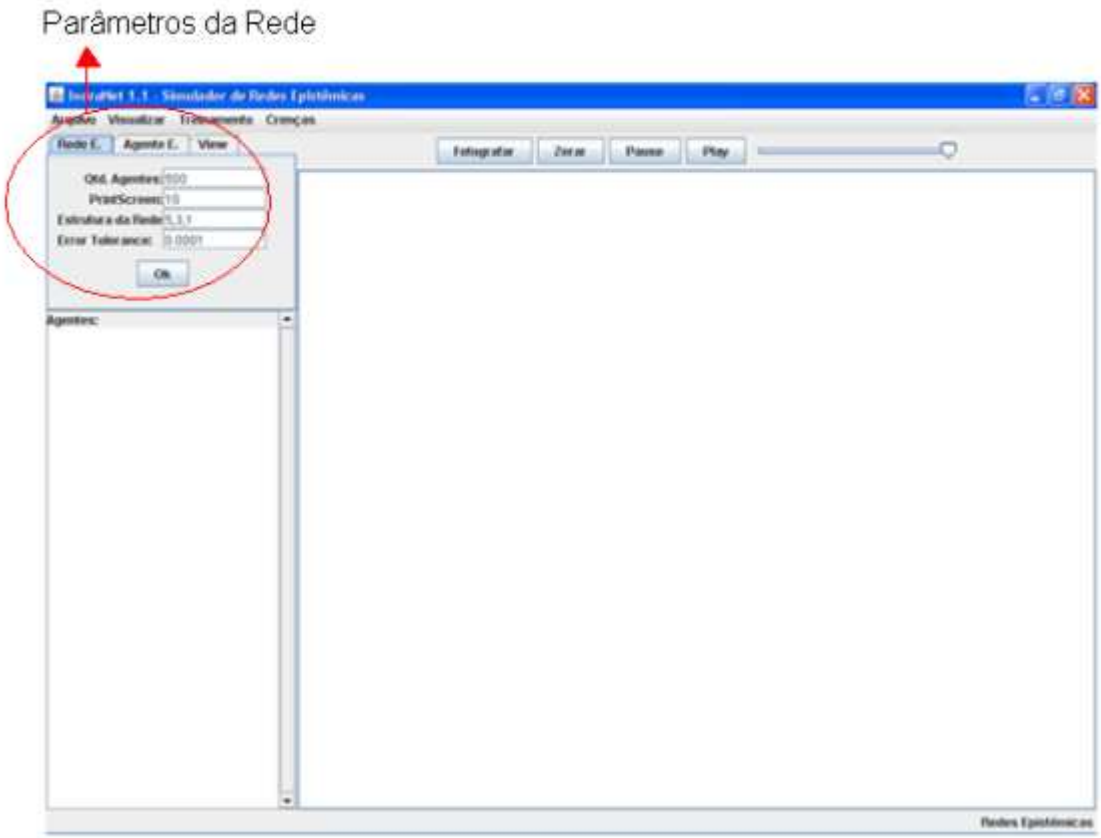


Figura 5.2 – Tela de configuração dos parâmetros da rede epistêmica
no simulador IndraNet

No nível da rede epistêmica, foram definidos os seguintes parâmetros configuráveis:

(a) Quantidade de agentes;
(b) Impressão de telas;
(c) Estrutura da rede neuronal de cada agente;
(d) Tolerância do erro;
(e) Limiar de comunicação.

A *quantidade de agentes* define o total de agentes epistêmicos da rede, inicialmente.

A funcionalidade de impressão de telas (*print screen*) permite guardar, após N etapas, o estado da rede epistêmica em um arquivo de imagens.

A *estrutura da rede* define, para os agentes, a estrutura das suas redes neuronais (camada de entrada, camadas intermediárias e camada de saída). Na implementação atual, todos os agentes epistêmicos possuem a mesma estrutura interna.

A *tolerância do erro* é um parâmetro do algoritmo de aprendizado por retro-propagação (*back-propagation*) dos agentes epistêmicos.

O *limiar de comunicação* (θ_i) é o parâmetro do modelo que define a partir de que valor de importância os agentes comunicam as suas crenças aos seus pares.

Na Figura 5.3, podemos ver as opções de configuração dos parâmetros da visualização da rede epistêmica.

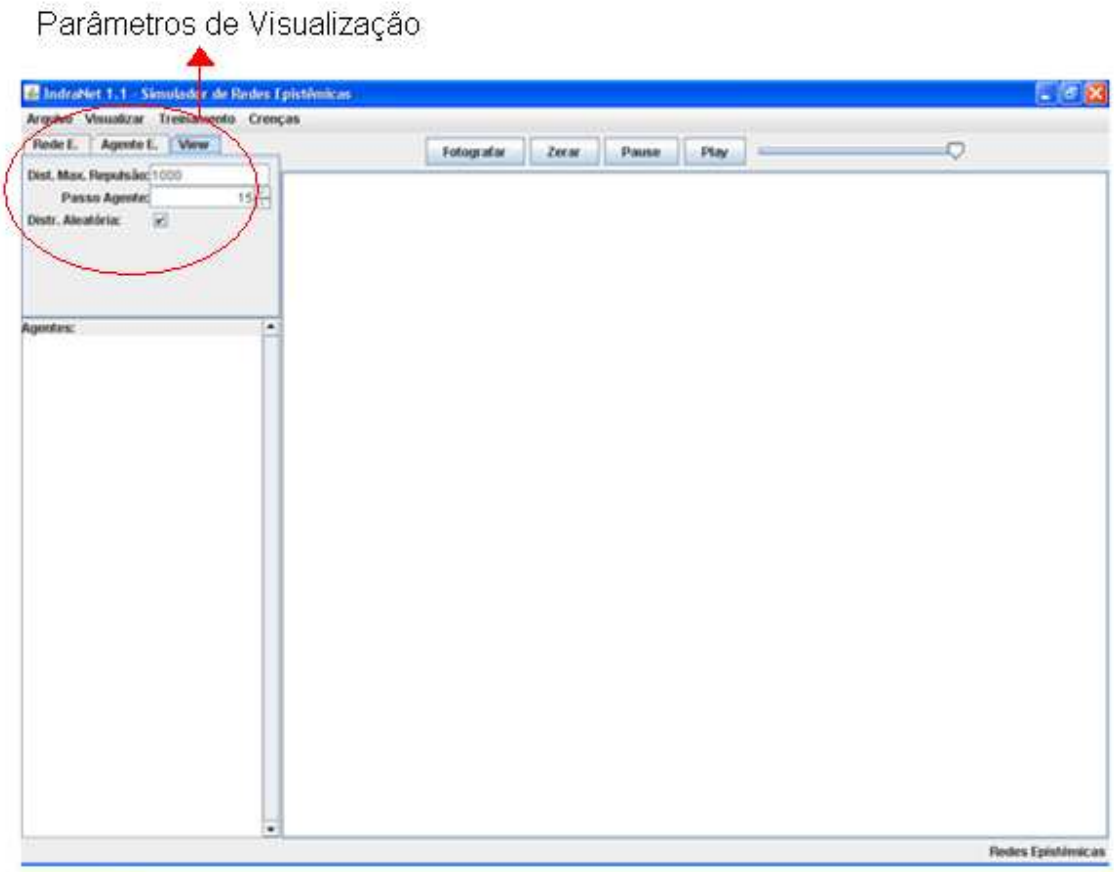


Figura 5.3 – Tela de configuração dos parâmetros de visualização da rede epistêmica no simulador IndraNet

No nível da configuração da visualização, foram definidos os seguintes parâmetros:

(a) Distância Máxima de Repulsão;
(b) Passo do Agente;
(c) Distribuição Aleatória.

A *Distância Máxima de Repulsão* representa um parâmetro do distanciamento físico dos agentes na tela da simulação, de forma a evitar que saiam do campo de visão.

O *Passo do Agente* representa a medida física da aproximação dos agentes na tela, permitindo melhor visualização.

O parâmetro booleano *Distribuição Aleatória* (sim ou não) permite escolher se os agentes serão distribuídos na tela inicialmente de forma aleatória ou se serão dispostos de maneira regular, por escolha do responsável pelo experimento.

A implementação permite, ainda, as seguintes funcionalidades:

(a) Criar e Salvar Arquivos;
(b) Visualizar Pesos dos Agentes Epistêmicos;
(c) Aproximação e Afastamento (*zoom in*, *zoom out*) nas Telas;
(d) Treinamento Prévio dos Agentes;
(e) Identificação de Crenças por Cores;
(f) Visualizar Gráfico de Potência da Rede.

Com a implementação do simulador IndraNet, se pode simular um conjunto amplo de situações do mundo real, bastando que sejam escolhidos os parâmetros e as condições iniciais dos agentes epistêmicos e da rede epistêmica. Trata-se de uma ferramenta simples, que utiliza uma interface gráfica que permite a observação visual da propagação de crenças e da proximidade entre agentes.

Decidiu-se pela utilização de cores para identificação das crenças por questões de espaço da interface gráfica. Caso fossem marcados os agentes com letras ou números não seria possível ter uma boa leitura em simulações com 50 ou mais agentes simultaneamente.

A implementação poderá ser futuramente aperfeiçoada, criando-se uma versão do algoritmo que seja executado de forma paralela em várias máquinas. Isso daria muito mais capacidade ao que hoje se restringe a apenas uma máquina. Cada agente seria independente dos demais, funcionando dentro de um processo separado e poderia se comunicar com outros agentes em outros computadores por TCP/IP.

No Anexo A encontra-se a documentação técnica completa do simulador IndraNet.

6. Estudos de Caso

No presente capítulo pretende-se apresentar alguns casos de simulação da rede epistêmica, ilustrativos do potencial do modelo para representação da evolução de redes sociais do mundo real.

6.1. Auto-organização de uma rede epistêmica

No primeiro caso, procura-se ilustrar a situação da pura auto-organização de uma rede epistêmica, na qual um número inicial de 100 agentes, com crenças iniciais aleatoriamente definidas, e que possuem a mesma estrutura de rede interna e a mesma frequência de comunicação, interagem e formam um *cluster*. Trata-se de situação potencialmente descrita por Atlan [2], na qual um sistema biológico, em um processo de aprendizagem não-dirigida, transforma o "caos inicial" em uma ordem (estrutura).

Vê-se, na Figura 6.1, a situação inicial da rede epistêmica, na qual todos os agentes possuem inicialmente os mesmos pesos e distribuem-se aleatoriamente pela tela da simulação (Exemplo 1.(a)).

Na Figura 6.2, após 13 etapas, alguns agentes já comunicaram suas crenças, e forma-se uma hierarquia por ordem de importância, na qual alguns agentes começam a receber mais Ws do que outros.

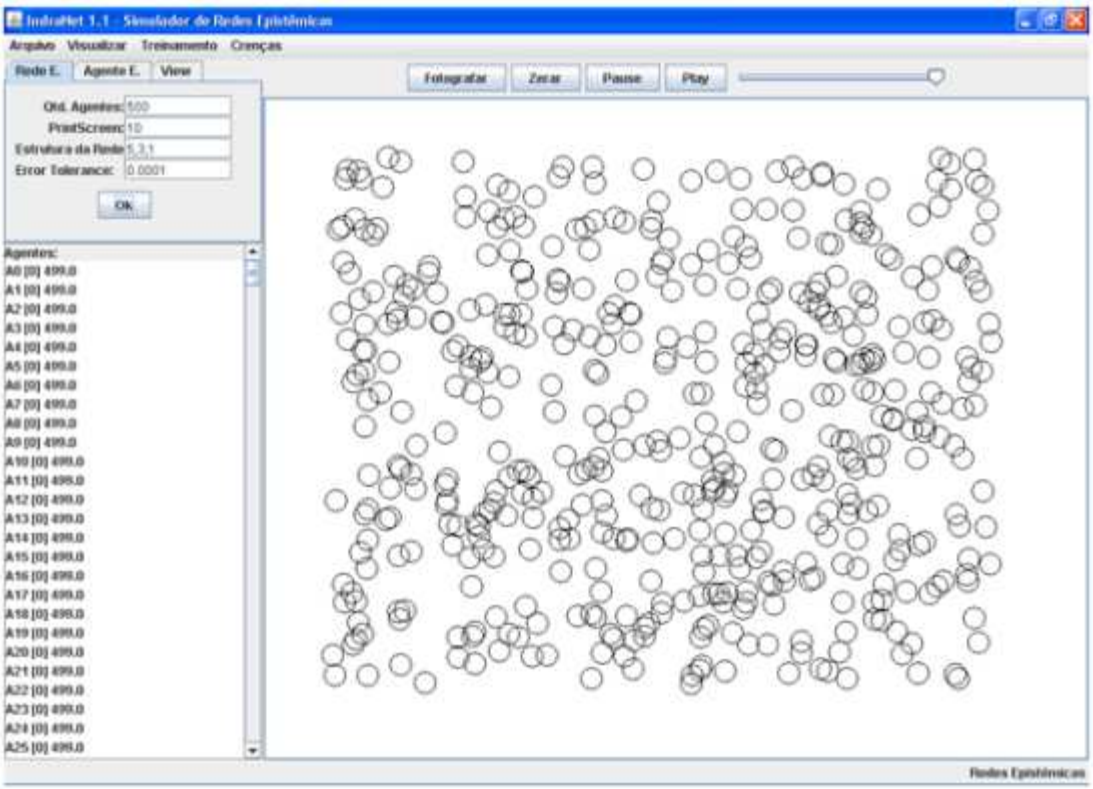


Figura 6.1 – Situação inicial do Exemplo 1.(a)

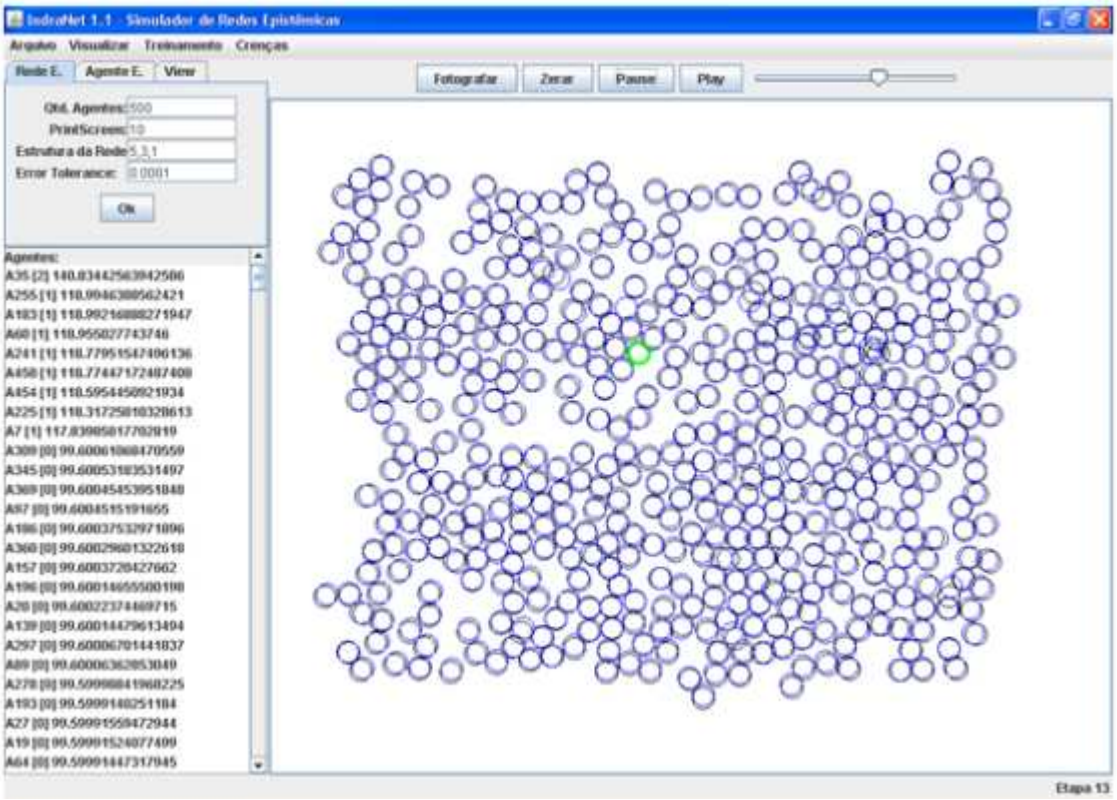


Figura 6.2 – Exemplo 1.(a) após 13 etapas

Pode-se perceber que, ao executar-se a simulação, os agentes se aproximam gradualmente, formando crenças cada vez mais comuns.

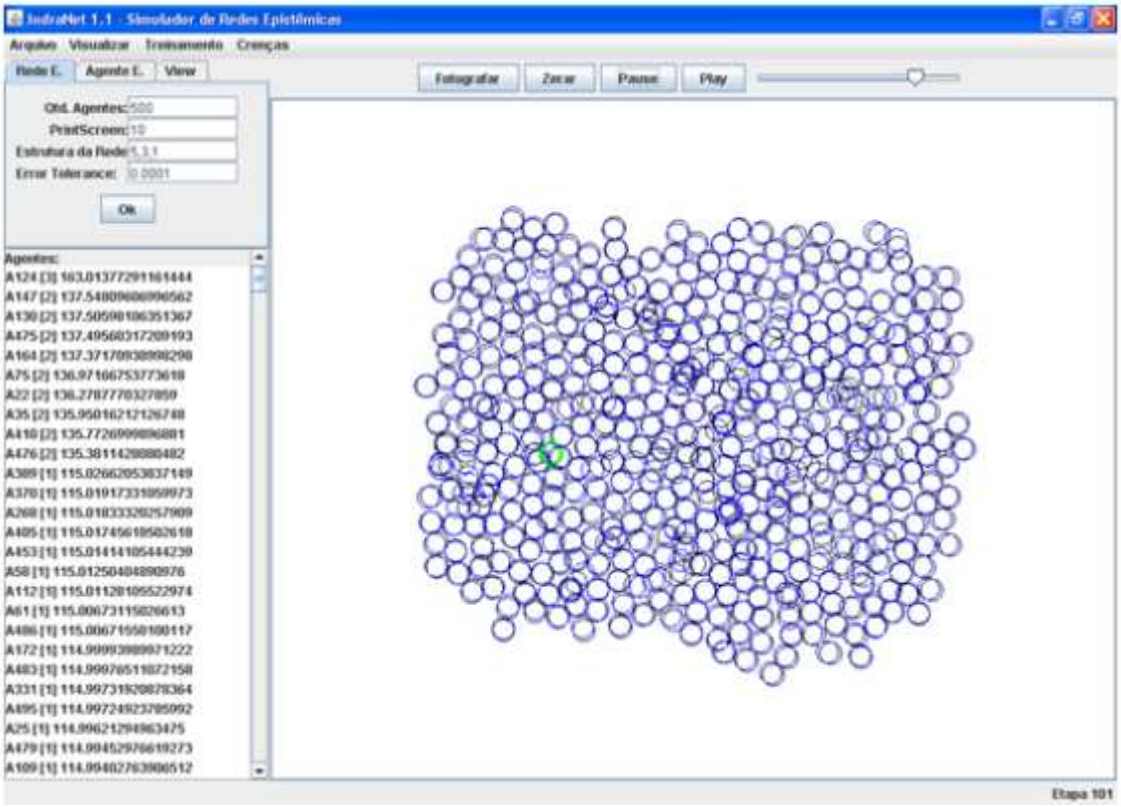


Figura 6.3 – Exemplo 1.(a) após 101 etapas

Com 101 etapas do algoritmo, os agentes estão cada vez mais próximos (Figura 6.3), até que, ao chegar a 1.000 etapas, as crenças praticamente se tornaram todas comuns, conforme a sequência de figuras:

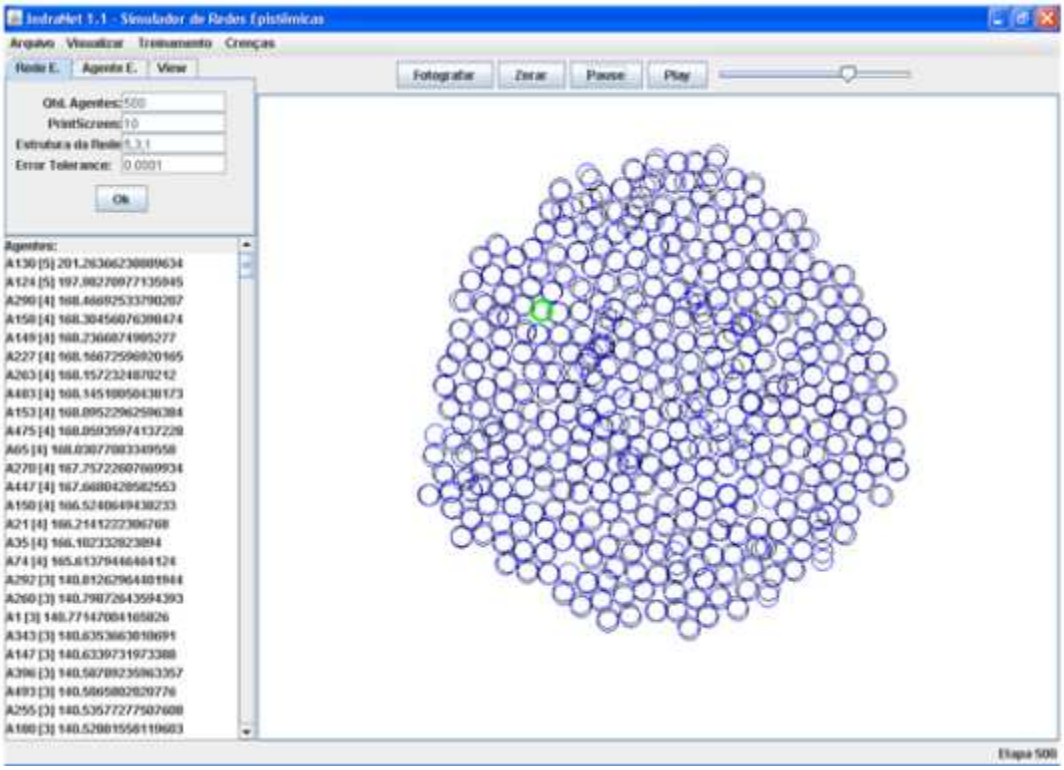


Figura 6.4 – Exemplo 1.(a) após 500 etapas

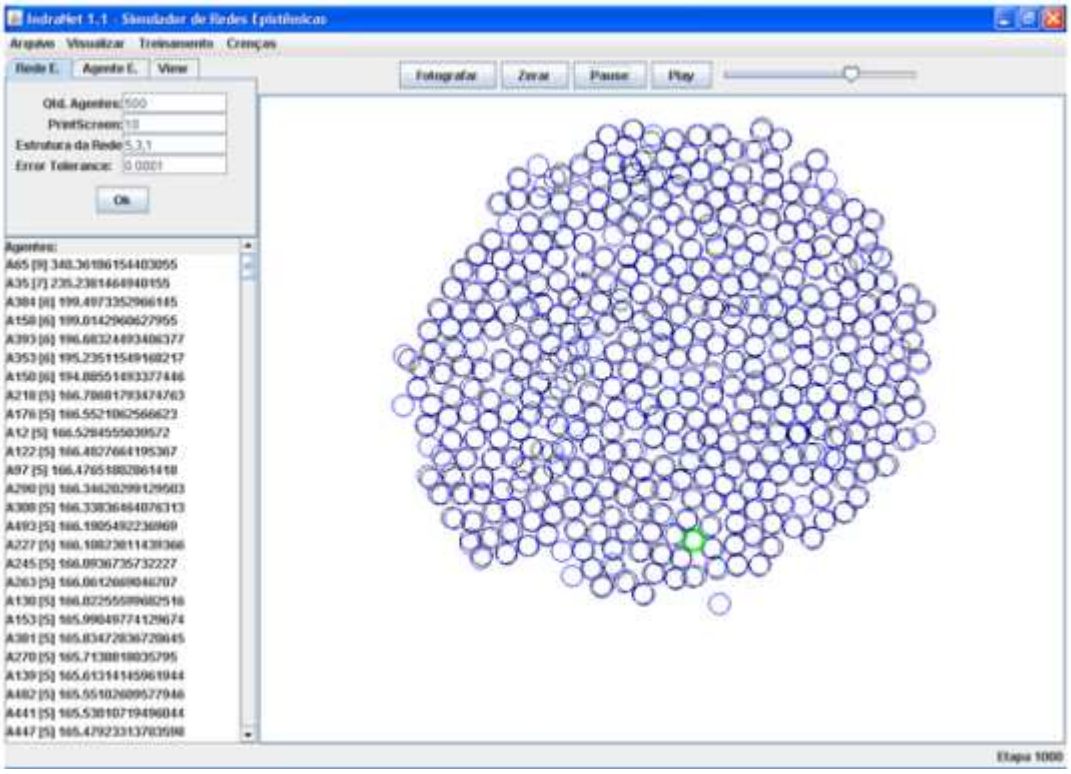


Figura 6.5 – Exemplo 1.(a) após 1.000 etapas

Note-se que um agente (A65) possui um “peso” maior do que os outros agentes, e em torno dele forma-se o *cluster*.

Na sequência de figuras, vê-se um outro exemplo (Exemplo 1.(b)) com os mesmos parâmetros, em outra simulação, agora com a visualização do gráfico que mostra a distribuição dos pesos dos agentes por faixas de “importância”, medida pelos seus somatórios de Ws aferentes. Um resultado importante das redes epistêmicas é a presença, a partir de um número grande de etapas, de um gráfico de distribuição de pesos com a formação de uma “Lei de Potência”. Isto significa que as redes epistêmicas evoluem naturalmente para uma estrutura *scale-free*, na qual alguns poucos agentes possuem muita importância, e muitos outros agentes possuem pouca importância.

Na Figura 6.10, vê-se o gráfico de distribuição após 5 simulações de 3.500 etapas. O eixo X representa as faixas de valores dos somatórios de Ws que apontam para um agente, dando uma medida da importância do agente. O eixo Y representa a quantidade de agentes que possuem tal faixa de valores de W. Intuitivamente, podemos dizer que as redes epistêmicas possuem a propriedade de que poucos agentes recebem muita importância, e muitos agentes recebem pouca importância (“*the rich gets richer”*), conforme prevista na Equação 3.2, e presente em várias outras situações do mundo real [8].

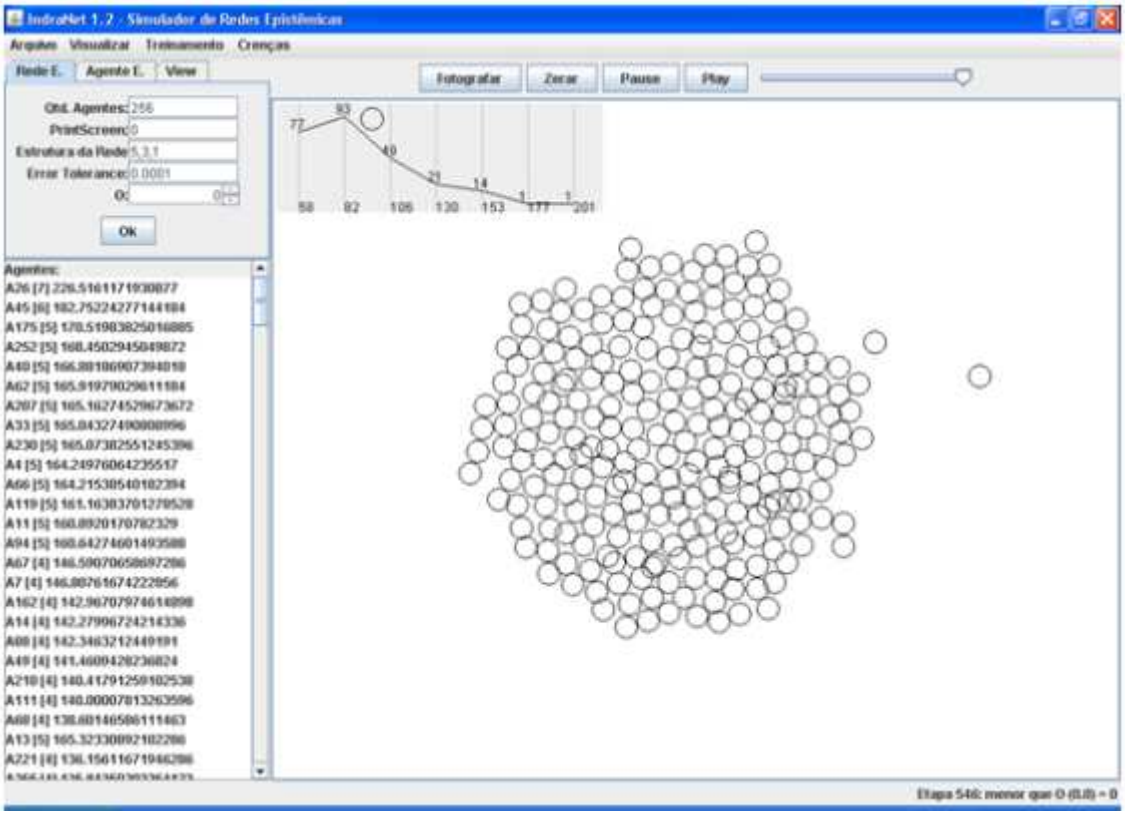


Figura 6.6 – Tela inicial do Exemplo 1.(b)

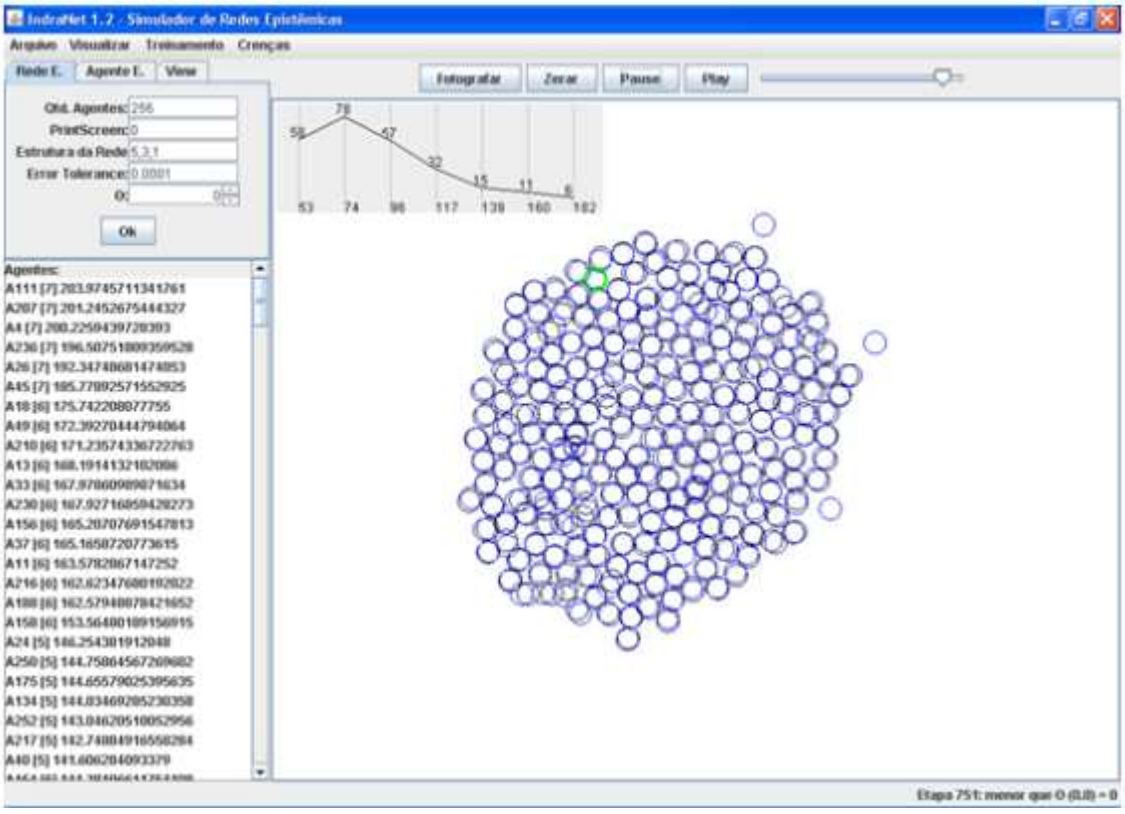


Figura 6.7 – Exemplo 1.(b) após 751 etapas

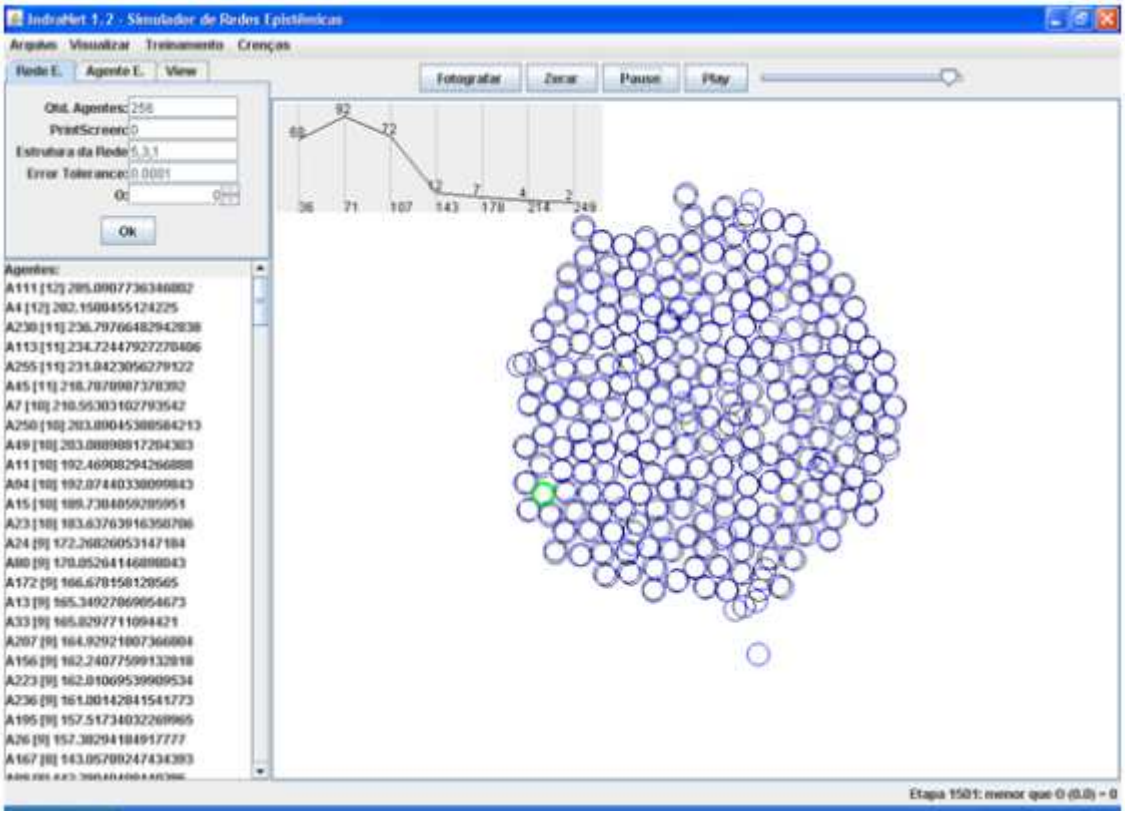


Figura 6.8 – Exemplo 1.(b) após 1501 etapas

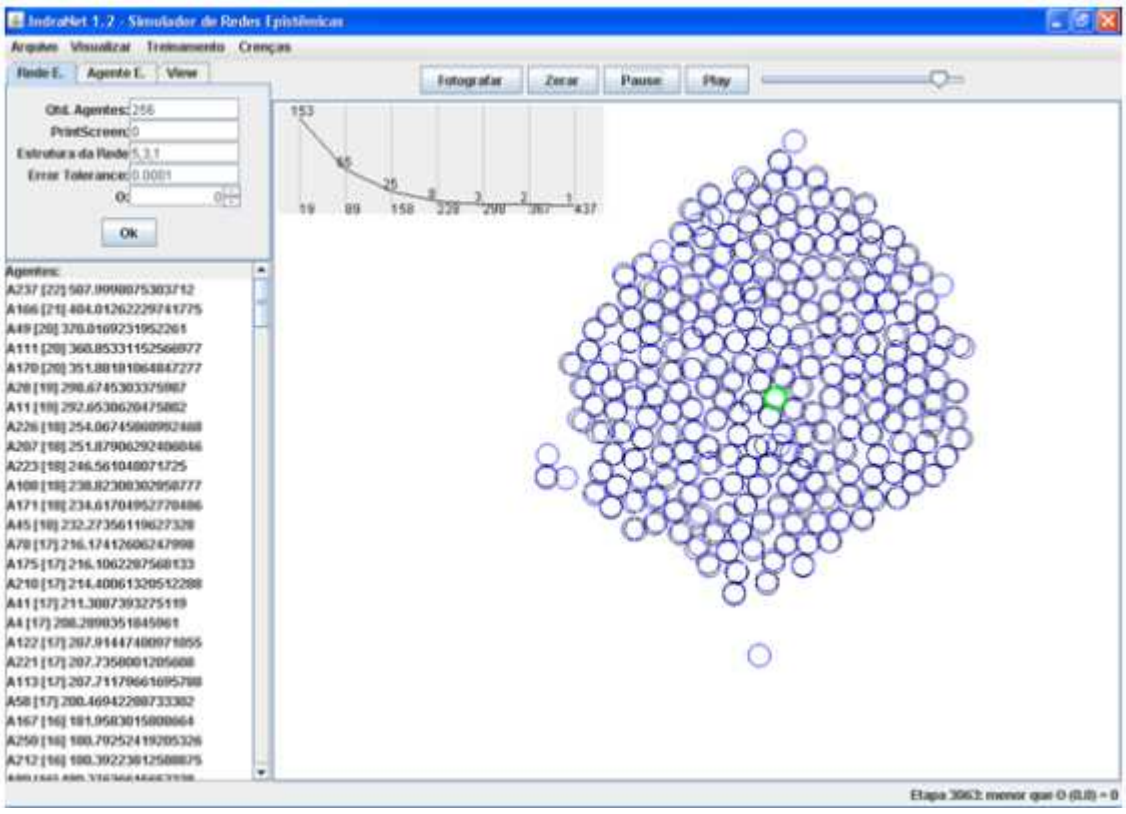


Figura 6.9 – Exemplo 1.(b) após 3.063 etapas (Lei de Potência)

		QUANTIDADE DE AGENTES					
FAIXA DE PESOS		Simulação 1	Simulação 2	Simulação 3	Simulação 4	Simulação 5	Média
1	De 15 a 99	159	158	159	158	158	158,4
2	De 99.1 a 182.2	71	73	72	72	72	72
3	De 182.3 a 266.2	17	16	16	17	17	16,6
4	De 266.3 a 350.4	3	3	3	3	3	3
5	De 350.5 a 434	5	5	5	5	5	5
6	De 434.1 a 518	0	0	0	0	0	0
7	De 518.1 a 601.8	1	1	1	1	1	1

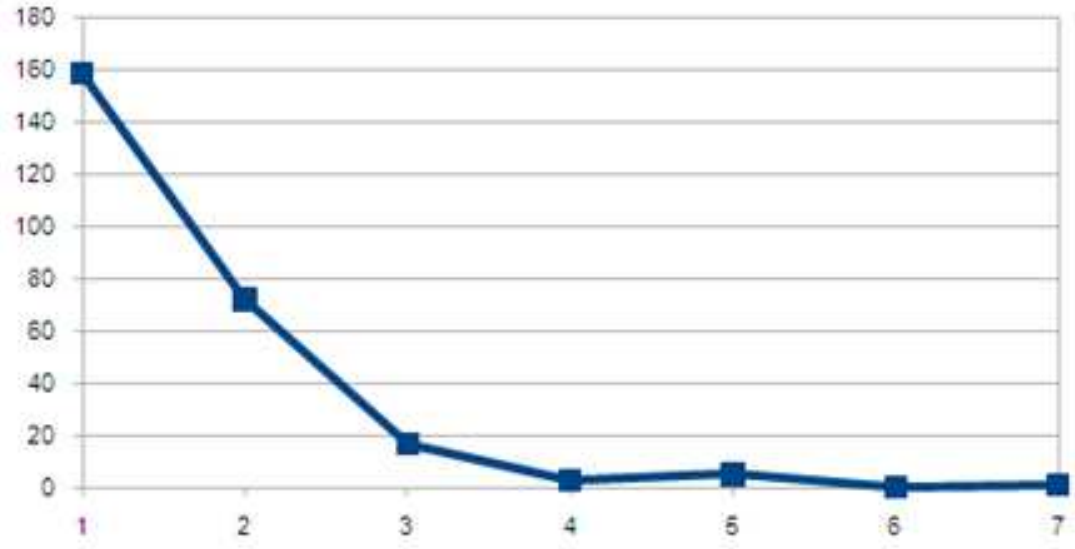


Figura 6.10 – Gráfico de Distribuição Média dos Somatórios de Pesos dos Agentes

6.2. Propagação de Crenças

No Exemplo 2, apresenta-se uma situação na qual se observa como uma crença de um agente epistêmico é propagada para outros agentes de uma mesma rede epistêmica.

No caso em tela, considera-se um agente com frequência de geração e comunicação fixas, um treinamento inicial do mesmo agente em uma crença pré-definida (que é caracterizada pela cor vermelha) e os outros agentes inicialmente gerados com crenças aleatórias e frequência de comunicação zero.

Na sequência de etapas (Figuras 6.11 a 6.17), os agentes gradualmente se aproximam do agente transmissor da crença, que ganha valores crescentes do somatório de Ws apontados para ele. A partir de um certo número de etapas, os agentes vão sendo “contagiados” pela crença, formando ao final um *cluster* totalmente vermelho. Trata-se de um caso emblemático de “propagação de ideias” (contágio social), em uma forma que podemos chamar de canônica: apenas um agente transmite a sua crença para um conjunto de outros agentes da rede epistêmica.

Tal exemplo poderá ser posteriormente ampliado para estudo da propagação de crenças em situações com freqüência de comunicação variável, ou com comunicações seletivas entre agentes (nem todos os agentes recebem a comunicação ao mesmo tempo).

Outro efeito que poderá ser futuramente estudado é o da importância da autoconfiança dos agentes na propagação de crenças, variando-se a volubilidade dos agentes e estudando-se o impacto na propagação das crenças na rede epistêmica.

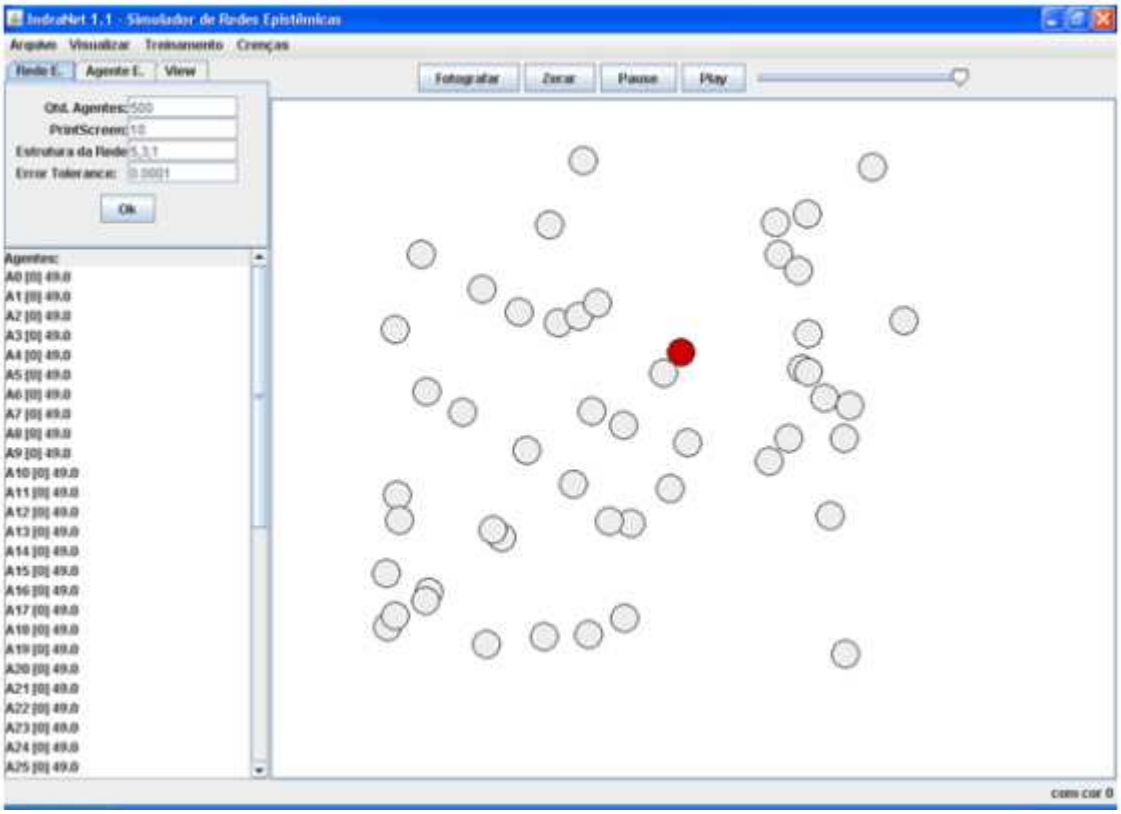


Figura 6.11 – Tela inicial do Exemplo 2

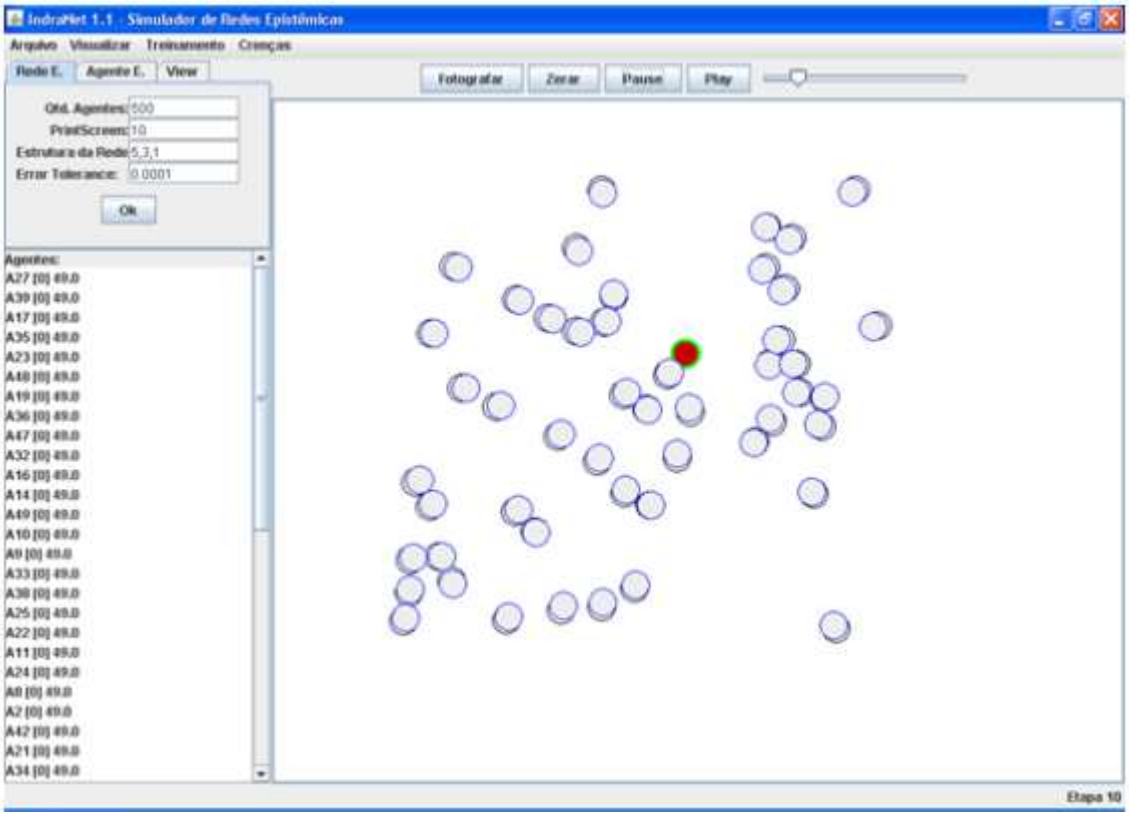


Figura 6.12 – Exemplo 2 após 10 etapas

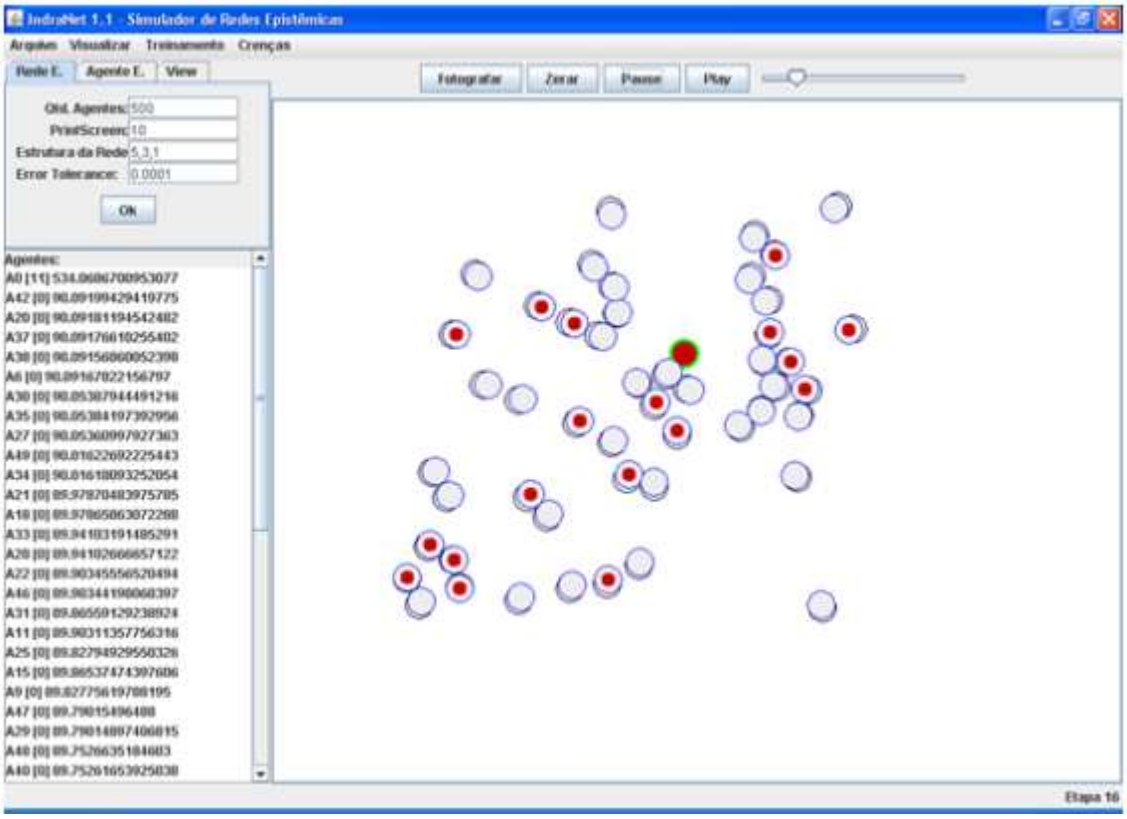


Figura 6.13 – Exemplo 2 após 16 etapas

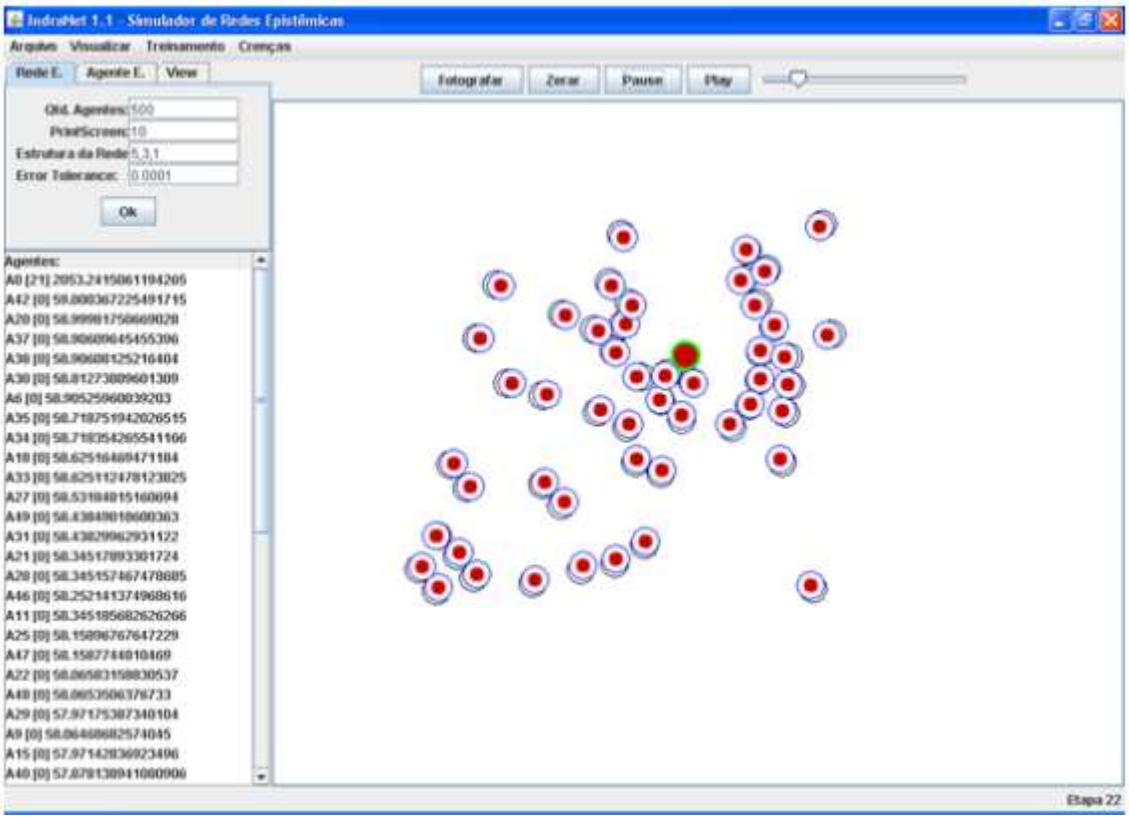


Figura 6.14 – Exemplo 2 após 22 etapas

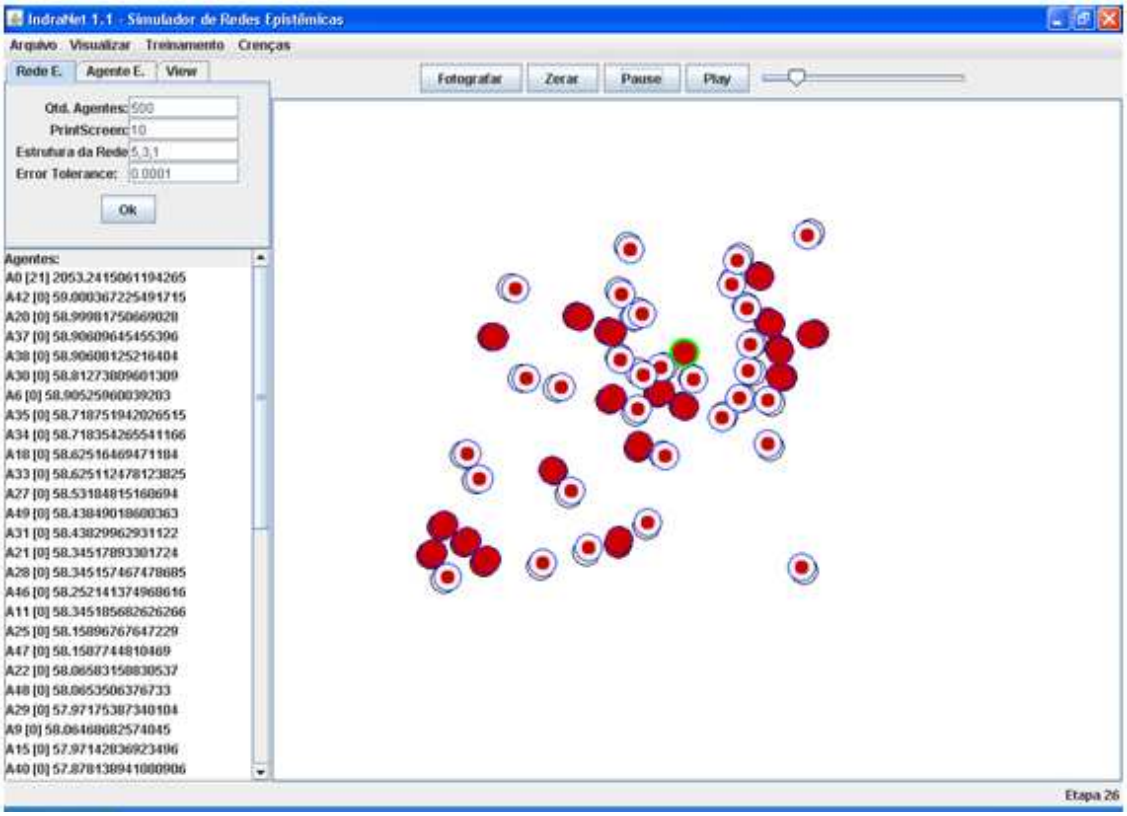


Figura 6.15 – Exemplo 2 após 26 etapas

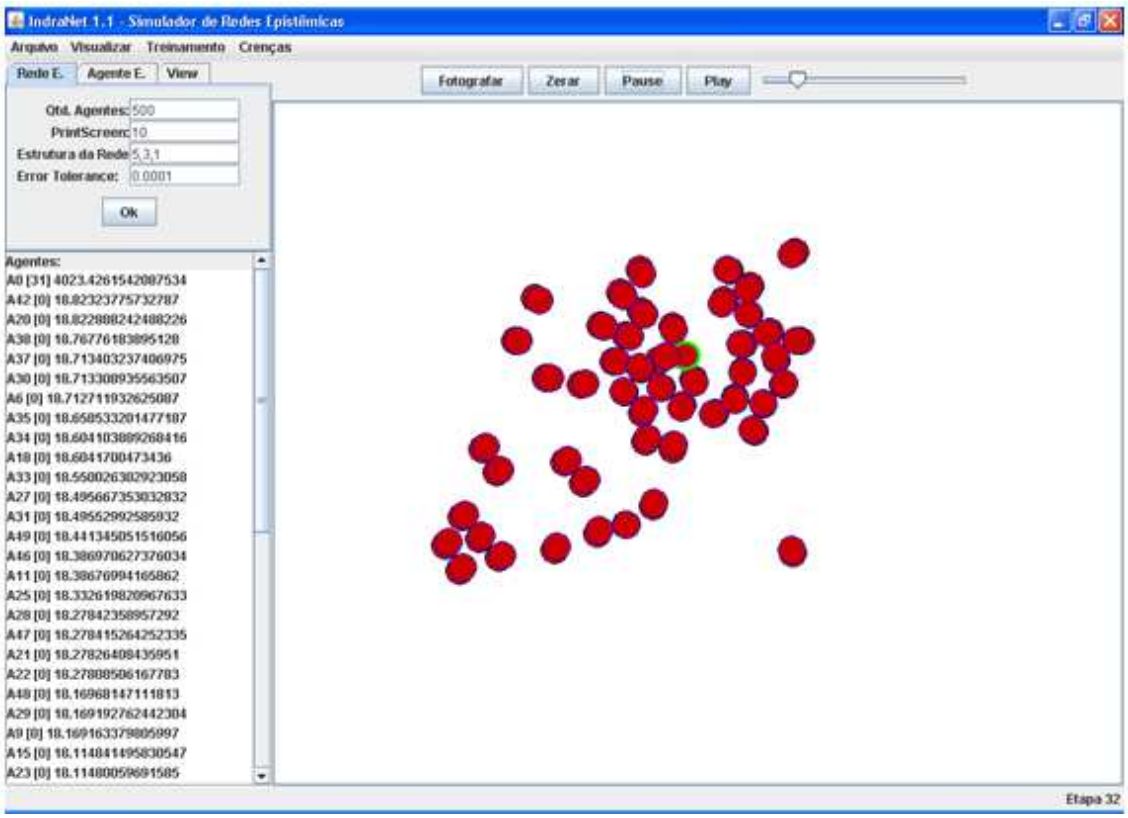


Figura 6.16 – Exemplo 2 após 32 etapas

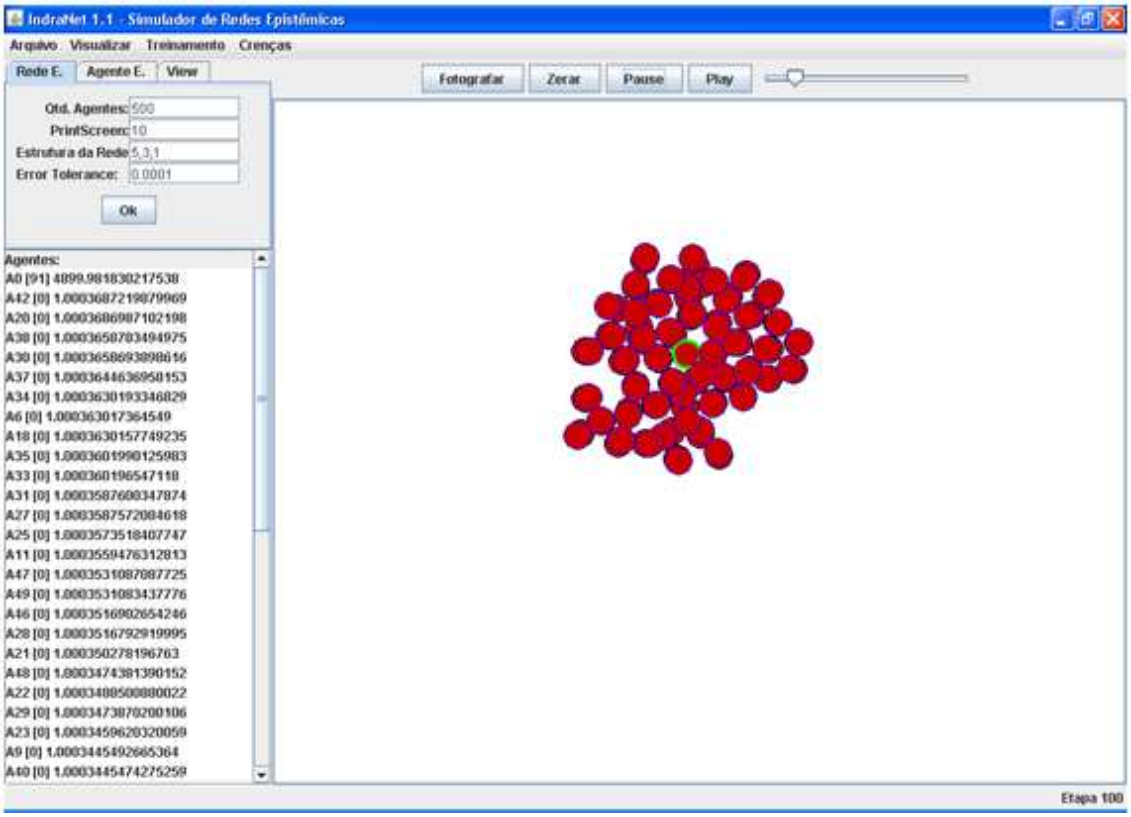


Figura 6.17 – Exemplo 2 após 100 etapas

6.3. Formação de *Clusters* Concorrentes

Em diversas situações do mundo real, determinados grupos se formam em torno de crenças comuns, e, em um determinado instante, diferentes grupos convivem com suas crenças distintas. Podemos usar a metáfora de Kuhn [42], considerando-os como paradigmas concorrentes.

No Exemplo 3, procura-se mostrar como as redes epistêmicas podem ser usadas para descrever tais situações. Inicialmente, dois agentes são treinados com crenças divergentes (mesmo antecedente, porém consequentes opostos), e com a mesma frequência de comunicação. Todos os outros agentes têm frequência de comunicação zero, e os pesos são distribuídos homogeneamente entre os agentes (não há, inicialmente, agentes mais ou menos importantes). Dá-se aos dois agentes cores distintas, para facilitar a visualização do experimento. Por exemplo, consideremos duas crenças opostas:

Crença Verde = <”A Terra é o Centro do Sistema Solar”, V>
Crença Vermelha = <”A Terra é o Centro do Sistema Solar”, F>

Vê-se, na sequência de telas do Exemplo 3, que após um número de etapas os agentes da rede se aproximam de um ou de outro agente transmissor, conforme a proximidade de suas crenças, até a formação final de dois *clusters* (paradigmas) distintos.

Podemos inferir, pelo exemplo, que a formação de *clusters* nas redes epistêmicas é função da diferença de frequências de comunicação e dos pesos inicialmente atribuídos aos agentes. Agentes com maior frequência de comunicação atraem mais seguidores, e agentes com maior peso também.

As figuras de 6.18 a 6.28 apresentam o resultado da simulação (Exemplo 3). Futuramente, o modelo poderá ser tratado com variações nas frequências iniciais de comunicação e na distribuição inicial dos pesos, para avaliação das condições que aceleram a formação de *clusters* vencedores (paradigmas).

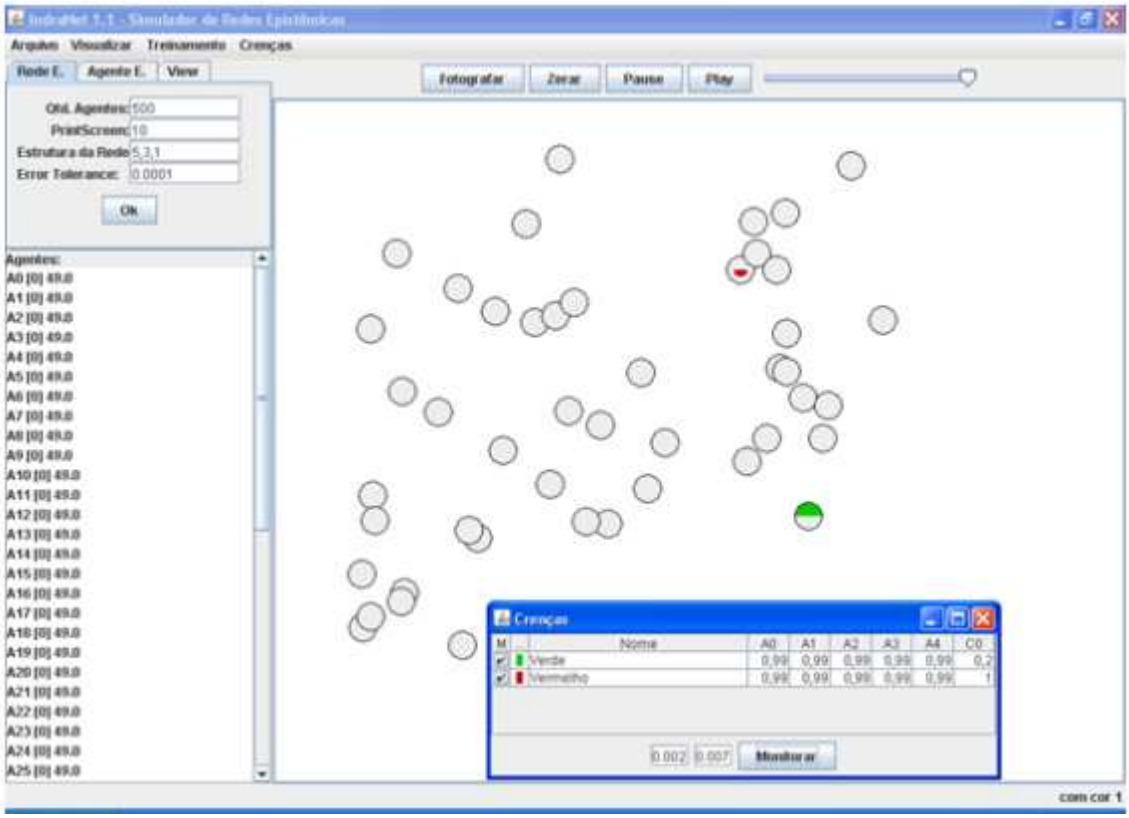


Figura 6.18 – Crenças iniciais dos dois agentes no Exemplo 3

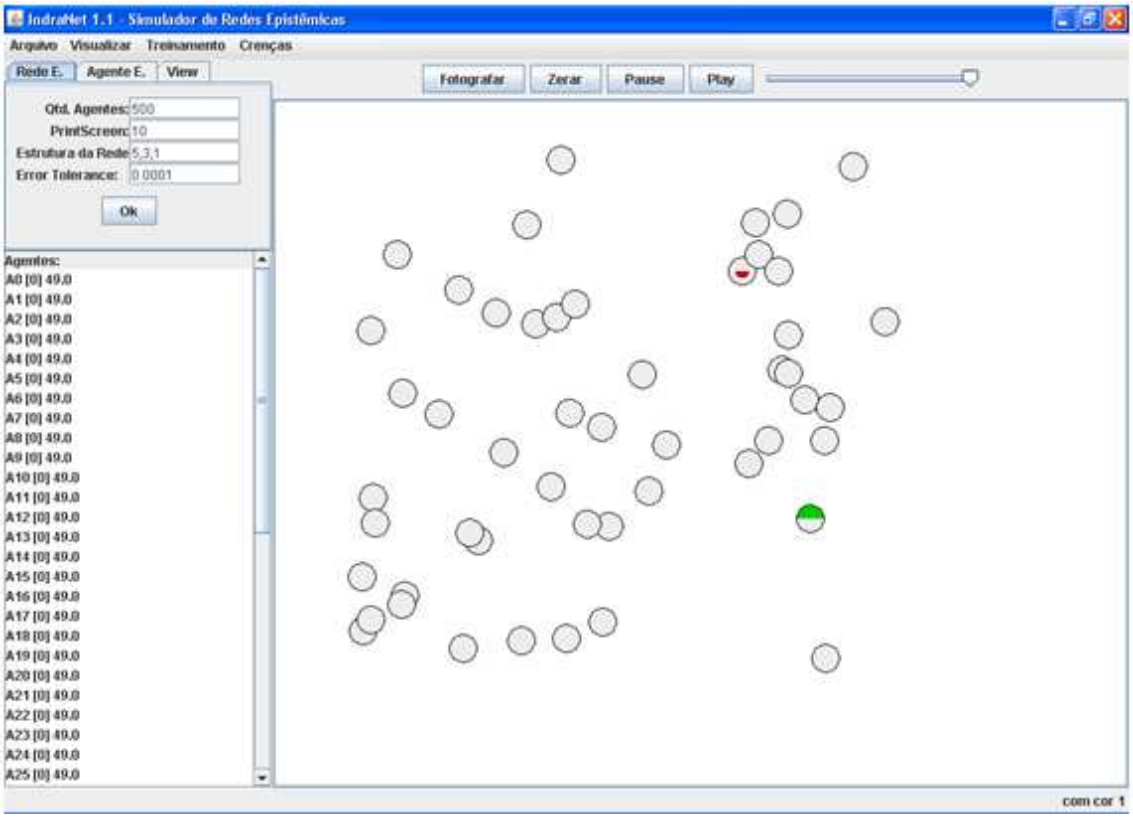


Figura 6.19 – Tela de início do Exemplo 3

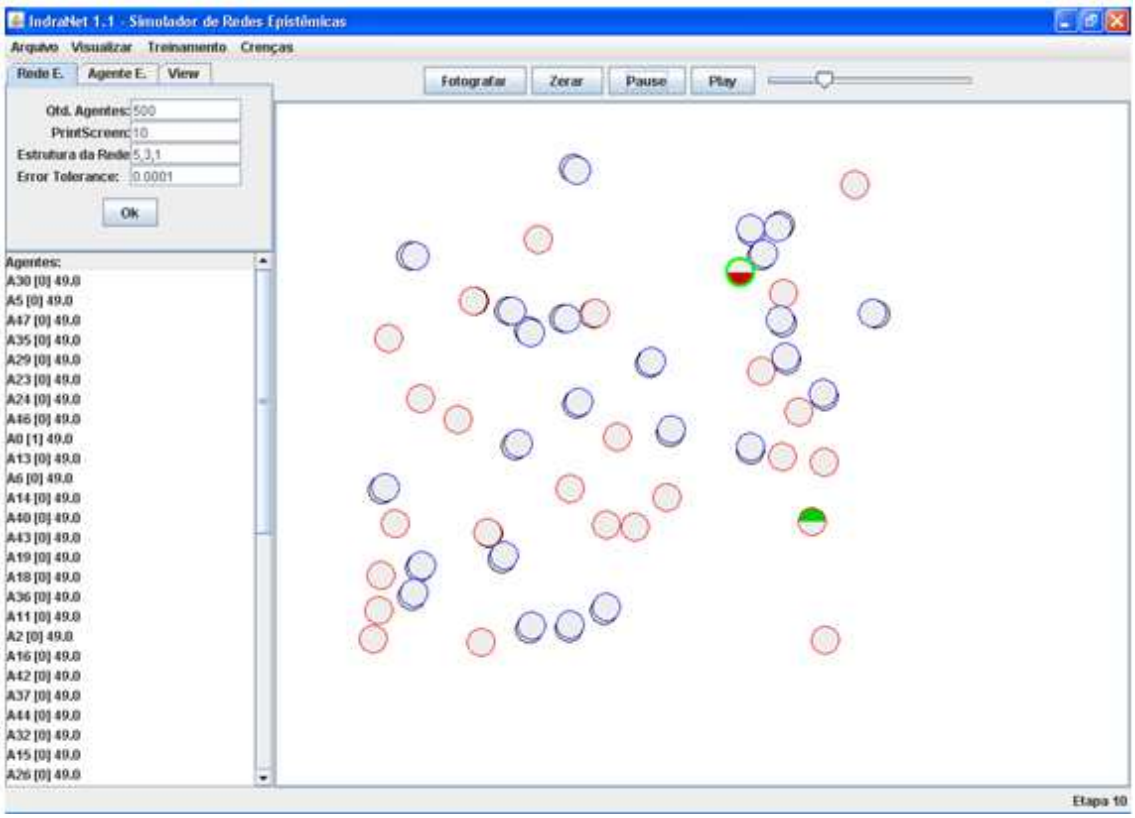


Figura 6.20 – Exemplo 3 após 10 etapas

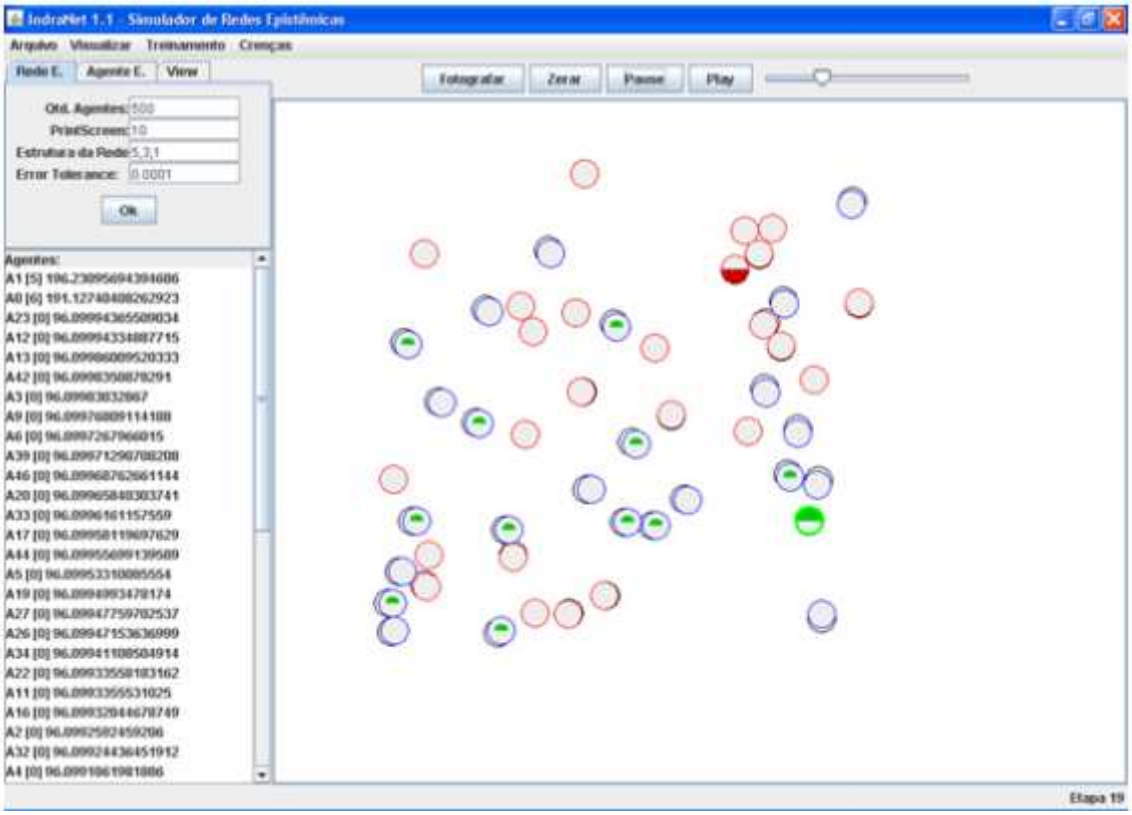


Figura 6.21 – Exemplo 3 após 19 etapas

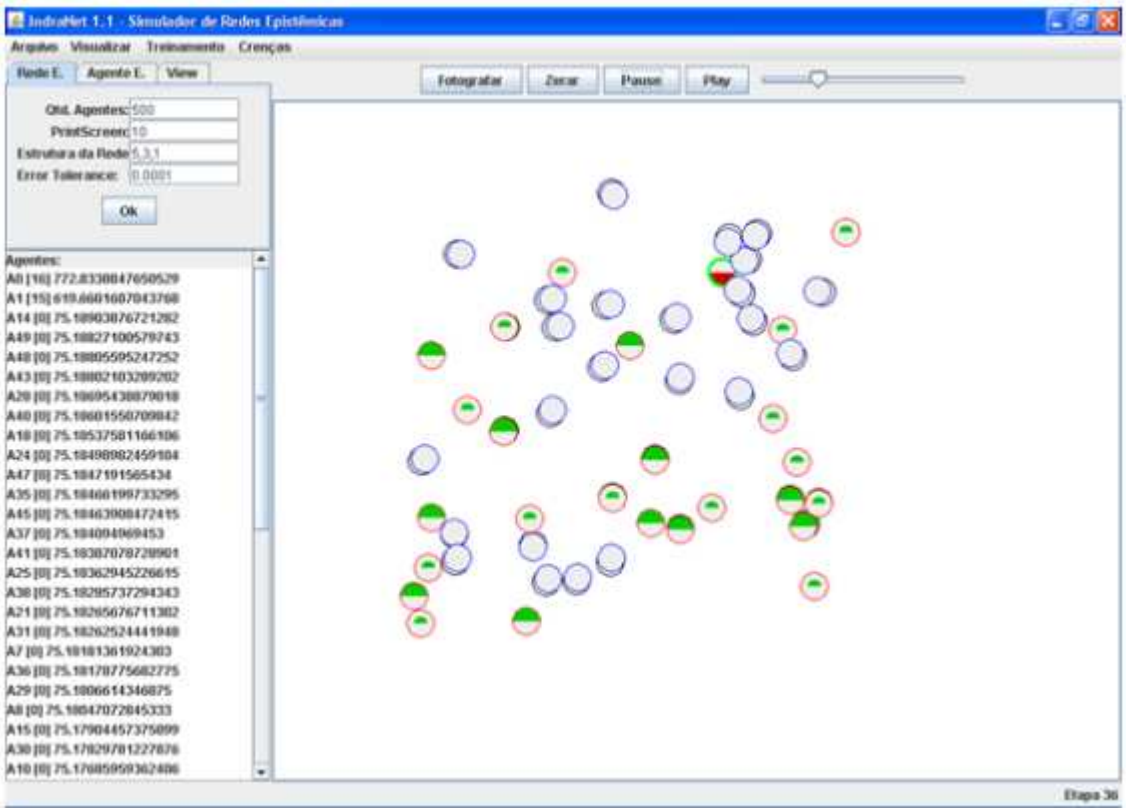


Figura 6.22 – Exemplo 3 após 36 etapas

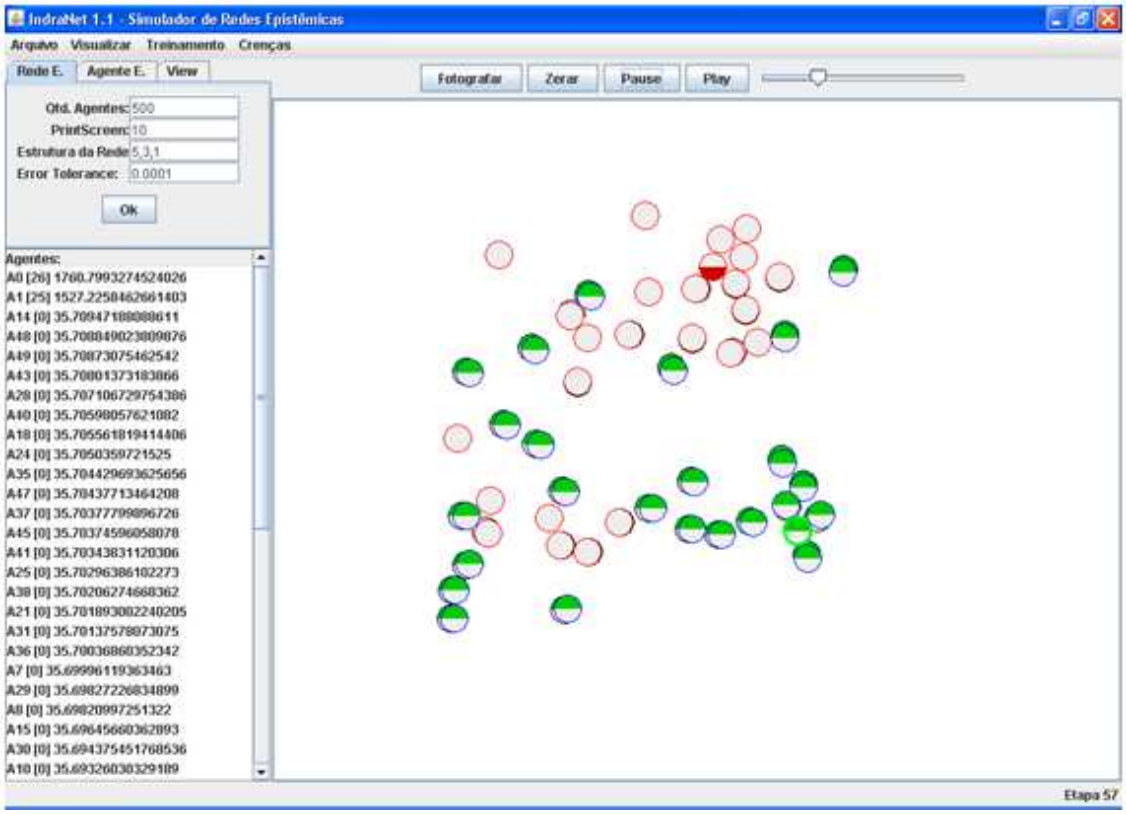


Figura 6.23 – Exemplo 3 após 57 etapas

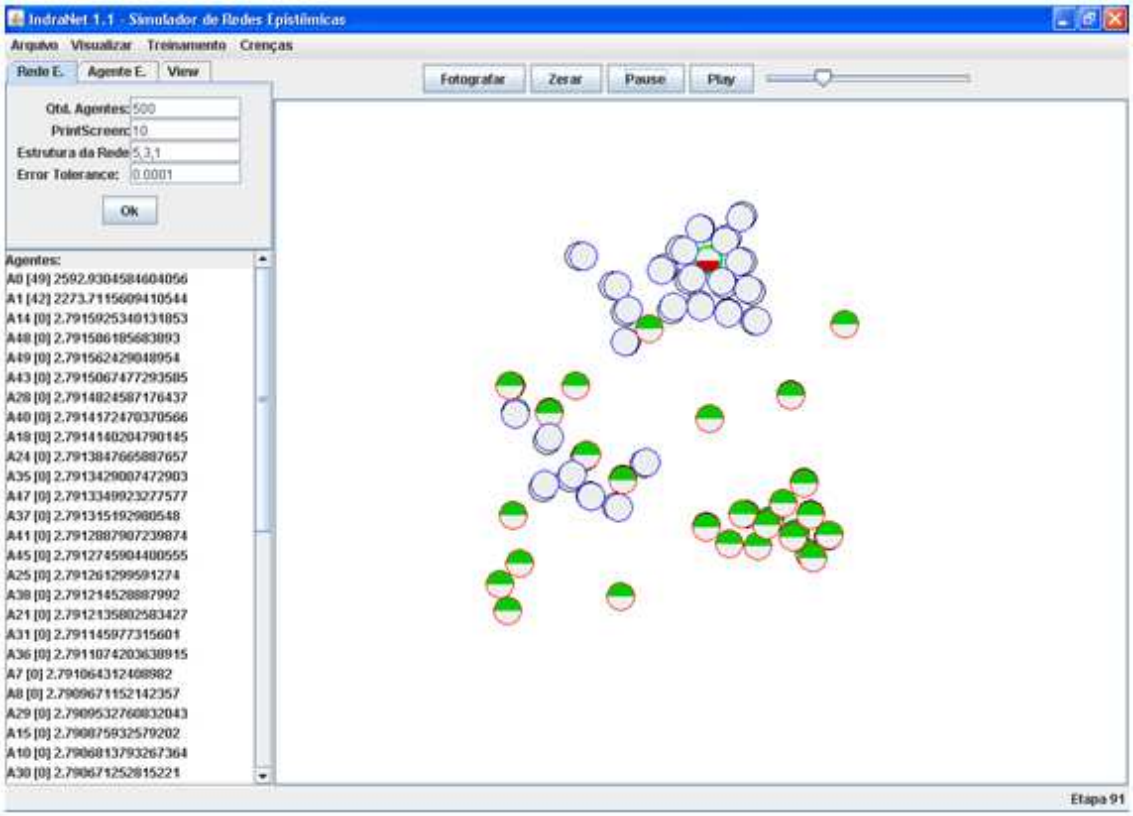


Figura 6.24 – Exemplo 3 após 91 etapas

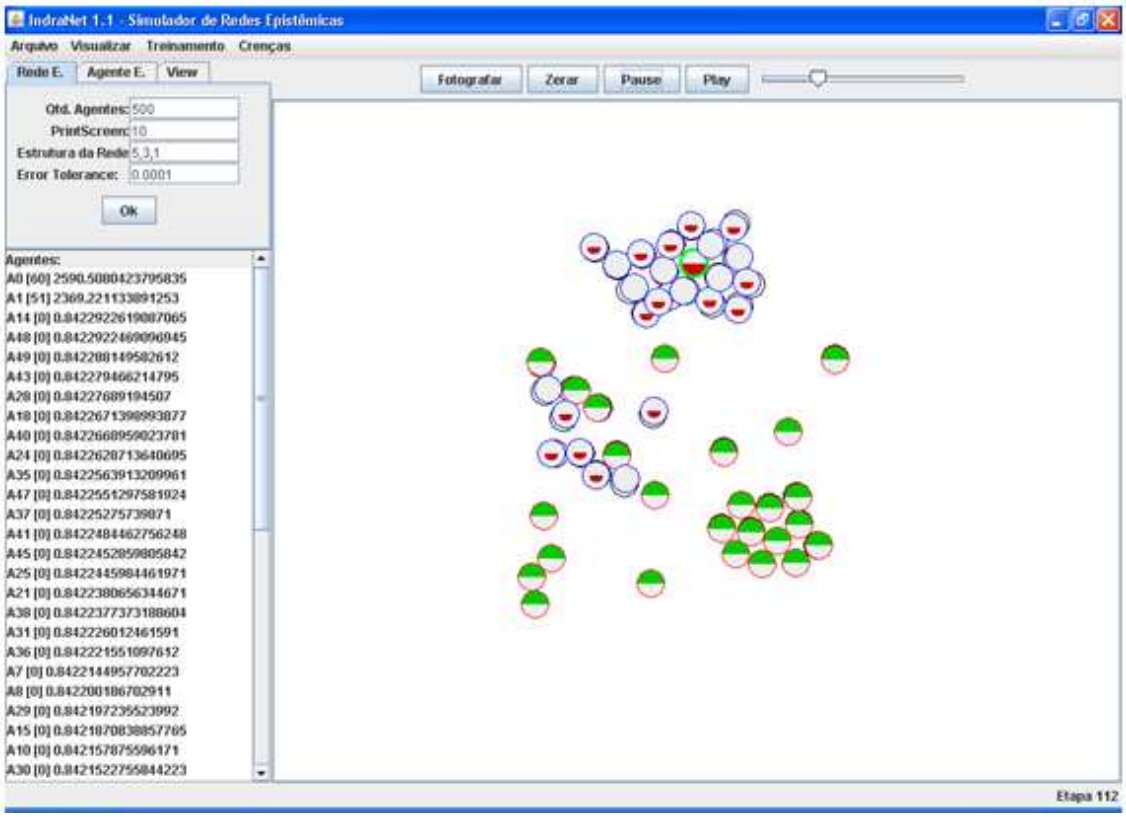


Figura 6.25 – Exemplo 3 após 112 etapas

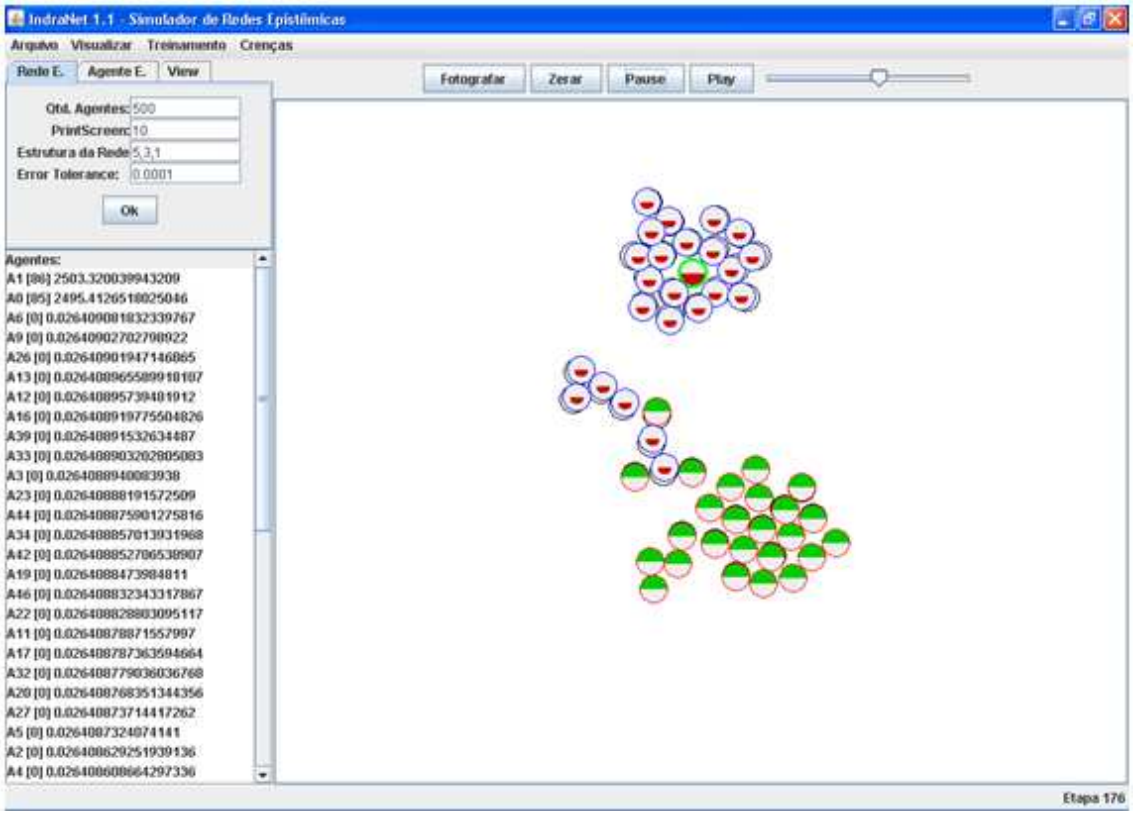


Figura 6.26 – Exemplo 3 após 176 etapas

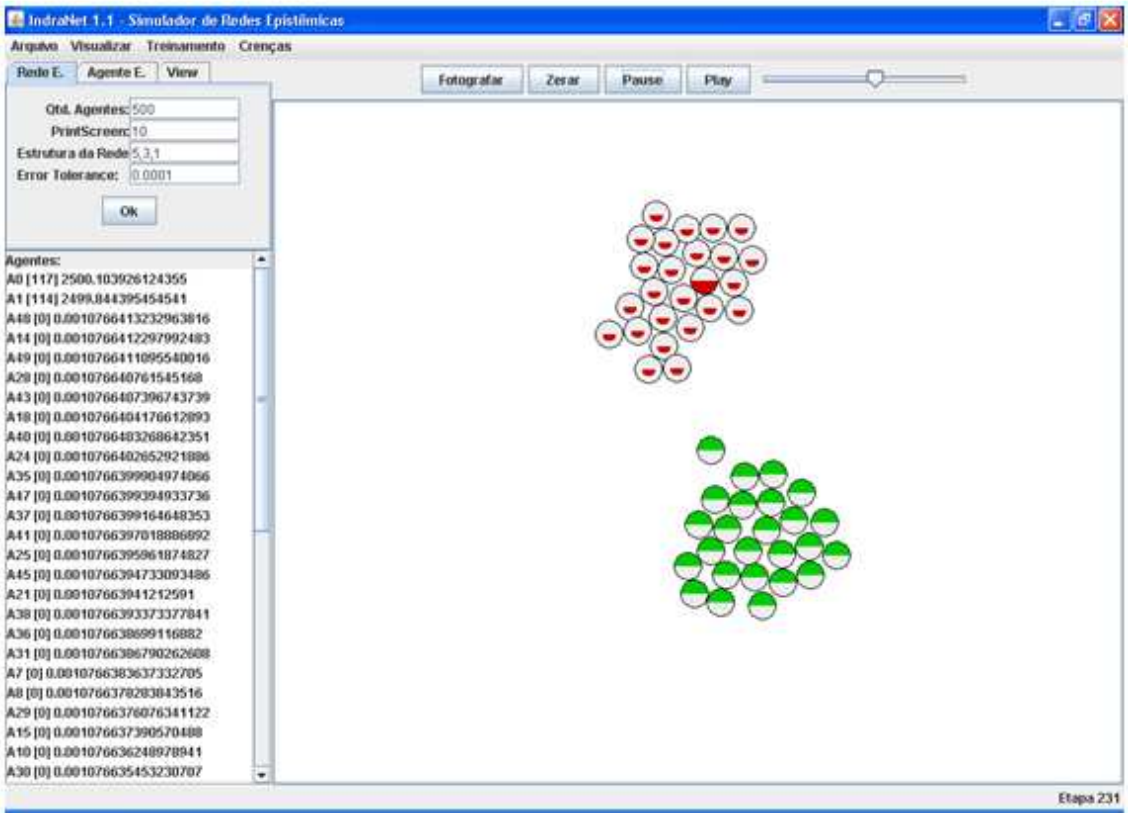


Figura 6.27 – Exemplo 3 após 231 etapas

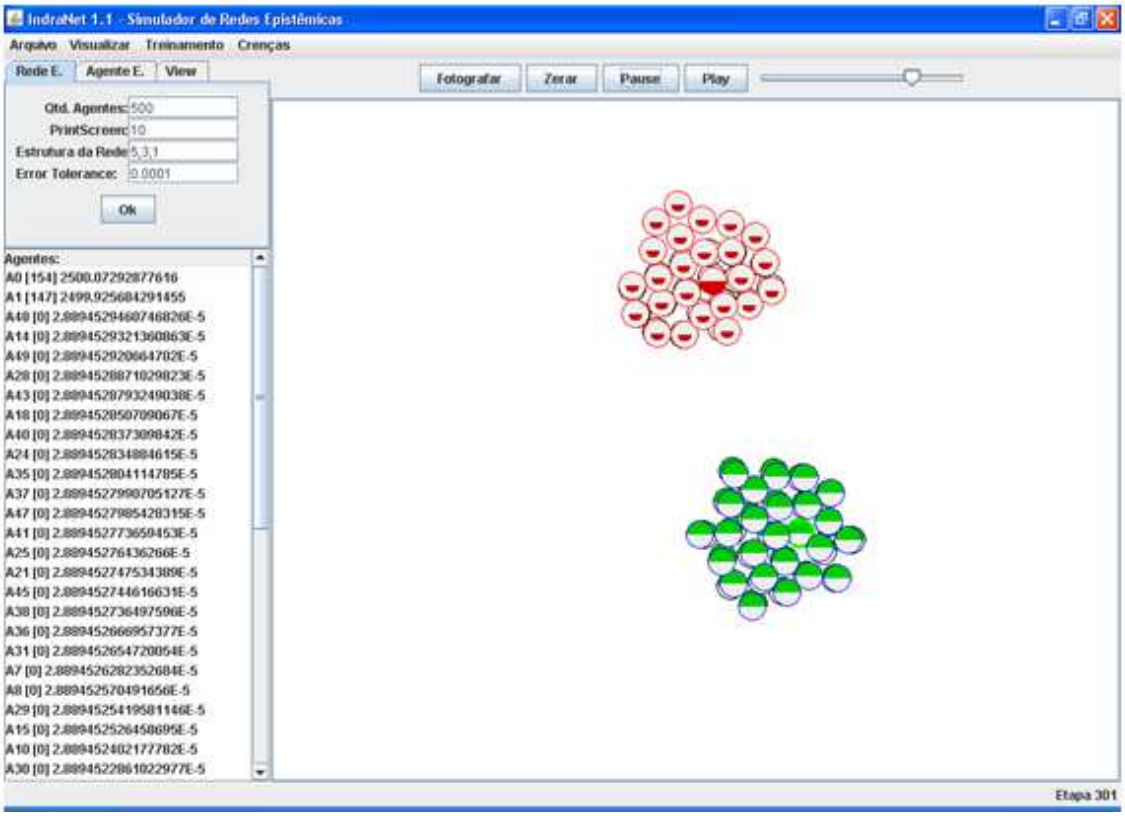


Figura 6.28 – Exemplo 3 após 301 etapas (formação de *clusters* concorrentes)

6.4. Formação de Consenso

No 4º exemplo a ser apresentado, considera-se a situação em que quatro agentes epistêmicos distintos inicialmente possuem crenças diferentes, e interagem até a obtenção de uma situação de consenso, na qual a rede se estabiliza em uma configuração em que os agentes adquirem algumas das crenças originais dos seus pares. Tal exemplo pode ser interpretado como uma generalização dos modelos de formação de consenso de DeGroot [19] e de Axelrod [4].

Na tela inicial de configuração (Figura 6.29), vê-se a escolha das cores das crenças inicialmente treinadas, considerando-se uma estrutura interna dos agentes (rede neuronal de aprendizado por retro-propagação) no formato 5, 7, 1 (sete camadas internas). As frequências de comunicação são iguais para todos os quatro agentes, e os pesos iniciais são distribuídos de forma homogênea (não há agentes mais ou menos importantes).

Após um número de etapas, os agentes se aproximam gradualmente, e vão sendo “convertidos” para as crenças dos demais, de forma não-linear, até que chegam a um consenso a partir do qual não mais se alteram (Figura 6.36).

Uma outra maneira de interpretar o experimento é como um modelo de um ambiente de “aprendizagem colaborativa”, no qual diferentes “aprendizes” colaboram entre si, ensinando e aprendendo (como em uma sala de aula, por exemplo, ou através de um ambiente de *e-learning*).

Vê-se, também, que a distribuição homogênea dos pesos iniciais e a igualdade na frequência de comunicação são condições necessárias para a formação de consenso, como observado em seguidas simulações do Exemplo 4.

Um futuro aprimoramento deste exemplo permitirá variar as características iniciais dos agentes (pesos e frequências de comunicação), estudando as condições suficientes para a formação de consenso.

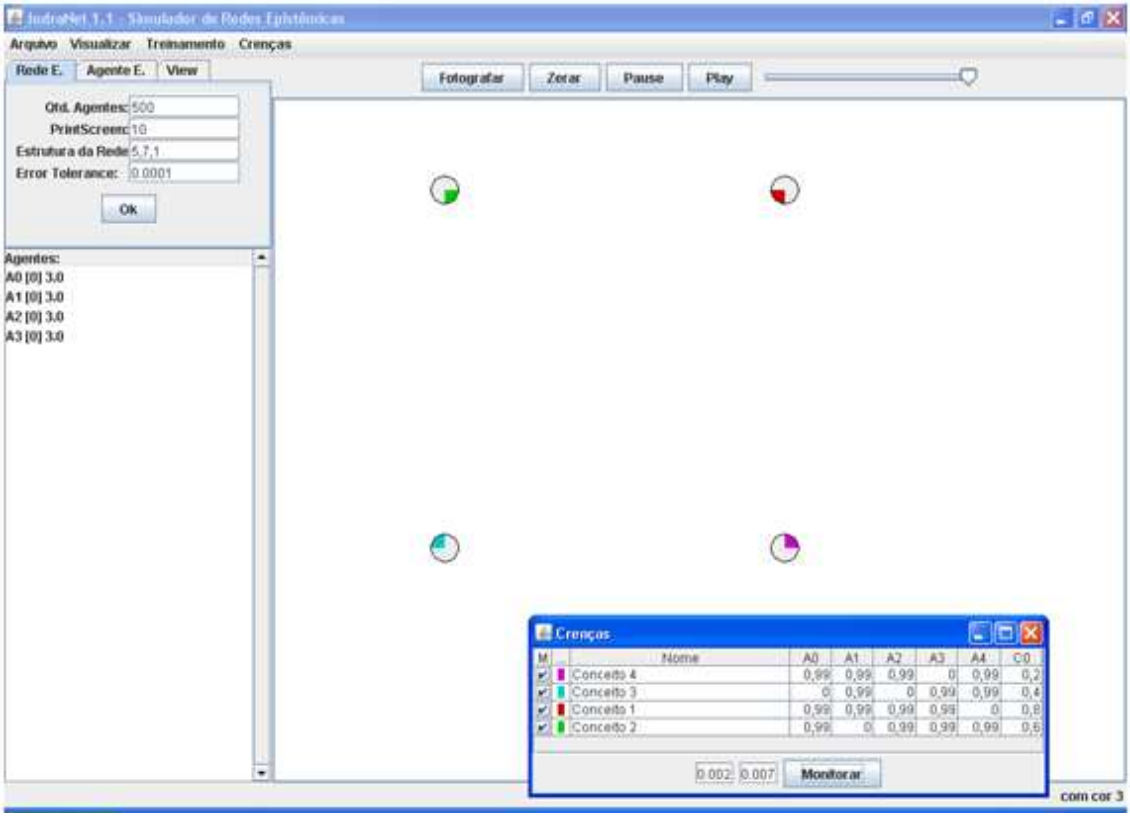


Figura 6.29 – Configuração inicial do Exemplo 4

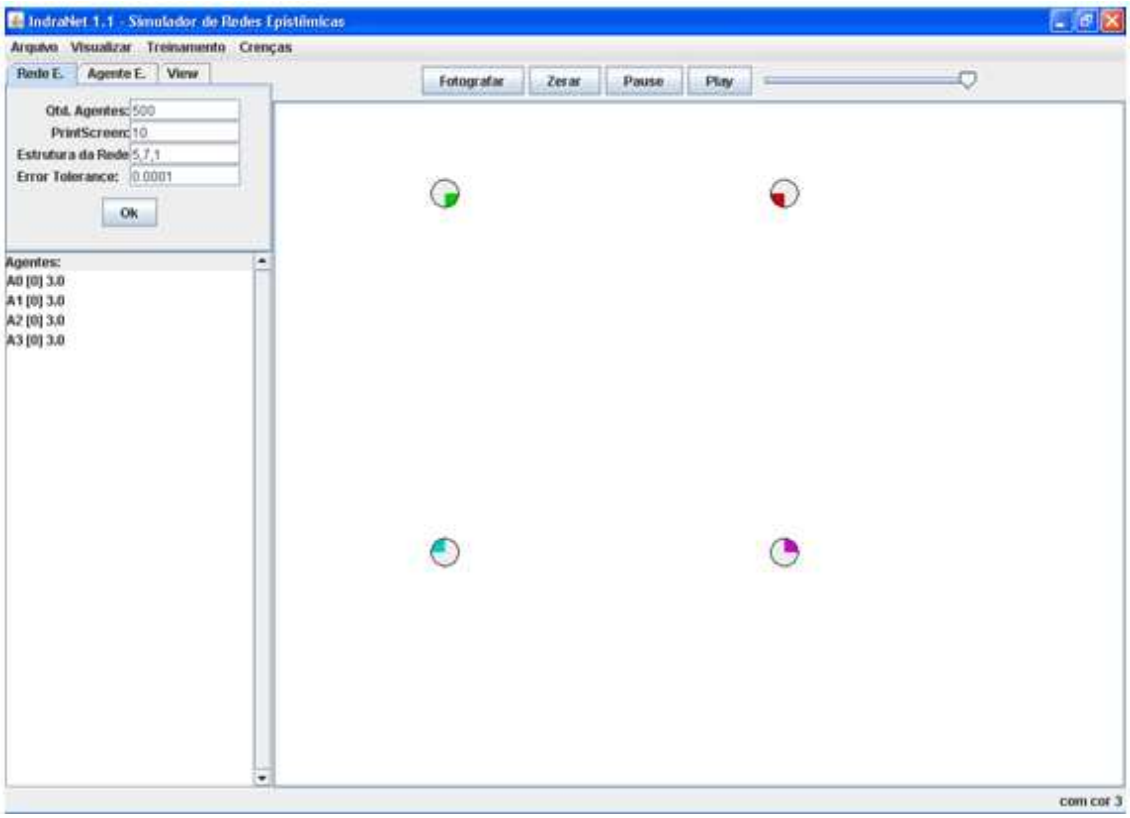


Figura 6.30 – Tela inicial do Exemplo 4

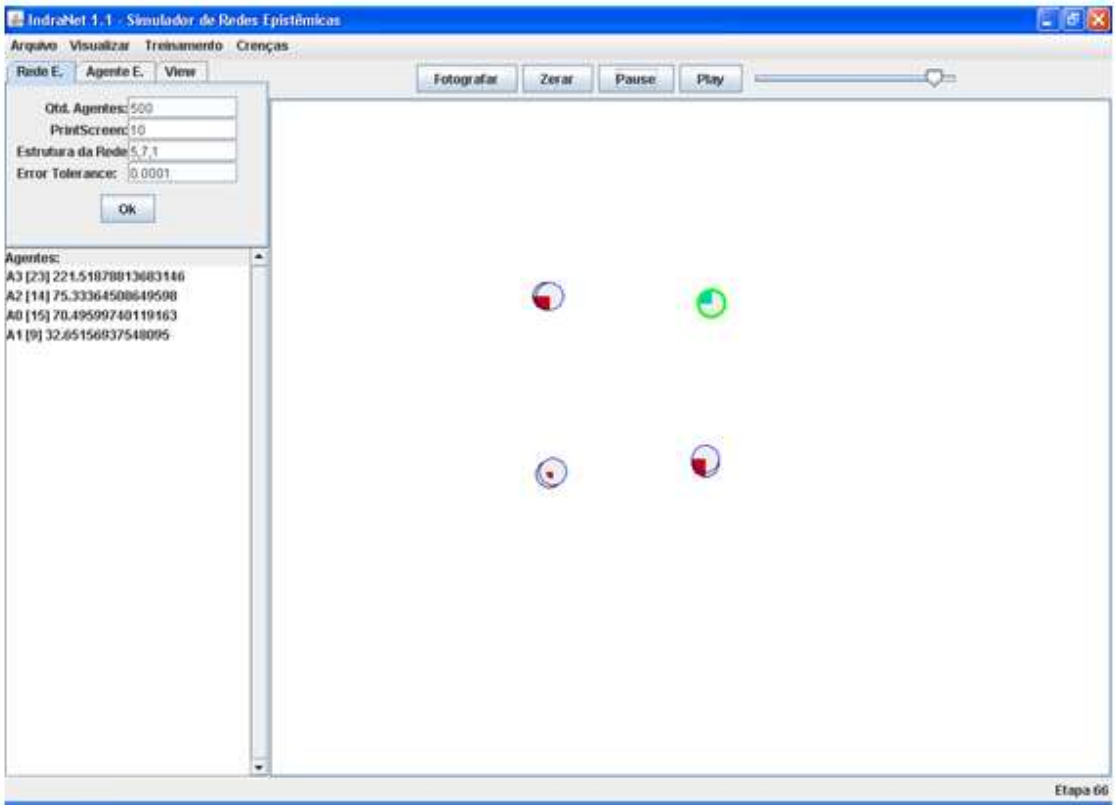


Figura 6.31 – Exemplo 4 após 66 etapas

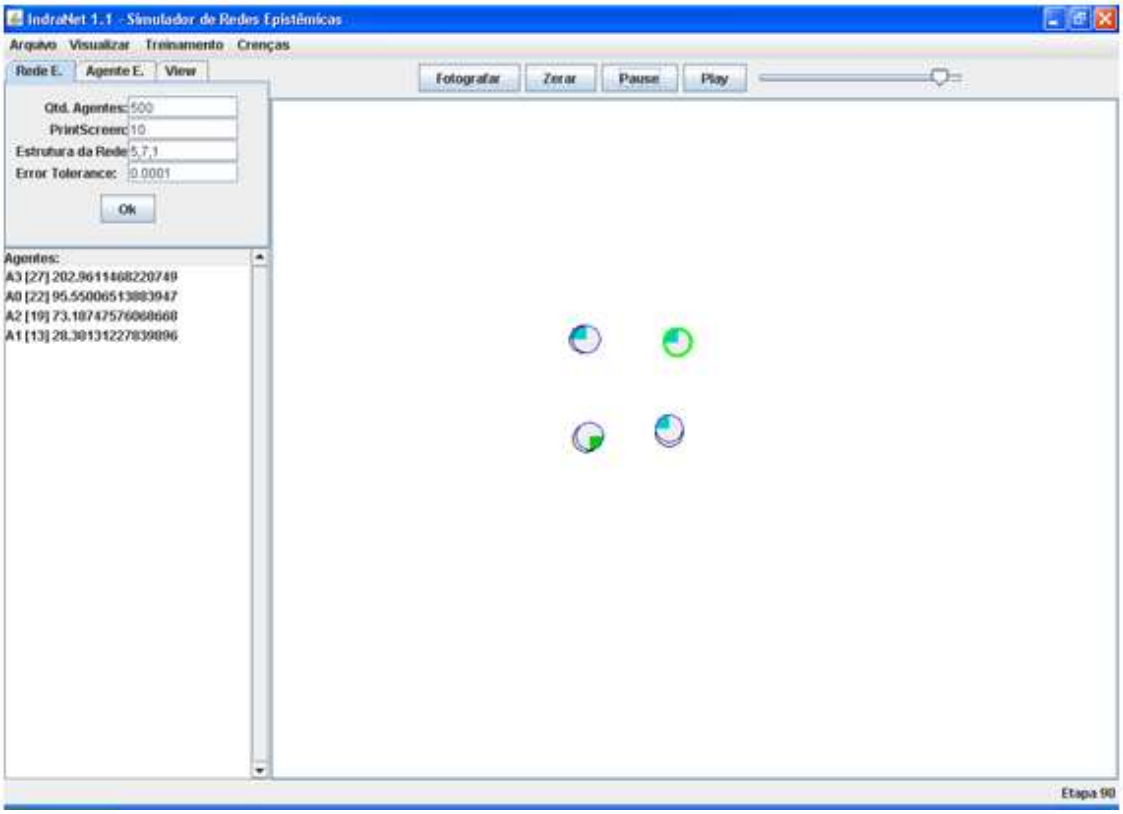


Figura 6.32 – Exemplo 4 após 90 etapas

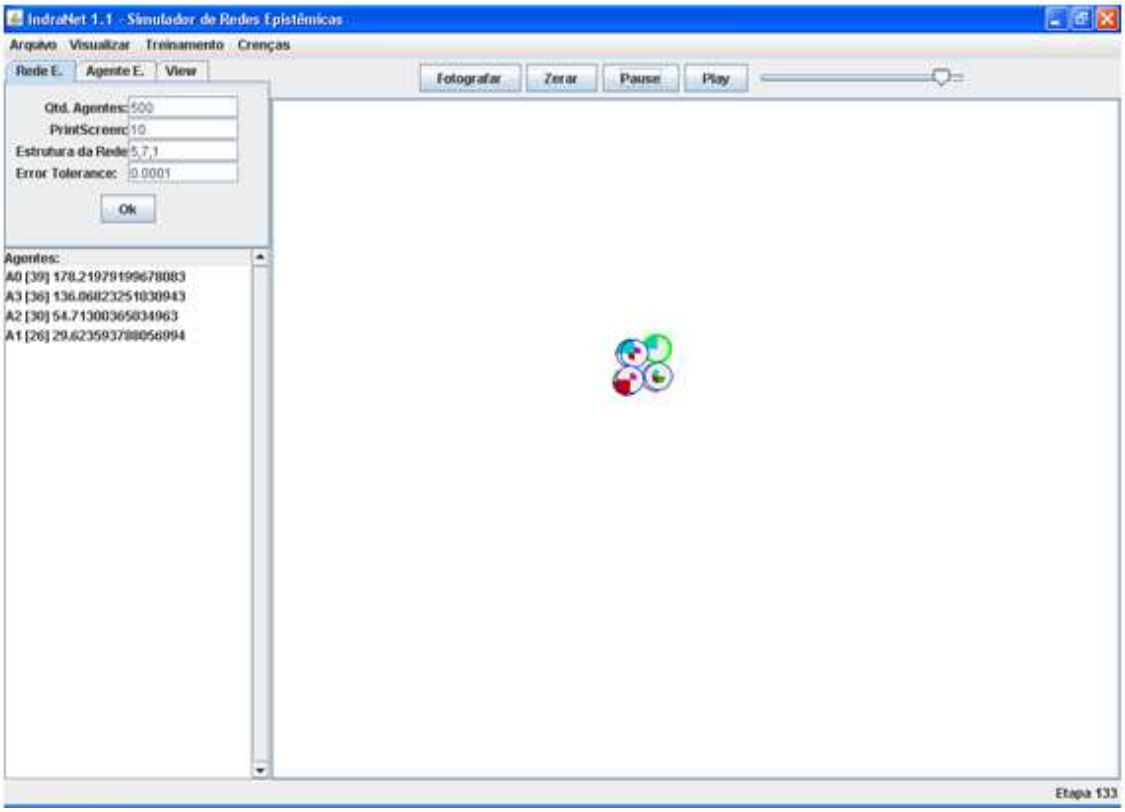


Figura 6.33 – Exemplo 4 após 133 etapas

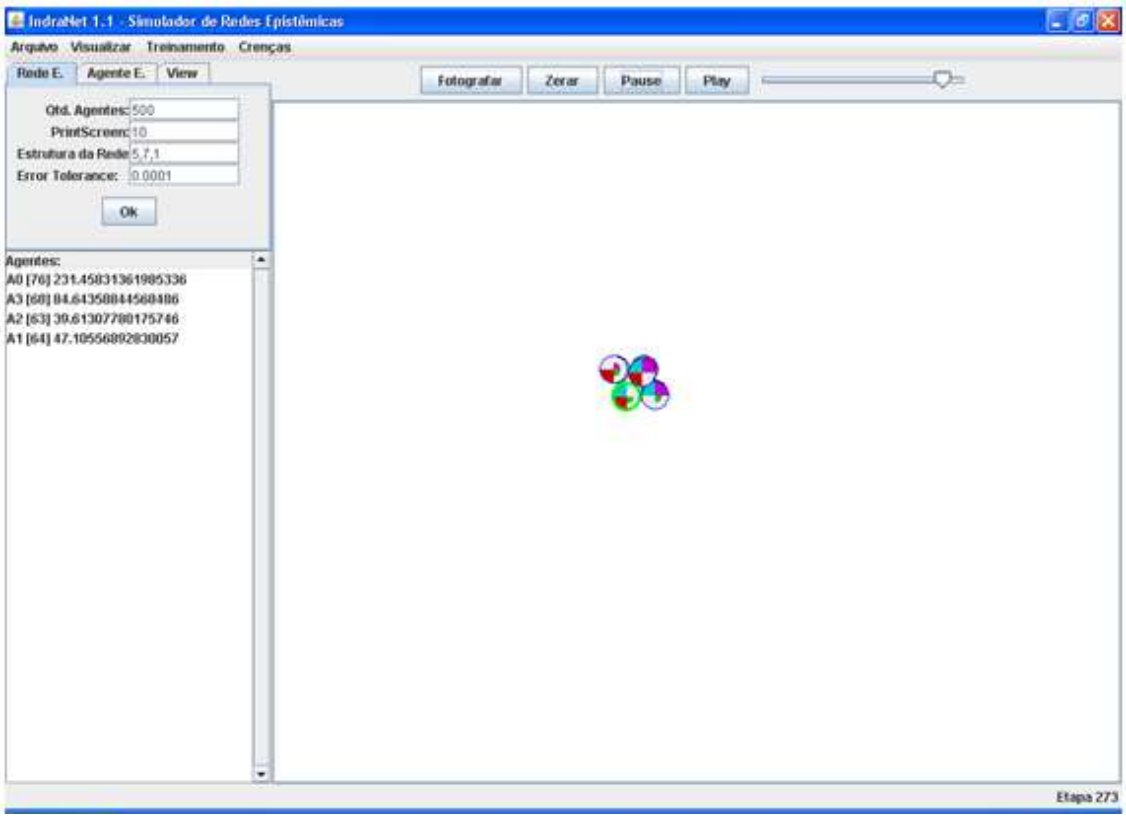


Figura 6.34 – Exemplo 4 após 273 etapas

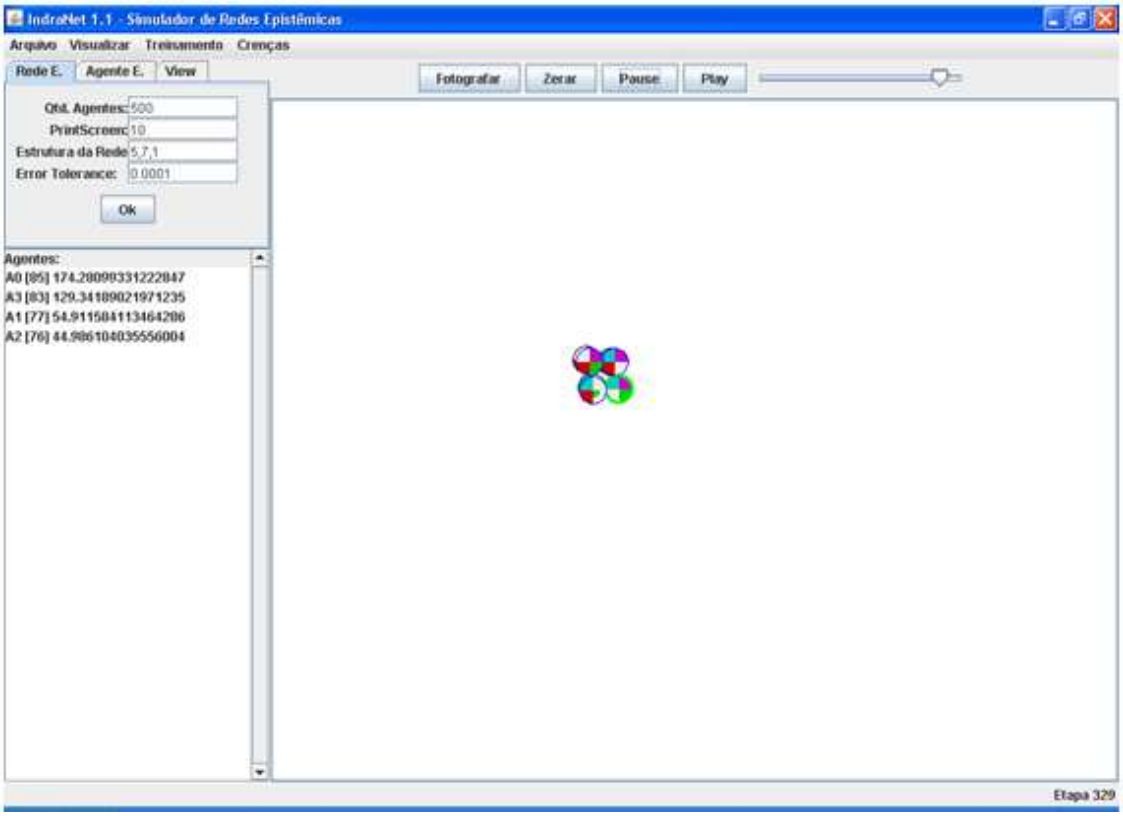


Figura 6.35 – Exemplo 4 após 329 etapas

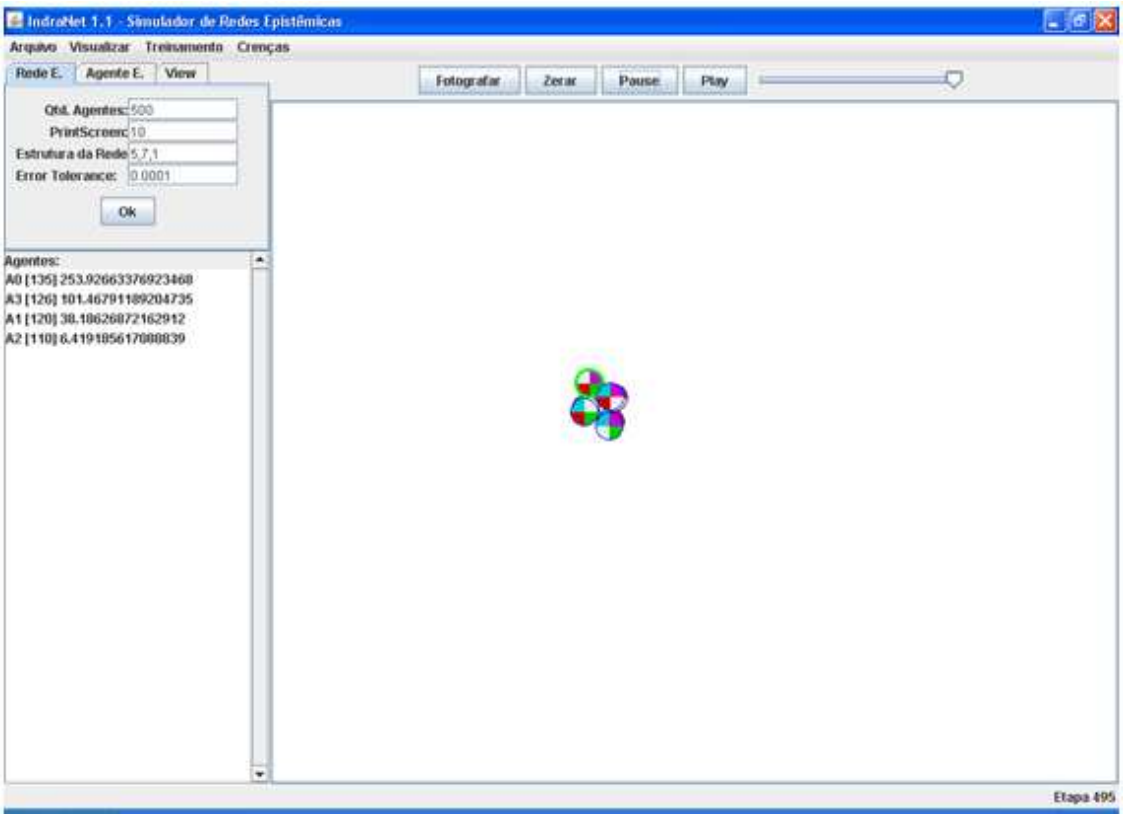


Figura 6.36 – Exemplo 4 após 495 etapas (formação de consenso)

6.5. A Rede Social Orkut

No 5º exemplo escolhido procurou-se captar do mundo real uma rede social de amigos, como o Orkut, ilustrando a aplicabilidade do modelo. O Orkut, assim como o Facebook, o Twitter e vários outros exemplos de redes sociais na Web, constitui-se como um espaço virtual no qual amigos e conhecidos conectam-se a outros amigos e conhecidos, afirmando seus interesses e preferências e trocando informações e conhecimentos.

Na Figura 6.37 vê-se um exemplo de um perfil de usuário do Orkut, obtido mediante prévia autorização e mantida a necessária privacidade.

Figura 6.37 – Perfil de um usuário do Orkut (com permissão)

De forma a modelar um subconjunto de usuários do Orkut como uma rede epistêmica, partiu-se das seguintes premissas:

(a) Os pesos (Ws) iniciais de cada agente epistêmico foram definidos a partir das preferências dos usuários do Orkut (“Meus amigos”);

(b) As frequências de comunicação de cada agente epistêmico foram estimadas a partir da quantidade de *scraps* (mensagens) por ele produzidas;

(c) As crenças de cada agente epistêmico foram identificadas a partir dos interesses e comunidades a que os agentes pertencem (“Minhas Comunidades”).

Na Figura 6.38, vê-se o grafo criado a partir de uma comunidade “realista” (e não real) de amigos do Orkut. Dizemos que o grafo é “inspirado” em uma comunidade de amigos, pois devido às questões de privacidade e sigilo da própria ferramenta, nem todos os dados sobre preferências e interesses pessoais encontram-se abertos e à disposição para a modelagem. São trinta e um nós/usuários/agentes epistêmicos, e os arcos entre eles representam as importâncias que eles se atribuem, a partir das suas relações de amizade. Os nós em amarelo são usuários com maior frequência de comunicação.

Na Figura 6.39, vê-se o conjunto de comunidades (crenças) que foram utilizadas no treinamento inicial dos agentes da rede epistêmica. Para facilitar a modelagem e a visualização, foram identificadas apenas as quatro principais crenças de cada usuário, em um total de quarenta crenças (comunidades) distintas.

A Figura 6.40 apresenta as crenças modeladas inicialmente, associando-se a cada uma valores distintos de antecedentes e de consequentes. Para acompanhar a dinâmica da simulação, escolheram-se quatro cores para representar 4 crenças distintas:

(a) “Salgueiro” – cor azul;
(b) “Livros”- cor verde;
(c) “Novelas da Globo” – cor vermelha;
(d) “Eu confio em Deus” – cor roxa.

Na Figura 6.41 vê-se a configuração inicial do Exemplo 5, já com as cores atribuídas às crenças selecionadas para acompanhamento. Pode-se notar que os amigos mais “populares” inicialmente possuem somatórios de Ws mais altos (medida da importância), o que indica um potencial maior de influência sobre os outros.

Após uma sequência de etapas, nota-se que os agentes aglutinam-se em torno do agente 9 (inicialmente mais popular), e que as crenças oscilam entre as cores azul, verde e vermelha.

Ao final, na Figura 6.51, pode-se ver pelo gráfico de distribuição de pesos dos agentes que, como no Exemplo 1, também forma uma Lei de Potência, com um dos agentes recebendo uma grande importância (Ws) dada pelos outros agentes.

Pode-se entender o resultado final de simulação do Exemplo 5 (Orkut) a partir da noção de aprendizagem competitiva 44], através da qual diversas crenças disputam a atenção dos usuários simultaneamente. O resultado demonstra que os agentes inicialmente mais “populares” têm maior capacidade de influenciar os demais, e que as preferências dos usuários oscilam periodicamente, ora apontando em uma direção, ora apontando em outra direção. Tal fenômeno expressa a natureza emergente das comunidades no Orkut e a dinâmica da propagação de idéias em uma rede social virtual.

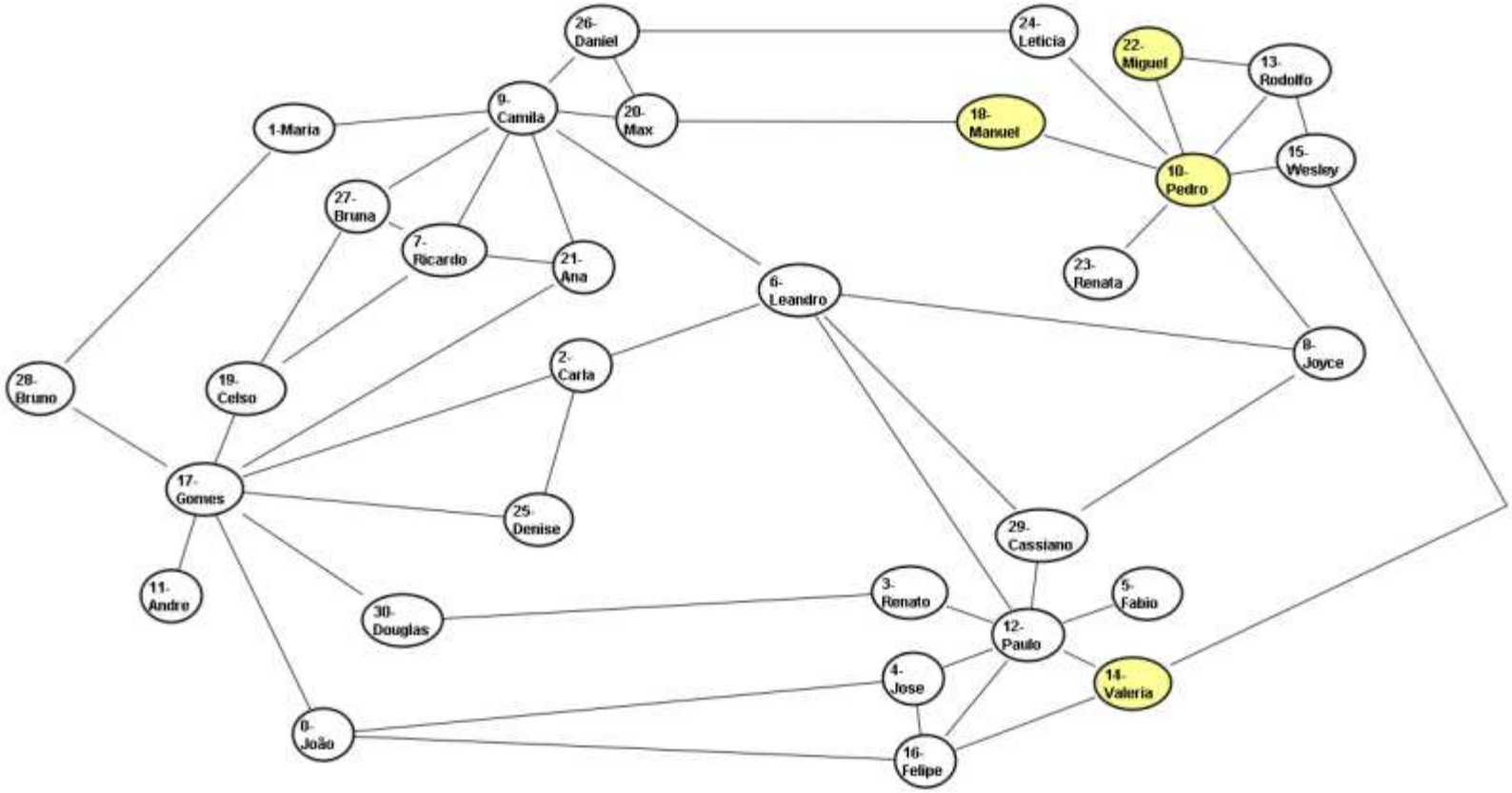


Figura 6.38 – Grafo do Exemplo do Orkut

João	Recomeçar é preciso Java Unicarioca PSP Brasil C/C++	Maria	Só bebo skol O Livreiro Eu amo minha mãe Jogos de Nave Gears of War	Carla	God of War Game Development Eu confio em Deus Salgueiro	Renato	Save Galvão Birds Lost Experience John Lock é meu pai Festas Infantis e Afins	José	Simpsons Caverna do Dragão Elite Photoshop Fotografia	Fabio	Milkshake do Bobs Dá Hadouken Ryu O Curioso Caso de Benjamin Button Quero ser presidente do Brasil
Leandro	Clarice Lispector Receitas Fáceis Filmes Antigos Game Retrô	Ricardo	Seu Madruga Merece uma Estátua Diga não as drogas Livros Anos 80	Joyce	Fotografias Noturnas Jim Carrey Silvio Santos é um robô Cazuza	Camila	Novelas da Globo Sou Fã do Silvio Santos Recomeçar é preciso Java	Pedro	Unicarioca PSP Brasil C/C++ Só bebo skol	André	O Livreiro Eu amo minha mãe Jogos de Nave Gears of War
Paulo	God of War Game Development Eu confio em Deus Salgueiro 1 1 2 3 5 8 13	Rodolfo	Save Galvão Birds Lost Experience John Lock é meu pai Festas Infantis e Afins	Valéria	Simpsons Caverna do Dragão Elite Photoshop Fotografia	Wesley	Milkshake do Bobs Dá Hadouken Ryu O Curioso Caso de Benjamin Button Quero ser presidente do Brasil	Felipe	Clarice Lispector Receitas Fáceis Filmes Antigos Game Retrô	Gomes	Seu Madruga Merece uma Estátua Diga não as drogas Livros Anos 80
Manual	Fotografias Noturnas Jim Carrey Silvio Santos é um robô Cazuza	Celso	Novelas da Globo Sou Fã do Silvio Santos Recomeçar é preciso Java	Max	Unicarioca PSP Brasil C/C++ Só bebo skol	Ana	O Livreiro Eu amo minha mãe Jogos de Nave Gears of War	Miguel	God of War Game Development Eu confio em Deus Salgueiro	Renata	Save Galvão Birds Lost Experience John Lock é meu pai Festas Infantis e Afins
Letícia	Simpsons Caverna do Dragão Elite Photoshop Fotografia	Denise	Milkshake do Bobs Dá Hadouken Ryu O Curioso Caso de Benjamin Button Quero ser presidente do Brasil	Daniel	Clarice Lispector Receitas Fáceis Filmes Antigos Game Retrô	Bruna	Seu Madruga Merece uma Estátua Diga não as drogas Livros Anos 80	Bruno	Fotografias Noturnas Jim Carrey Silvio Santos é um robô Cazuza	Cassiano	Novelas da Globo Sou Fã do Silvio Santos Recomeçar é preciso Java
Douglas	Novelas da Globo Sou Fã do Silvio Santos Recomeçar é preciso Java										

Figura 6.39 – Subconjunto de Usuários e seus interesses no Orkut

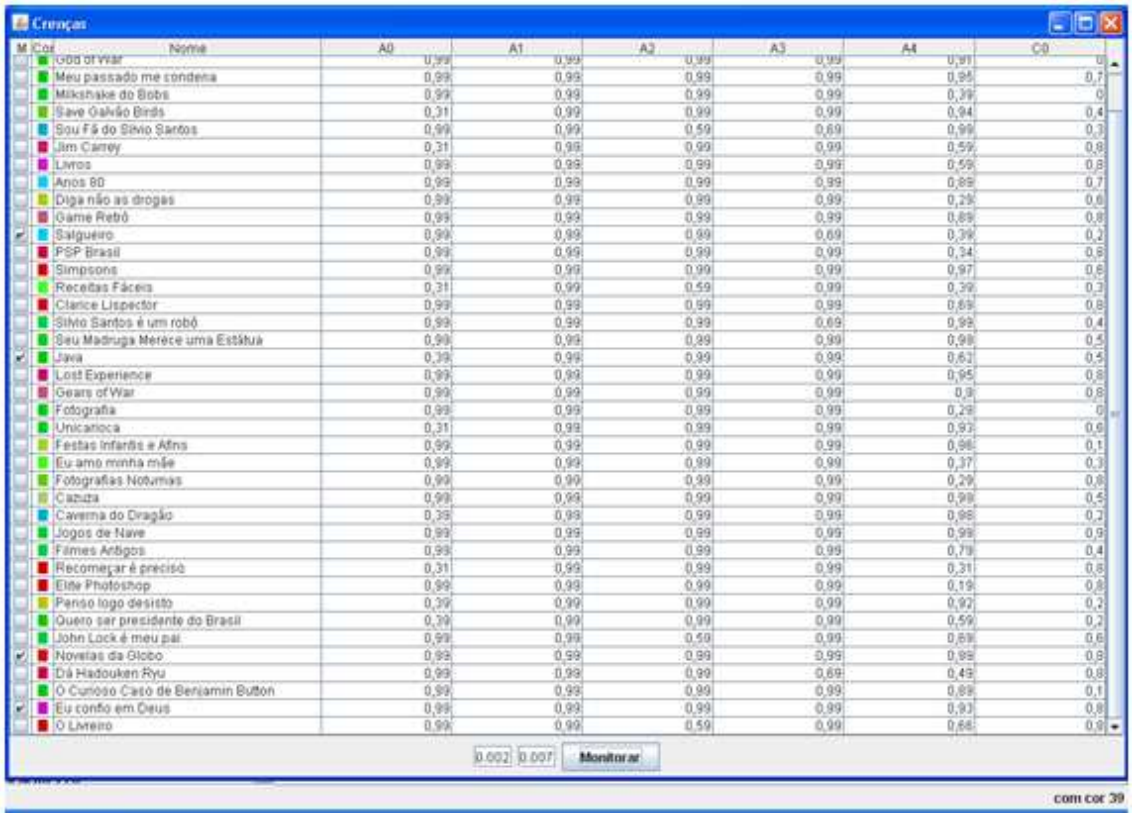


Figura 6.40 – Crenças Modeladas do Exemplo 5

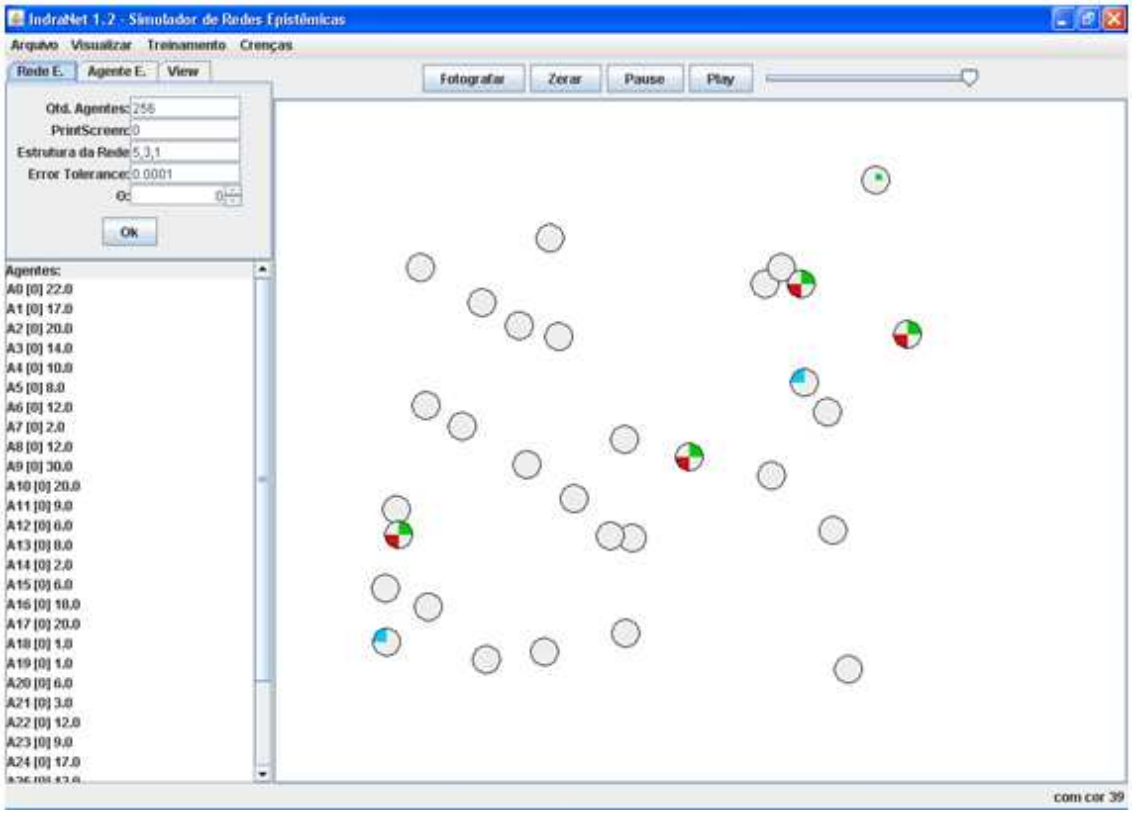


Figura 6.41 – Tela inicial do Exemplo 5

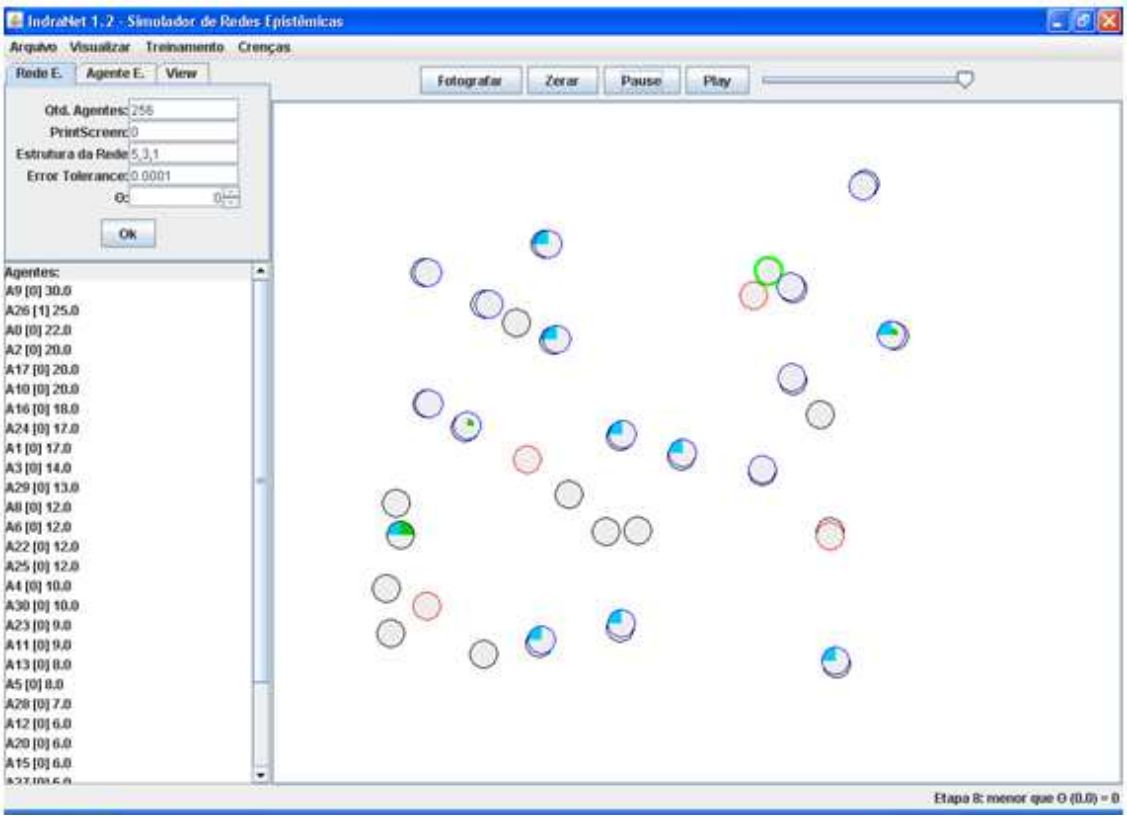


Figura 6.42 – Exemplo 5 após 6 etapas

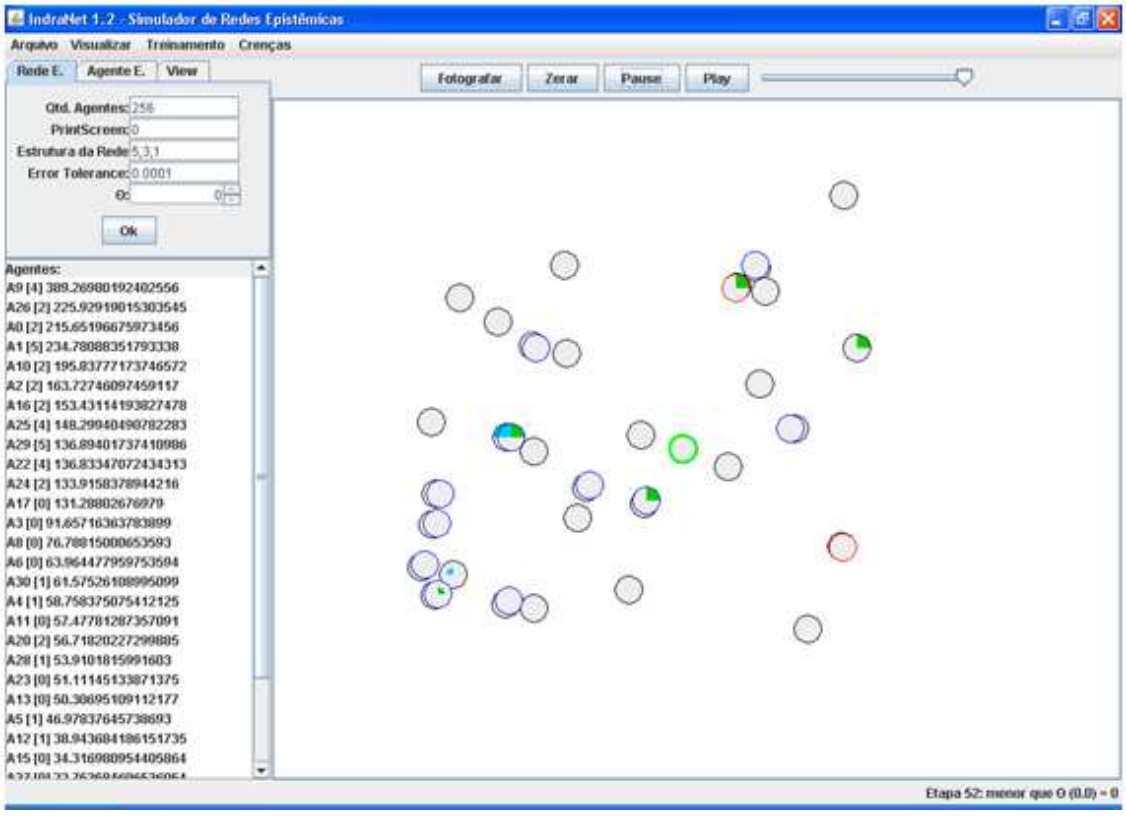


Figura 6.43 – Exemplo 5 após 52 etapas

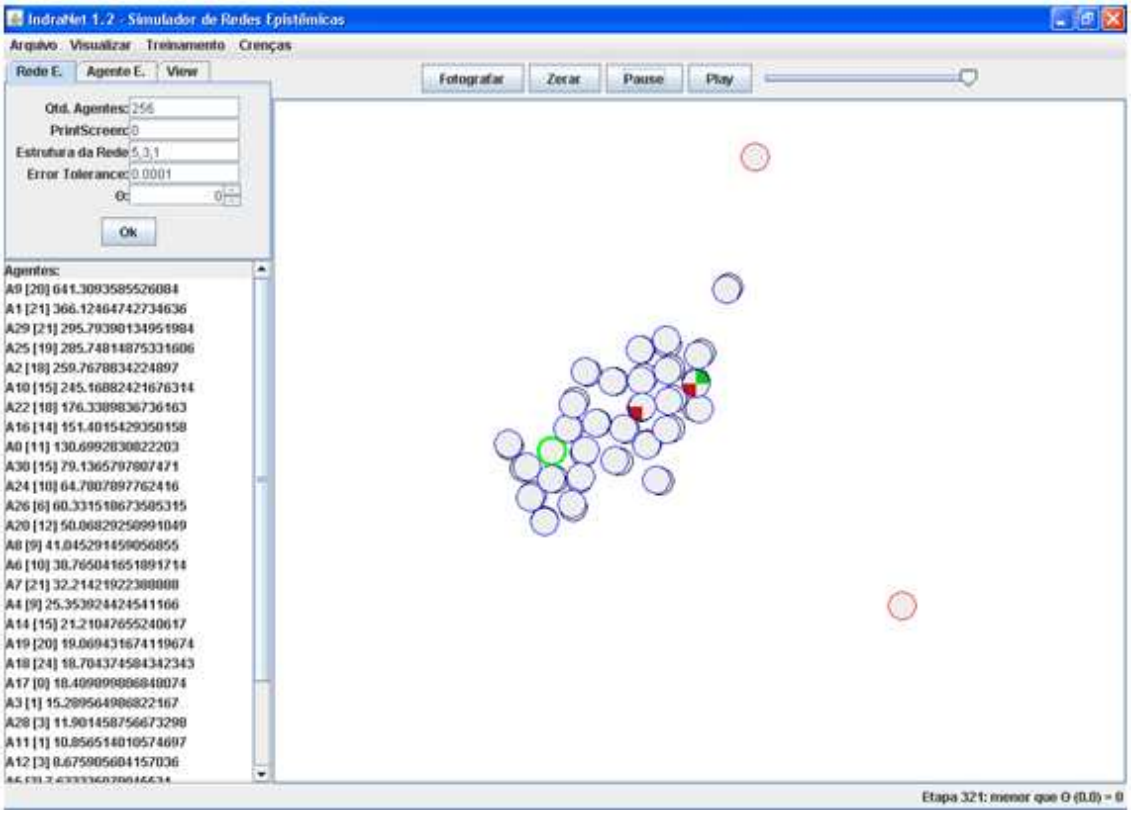


Figura 6.44 – Exemplo 5 após 321 etapas

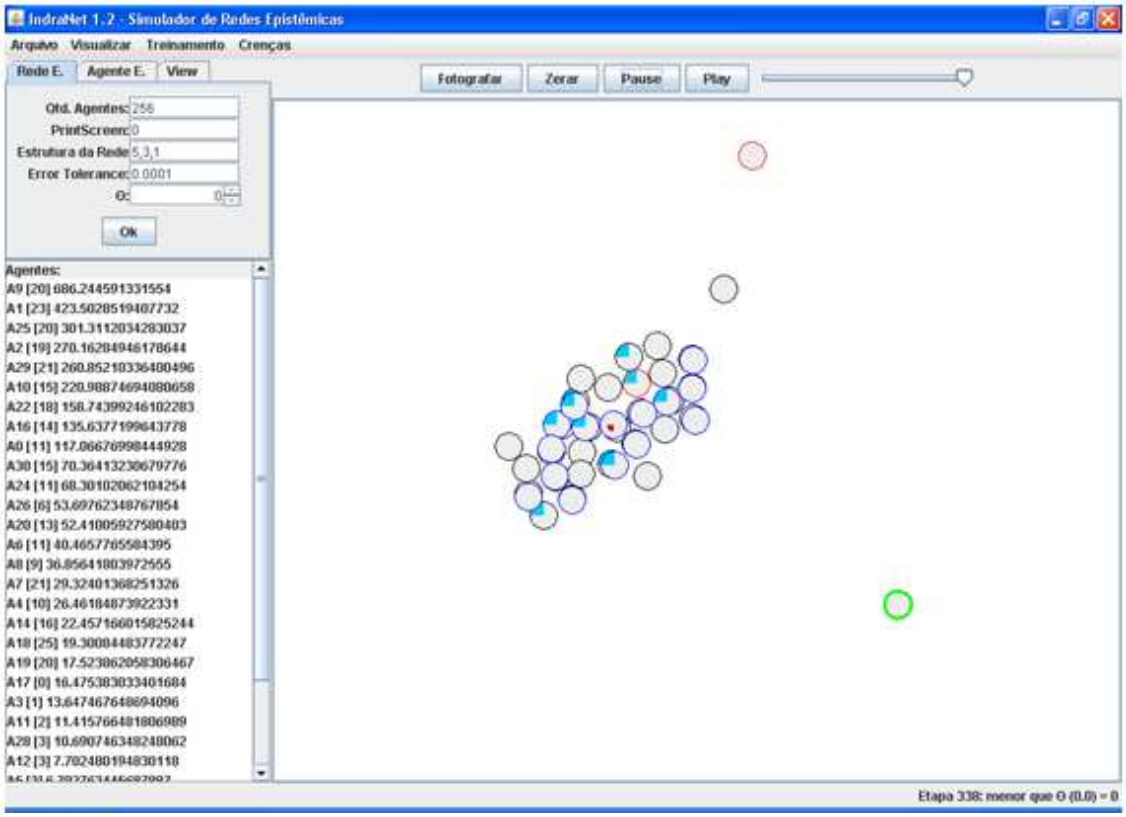


Figura 6.45 – Exemplo 5 após 338 etapas

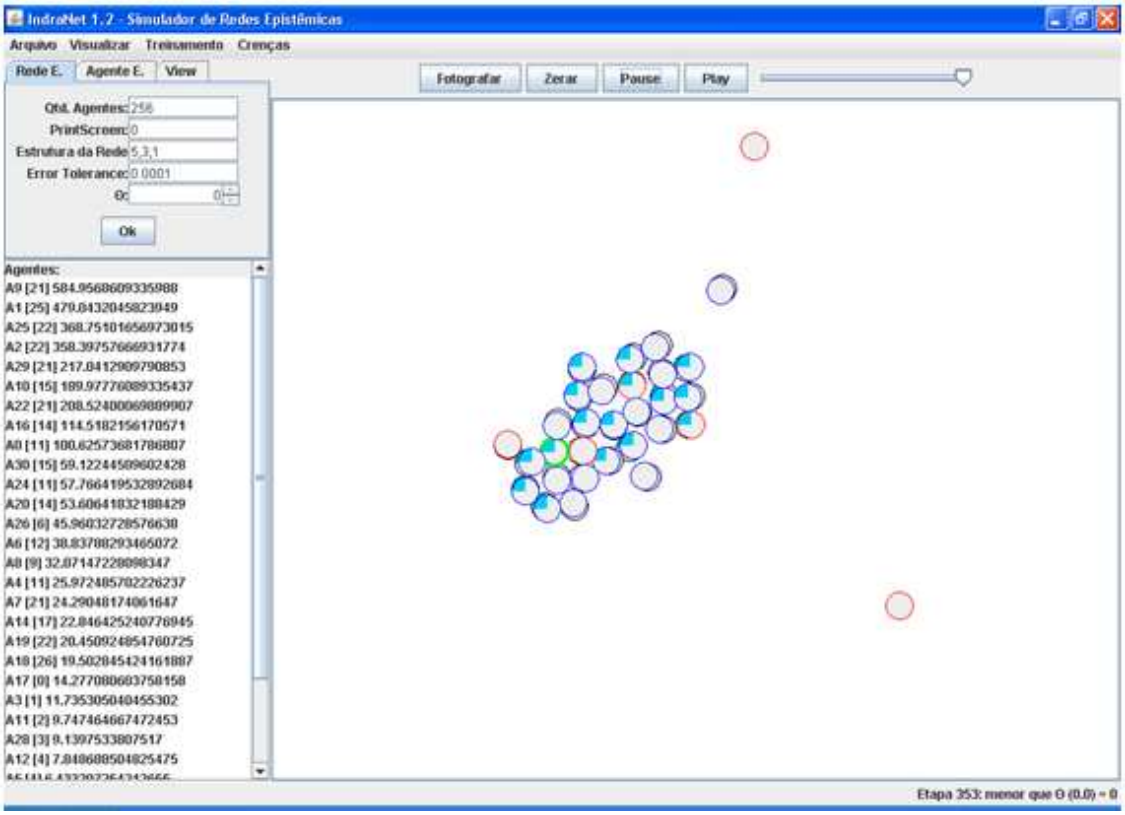


Figura 6.46 – Exemplo 5 após 353 etapas

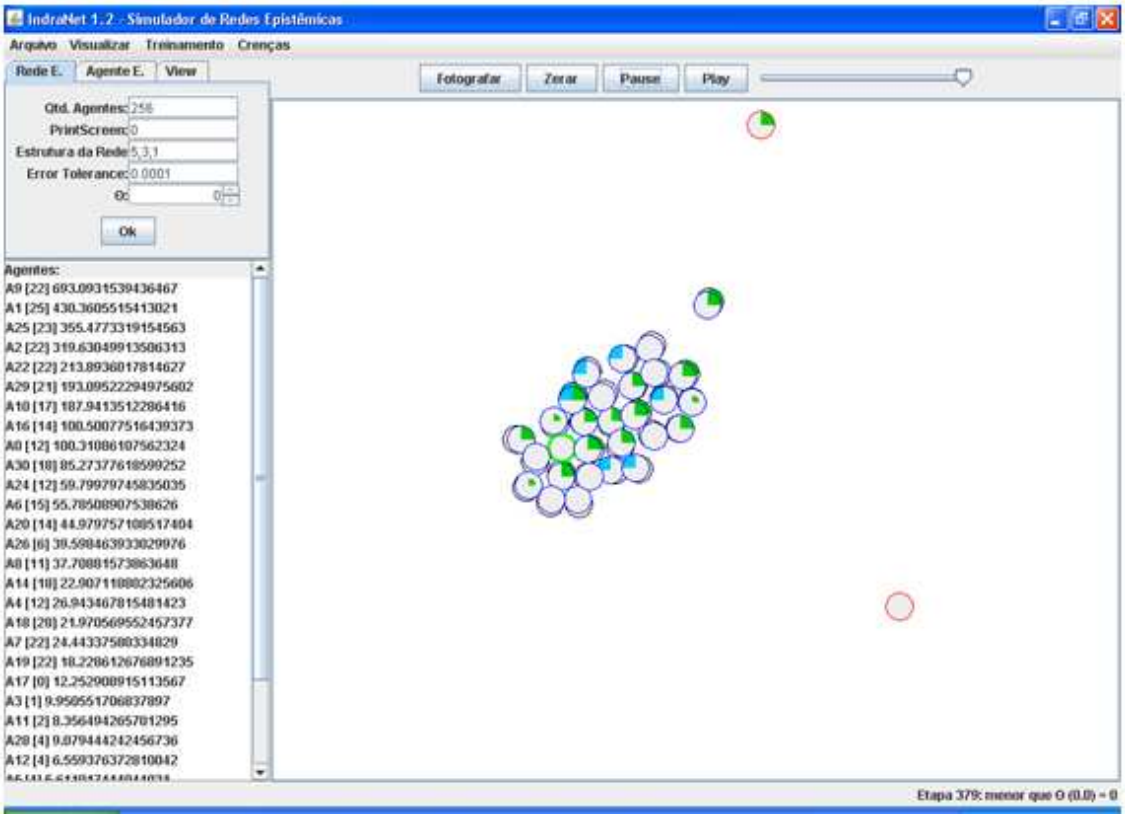


Figura 6.47 – Exemplo 5 após 379 etapas

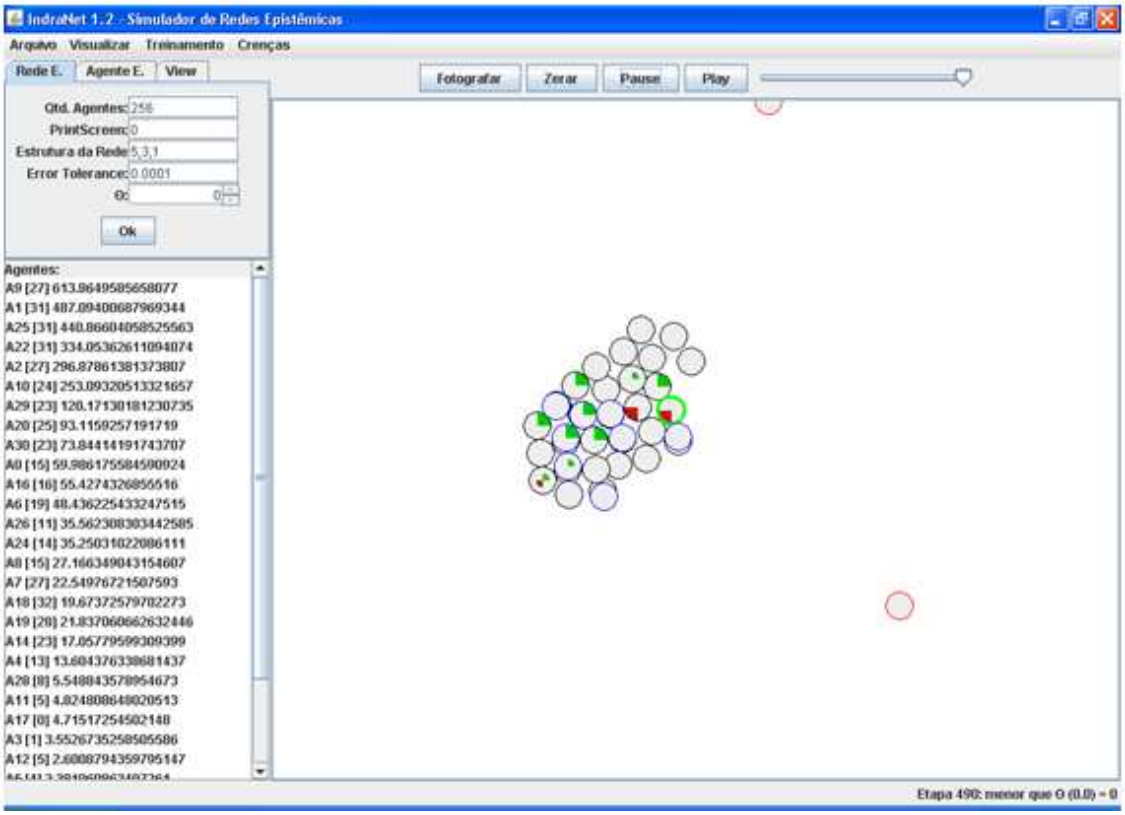


Figura 6.48 – Exemplo 5 após 490 etapas

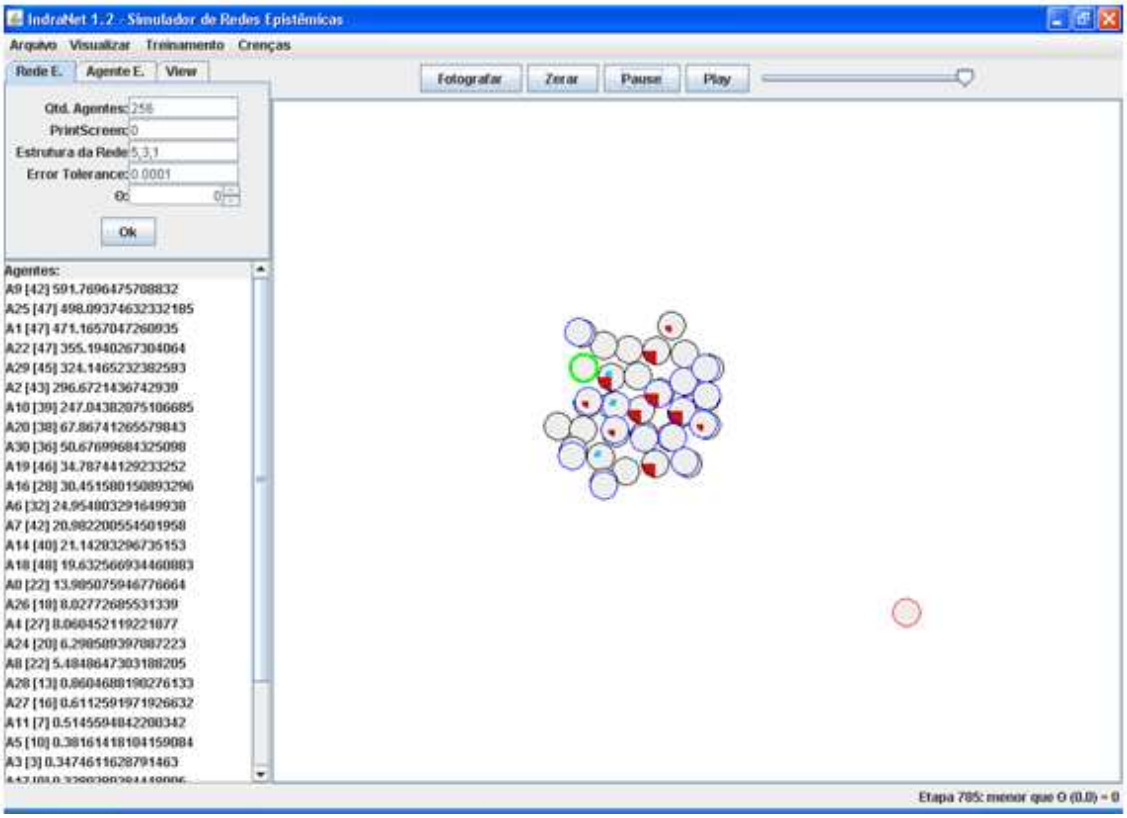


Figura 6.49 – Exemplo 5 após 785 etapas

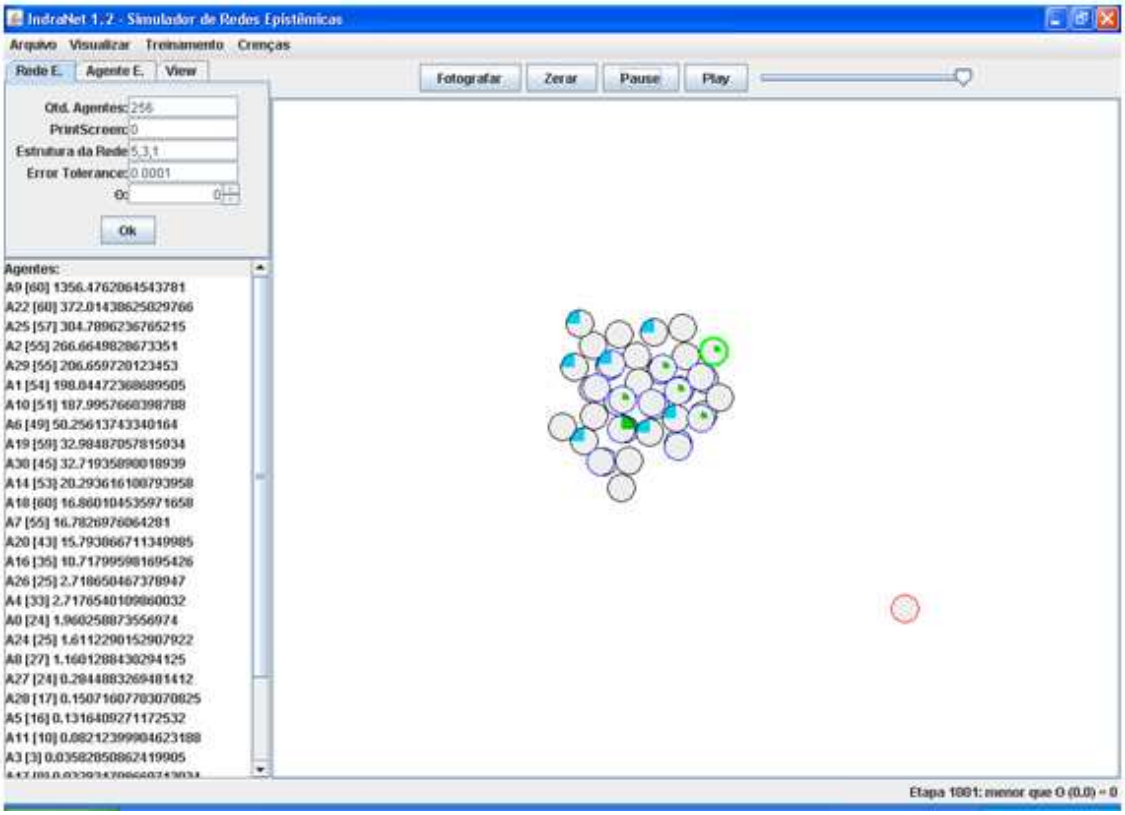


Figura 6.50 – Exemplo 5 após 1001 etapas

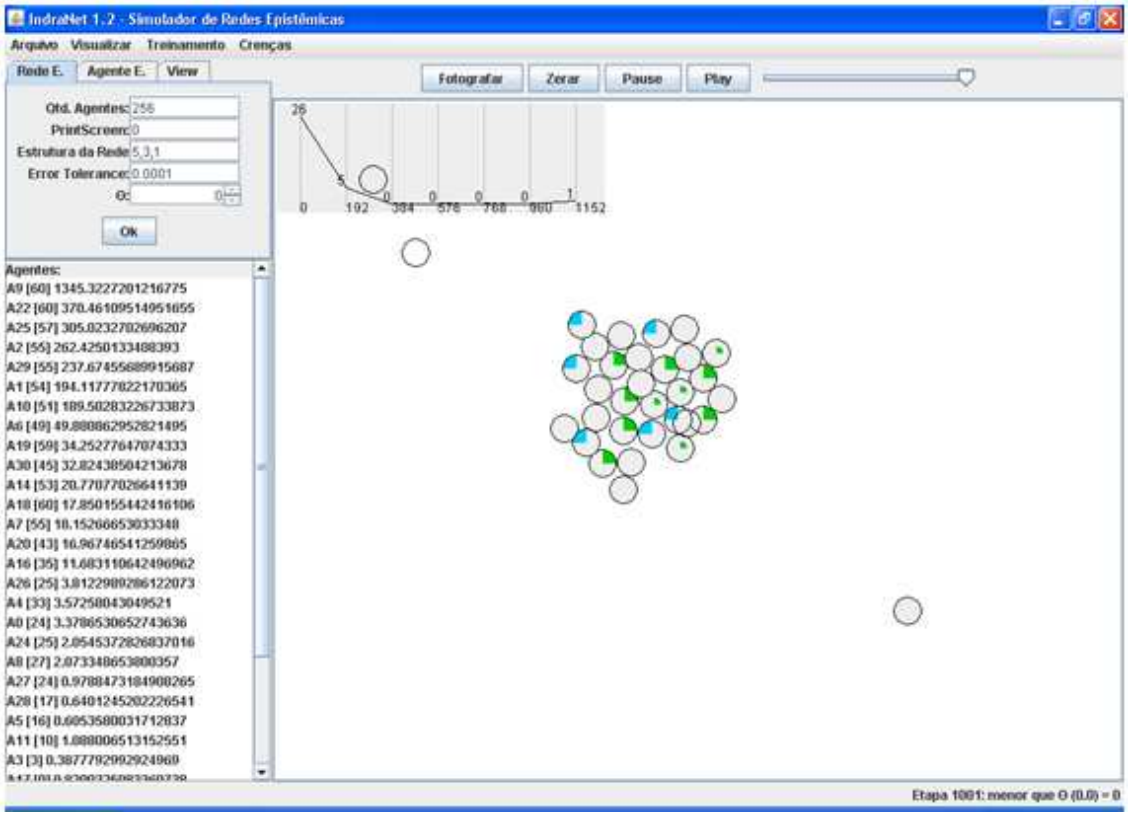


Figura 6.51 – Situação final do Exemplo 5 com gráfico de distribuição (Lei de Potência)

7. Conclusões

O objetivo do presente trabalho foi a proposta de um modelo conexionista para a representação da dinâmica da aprendizagem social. Foi feita a justificativa da importância do fenômeno da aprendizagem social nos mais variados campos de estudo, e foram apresentados exemplos que ilustram o potencial de aplicação do modelo para uma ampla gama de situações.

O modelo conexionista proposto nesse trabalho é um modelo simples, com poucos parâmetros, porém capaz de modelar uma variedade grande de fenômenos em que a aprendizagem social está presente.

Entre as contribuições do presente trabalho, destaca-se:

(a) A formalização de conceitos teóricos presentes nos trabalhos de Atlan [2], Tarde [67] e Bandura [5], em um modelo conexionista que se presta à simulação computacional, permitindo a verificação ***in silico*** de propriedades emergentes em redes sociais;

(b) A generalidade e a simplicidade do modelo proposto, que, a partir de um pequeno número de parâmetros e regras, é capaz de modelar diferentes fenômenos relacionados à aprendizagem social;

(c) A demonstração, por simulação, de que as redes epistêmicas se auto-organizam e se estruturam na forma de uma “Lei de Potência”, constituindo-se em um caso especial de redes *scale-free*;

(d) A exemplificação do modelo proposto para os casos de propagação de crenças e formação de consenso, como uma posível generalização dos modelos de DeGroot [19] e Axelrod [4];

(e) A utilização do modelo proposto para formalização de um subconjunto da rede social Orkut, simulando a dinâmica de um grupo de usuários e suas comunidades de interesse.

A capacidade do modelo é muito maior do que o tempo que dispusemos neste trabalho. Algumas limitações do trabalho puderam ser observadas:

(a) A escolha de exemplos simples para ilustrar a aplicabilidade do modelo, sabendo-se que as redes complexas do mundo real possuem muitas outras nuances, que ainda não foram analisadas;

(b) A ausência de uma formalização matemática das propriedades das redes epistêmicas, preferindo-se a construção da simulação computacional para a análise dos fenômenos emergentes;

(c) A simplificação da implementação do simulador IndraNet, que se limita a agentes com a mesma estrutura de rede interna e com focos e escopos iguais;

(d) A implementação sequencial do modelo, limitando-se a capacidade de representação de agentes e de seu funcionamento em paralelo.

Como desenvolvimento futuro do presente trabalho, vislumbram-se as seguintes possibilidades:

(a) Maior exploração das potencialidades do modelo nos exemplos escolhidos, variando-se, por exemplo, as frequências de comunicação, os focos e os escopos dos agentes, aproximando-se de situações mais realistas;

(b) Aplicação do modelo a outras áreas de investigação nas quais o fenômeno da aprendizagem social está presente;

(c) Aprimoramento do simulador IndraNet, com a paralelização do algoritmo e a sua implementação em máquinas paralelas;

(d) Investigação sobre a capacidade preditiva do modelo proposto, principalmente o seu potencial uso no acompanhamento da emergência de tendências na *Web* social;

(e) Uso de *exponential random graphs* [60] como formalismo para exploração das propriedades matemáticas do modelo de redes epistêmicas auto-organizáveis.

No quadro a seguir, como contribuição final, apresenta-se um resumo das principais áreas potenciais para futura exploração do modelo proposto, com indicação de possíveis propriedades emergentes a serem observadas.

Concluindo, faço minhas as palavras de Martin Luther King, Jr.:

> *“Nós estamos presos em uma inescapável rede de mutualidade. O que afeta um, diretamente, afeta todos, indiretamente”.*

Área de aplicação	Agentes Epistêmicos	Pares Epistêmicos	Ws	Propriedades Emergentes
Sistemas Imunológicos [3, 40, 43, 71]	Linfócitos	Antígeno e resposta ao antígeno	Semelhança entre linfócitos	Formação de clones linfocitários
Comunidade de Cientistas [42, 46, 55]	Cientistas	Experimentos científicos	Proximidade dos resultados	Formação de paradigmas científicos
Propagação de Doenças [52, 71]	Indivíduos	Vírus e resposta ao vírus	Reações imunológicas entre indivíduos	Dinâmica das epidemias
Comunidades de Aprendizagem	Aprendizes	Crenças, conhecimentos individuais	Conhecimento coletivo	*Clusters* de conhecimento comum
Contágio Social [4, 73, 74]	Indivíduos	Ideias, modas	Influência social entre indivíduos	Disseminação de ideias
Web Social	Usuários	Comunidades a que pertencem	Importância dos amigos	*Clusters* de interesses
Formação de Consenso [19, 31]	Votantes	Opiniões	Influência entre votantes	Consenso, formação de blocos

Figura 7.1 – Tabela de Possíveis Aplicações do Modelo Proposto

Referências Bibliográficas

[1] ANG, C.S., ZAPHIRIS, P., “Simulating Social Network of Online Communities: Simulation as a Method for Sociability Design”, *Human-Computer Interaction – Interact 2009*;

[2] ATLAN, H., *Entre o Cristal e a Fumaça: Ensaio sobre a organização do ser vivo*, Jorge Zahar Editora, 1992;

[3] ATLAN, H., Cohen, I.R., “Immune information, self-organization and meaning”, *International Immunology*, Vol. 10, n. 6, pp. 711-717, 1998;

[4] AXELROD, R., “The Dissemination of Culture: A Model with local convergence and global polarization”, *J. Conflict Resolut.*, Vol. 41, pp. 203-226, 1997;

[5] BANDURA, A., *Social Learning through Imitation*, University of Nebrasca Press, 1962;

[6] BANDURA, A., AZZI, R.G., POLYDORO, S. (orgs), *Teoria Social Cognitiva*, Artmed, 2008;

[7] BANERJEE, A. V., “A Simple Model of Herd Behavior”, *Quarterly Journal of Economics*, Vol. 107, pp. 797-817, 1992;

[8] BARABÁSI, A.-L., ALBERT, R., “Emergence of Scaling in Random Networks”, *Science*, Vol. 286, pp. 509-512, 1999;

[9] BARABÁSI, A.-L., JEONG, H. *et al*, “Evolution of the Social Network of Scientific Collaborations”, *Physica A*, Vol. 311, pp. 590-614, 2002;

[10] BARABÁSI, A.-L., *Linked: How Everything Is Connected To Everything Else And What It Means For Business, Science and Everyday Life*, Penguin, 2002;

[11] BARABÁSI, A.-L., *Bursts: The Hidden Pattern Behind Everything We Do*, Penguin, 2010;

[12] BARRAT, A., BARTHÉLEMY, M., VESPIGNANI, A., *Dynamical Processes on Complex Networks*, Cambridge University Press, 2008;

[13] BAR-YAM, Y., *Dynamics of Complex Systems*, Addison-Wesley, 1997;

[14] BOCCALETTI, S. et al, "Complex Networks: Structure and Dynamics", *Physics Reports*, Vol. 424, pp. 175-308, 2006;

[15] BORGATTI, S.P., MEHRA, A., Brass, D.J., LABLANCA, G., "Network Analysis in the Social Sciences", *Science*, vol. 323, pp. 892-895, 2009;

[16] BUBER, M. *I and Thou,* Simon and Schuster, 1970;

[17] CHOI, S., GALE, D., KARIV, S., "Behavioral Aspects of Learning in Social Networks: An Experimental Study", in *Advances in Behavioral and Experimental Economics*, John Morgan (editor), JAI Press, 2005;

[18] COSTA, L. da F. *et al*., "Analysing and Modeling Real-World Phenomena with Complex Networks: A Survey of Applications", arXiv.org > physics > arXiv:0711.3199v3, [physics.soc-ph], 2008;

[19] DEGROOT, M. H., "Reaching a Consensus", *Journal of the American Statistical Association*, Vol. 69, No. 345, pp. 118-121, 1974;

[20] EASLEY, D., KLEINBERG, J., *Networks, Crowds and Markets: Reasoning About a Highly Connected World*, Cambridge University Press, 2010;

[21] EDELMAN, G.M., *Bright Air, Brilliant Fire: On The Matter of the Mind*, Basic Books, 1982;

[22] EDELMAN, G.M., *Wider Than The Sky: The Phenomenal Gift of Consciousness*, Yale University Press, 2004;

[23] ERDOS, P., RÉNYI, A. “On Random Graphs”, *Publicationes Mathematicae*, Vol. 6, pp. 290-297, 1959;

[24] FINKELSTEIN, A., GOEDICKE, M., KRAMER, J., NISKIER, C., "ViewPoint oriented software development: methods and viewpoints in requirements engineering", *Algebraic methods II: Theory, Tools and Applications* , Springer-Verlag, 1991;

[25] FINKELSTEIN, A., FUKS, H., NISKIER, C., SADLER, M., “Constructing a dialogic framework for software development”, ACM *SIGSOFT Software Engineering Notes*, v. 14 n.4, p.68-72, June, 1989;

[26] FOUCAULT. M., *A Arqueologia do Saber*, Ed. Forense, 2002;

[27] FREEMAN, L. C., *The Development of Social Network Analysis: A Study In The Sociology of Science*, Empirical Press, 2004;

[28] GALE, D., KARIV, S., “Bayesian Learning in Social Networks”, Games and Economic Behavior, Vol. 45 (2) , pp. 329-346, 2003;

[29] GALEF Jr., LALAND, K. N., “Social Learning in Animals: Empirical Studies and Theoretical Models”, *BioScience*, Vol. 55 (6), pp. 489-499, 2005;

[30] GLADWELL, M., *The Tipping Point: How Little Things Can Make A Big Difference*, Little, Brown, New York, 2000;

[31] GOLUB, B., JACKSON, M. O., “Naïve Learning in Social Networks and the Wisdom of Crowds”, *American Economic Journal: Microeconomics*, Vol. 2, Num. 1, pp. 112-149, 2010;

[32] GONÇALVES, M.S., CLAIR, E.S., “Antes tarde do que nunca: notas sobre as contribuições de Gabriel Tarde para a análise da articulação entre comunicação e cultura”, *Revista Galáxia,* nº 14, pp. 137-148, 2007;

[33] GOYAL, S., “Learning in Networks: A Survey”, in *Group Formation in Economics: Networks, Clubs, and Coalitions,* G. DEMANGE e M. WOODERS (eds), Cambridge University Press (2005);

[34] GRANOVETTER, M. S., “The Strength of Weak Ties”, *American Journal of Sociology*, Vol. 78, pp. 1360-1380, 1973;

[35] HEDSTROM, P., “Experimental Macro Sociology: Predicting the Next Best Seller”, *Science*, Vol. 311, pp. 786-787, 2006;

[36] HEDSTROM, P., “Rational Imitation”, in *Social Mechanisms*, Peter Hedstrom and Richard Swedberg, eds., Cambridge University Press, 2008;

[37] HOLLAND, J. H., *Emergence*, Basic Books, Perseus Publishing, 1998;

[38] HOLLAND, J. H., *Hidden Order: How Adaption Builds Complexity*, Perseus Books, 1995;

[39] *Indra´s Net*, disponível em: http://www.cs.kent.ac.uk/people/staff/saf/networks/networking-networkers/indras-net.html acesso em: 10/09/2010;

[40] JERNE, N.K., “Towards a network theory of the immune system”, *Ann. Immunol (Inst. Pasteur)*, v. 125 C, pp. 373-389, 1974;

[41] KOVÁCS, A., UENO, H., “Is the immune system an interactive system?”, draft abstract (to be presented at the Interactivist Summer Institute), 2005.

[42] KUHN, T.S., *The Structure of Scientific Revolutions*, Second Edition, Univ. of Chicago Press, 1970;

[43] LALAND, K. N., “Social Learning Strategies”, *Learning & Behavior*, Vol. 32 (1), pp. 4-14, 2004;

[44] MCCLELLAND, J.L. & RUMELHART, D.E. (eds), *Parallel Distributed Processing*, The MIT Press, 1986;

[45] MILGRAM, S., “The Small World Problem”, *Psychology Today*, Vol. 2, pp. 60-67, 1967;

[46] NEWMAN, M. E. J., “The Structure of Scientific Collaboration Networks”, *Proc. Natl. Acad. Sci. USA*, Vol. 98, pp. 404-409, 2001;

[47] NEWMAN, M. E. J., BARABÁSI, A.-L, Watts, D. J., *The Structure and Dynamics of Networks*, Princeton University Press, 2006;

[48] NEWMAN, M. E. J., *Networks: An Introduction*, Oxford University Press, 2010;

[49] NISKIER, C., *PRISMA: Utilização de Paradigmas Complementares para a Aquisição da Especificação de Software*, Dissertação de Mestrado, Depto. de Informática, PUC/RJ, 1986;

[50] NISKIER, C., MAIBAUM, T.S.E., SCHWABE, D., “A Look Through PRISMA: towards pluralistic knowledge-based environments for software specification acquisition”, *ACM SIGSOFT Software Engineering Notes*, Vol. 14(3), pp. 128-136, 1989;

[51] NISKIER, C., MAIBAUM, T.S.E., SCHWABE, D., “A Pluralistic Knowledge-Based Approach to Software Specification”, *Lecture Notes In*

Computer Science, Vol. 387 (Proceedings of the 2nd European Software Engineering Conference), 1989;

[52] PASTOR-SATORRAS, R., VESPIGNANI, A., "Epidemic Spreading in Scale-Free Networks", *Physical Review Letters*, Vol. 86, pp. 3200-3203, 2001;

[53] PEREIRA, L. S., *Modelo Matemático-Computacional da Gênese da Rede Idiotípica do Sistema Imunológico*, Dissertação de Mestrado, COPPE, 2006;

[54] POLANYI, M., *A Lógica da Liberdade*, Topbooks, 2003;

[55] POPPER, K., *A Lógica da Pesquisa Científica*, Editora Cultrix, 1972;

[56] ROTH, C., "Co-evolution in Epistemic Networks: Reconstructing Social Complex Systems", *Structure and Dynamics: eJournal of Anthropological and Related Sciences*, Vol. 1(3), 2007;

[57] RUELLE, D., *Acaso e Caos*, Editora Unesp, 1993;

[58] RUSSEL, R. & NORVIG, P., *Inteligência Artificial*, Elsevier Editora Ltda., 2004;

[59] SHOHAM, Y., Powers, R., GRENAGER, T., "If Multi-Agent Learning Is The Answer, What Is The Question?", *Artificial Intelligence*, Vol. 171, pp. 365-377, 2007;

[60] SNILDERS, T.A.B., PATTISON, P.E. *et al*, "New Specifications for Exponential Random Graph Models", *Sociological Methodology*, Vol. 36(1), pp. 99-153, 2006;

[61] *Social and Cognitive Networks Academic Research Center*, disponível em: http://scnarc.rpi.edu/ acesso em: 11/08/2010;

[62] “Special Issue on Social Learning”, *IEEE Intelligent Systems*, Vol. 25, nº4, Julho/Agosto 2010;

[63] “Special Issue on Learning and the Social Web”, *Journal of Emerging Technologies en Web Intelligence*, Vol. 2, nº1, 2010;

[64] “Special Issue on Computational Models of Collective Intelligence in the Social Web”, *ACM Transactions on Intelligent Systems and Technology*, prevista para Abril/Maio de 2011;

[65] “Special Issue on Complex Systems and Networks”, *Science*, Vol. 325, Issue 5939, pp. 357 – 504, July, 2009;

[66] TARDE, G., *A Opinião e as Massas*, Martins Fontes Editora, São Paulo, 2005;

[67] TARDE, G., *Les Lois de l´imitation*, Les Empécheurs de Penser em Rond/Éditions Du Seuil, Paris, 2001;

[68] TARDE, G., *Monadologia e Sociologia*, Cosac Naify, 2007;

[69] VIDAL DE CARVALHO, L.A., *Datamining*, Editora Ciência Moderna, 2005;

[70] VIDAL DE CARVALHO, L.A., *Síntese de Redes Neuronais com Aplicações à Representação do Conhecimento e à Otimização*, Tese de Doutorado, COPPE/UFRJ, 1989;

[71] VIDAL DE CARVALHO, L.A., BARBOSA, V.C., AGUILAR, E., FLORES, L., “A Graph Model for the Evolution of Specificity in Humeral Immunity*”, Journal of Theoretical Biology*, v. 229, pp 311-325, 2004;

[72] WATTS, D. J., *Small Worlds: The Dynamics of Networks Between Order and Randomness*, Princeton Studies in Complexity, Princeton University Press, 1999;

[73] WATTS, D.J., *Six Degrees – The Science of a Connected Age*, W.W.Norton, 2003;

[74] WATTS, D.J., “The ‘New’ Science of Networks”, *Annu. Rev. Sociol*., Vol. 30, pp. 243-270, 2004;

[75] WATTS, D.J., “A Twenty-First Century Science”, *Nature*, Vol. 445, pp. 489, Fev. 2007;

[76] WATTS, D. J., STROGATZ, S. H., “Collective Dynamics of ‘Small-World’ Networks”, *Nature*, Vol. 393, pp. 440-442, 1998;

[77] WEISS, G. (ed), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 1999;

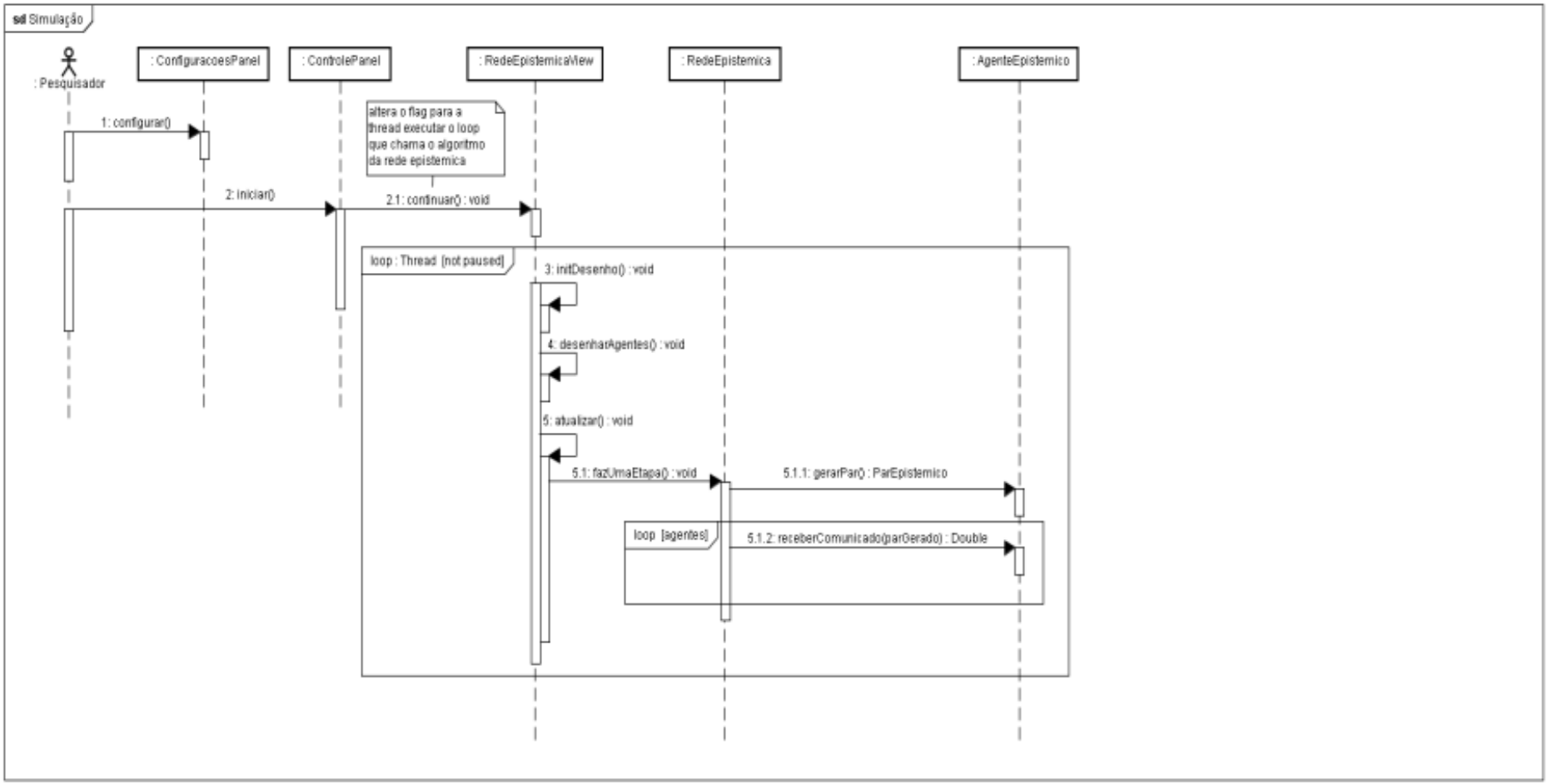
[78] WOOLDRIDGE, M., *An Introduction to MultiAgent Systems*, John Wiley & Sons, 2002.

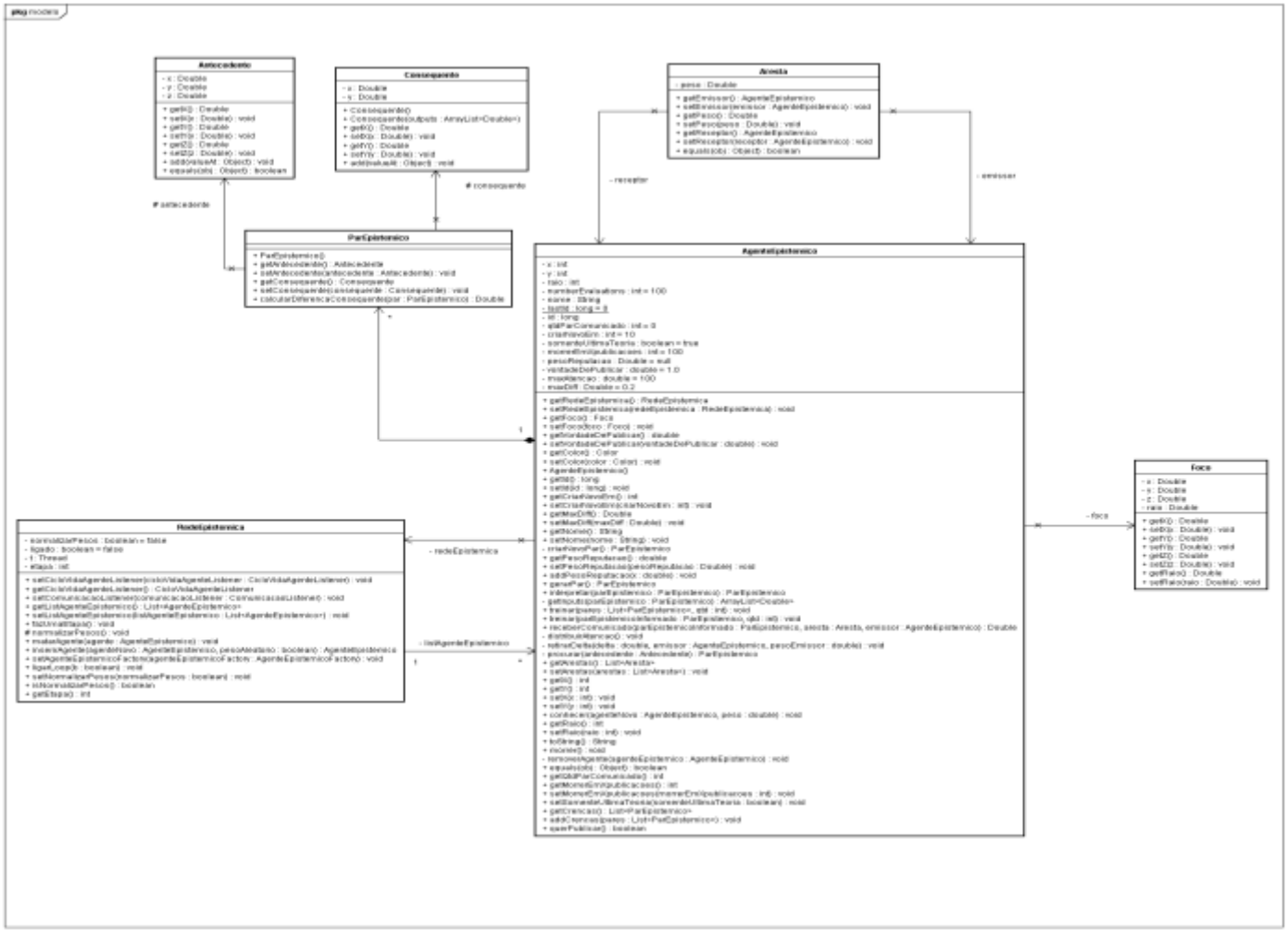
Anexo A

Documentação Técnica do Simulador Indranet 1.2

Anexo A.1

Diagramas de Sequência e de Classes/Objetos


sd Simulação
: Pesquisador
: ConfiguracoesPanel
: ControlePanel
: RedeEpistemicaView
: RedeEpistemica
: AgenteEpistemico
1: configurar()
altera o flag para a thread executar o loop que chama o algoritmo da rede epistemica
2: iniciar()
2.1: continuar() : void
loop : Thread [not paused]
3: initDesenho() : void
4: desenharAgentes() : void
5: atualizar() : void
5.1: fazUmaEtapa() : void
5.1.1: gerarPar() : ParEpistemico
loop [agentes]
5.1.2: receberComunicado(parGerado) : Double


pkg models
Antecedente
- x : Double
- y : Double
- z : Double
+ getX() : Double
+ setX(x : Double) : void
+ getY() : Double
+ setY(x : Double) : void
+ getZ() : Double
+ setZ(z : Double) : void
+ addValueM : Object() : void
+ equals(obj : Object) : boolean
Consequente
- x : Double
- y : Double
+ Consequente()
+ Consequente(outputs : ArrayList<Double>)
+ getX() : Double
+ setX(x : Double) : void
+ getY() : Double
+ setY(y : Double) : void
+ add(value : Object) : void
Aresta
- peso : Double
+ getEmissor() : AgenteEpistemico
+ setEmissor(emissor : AgenteEpistemico) : void
+ getPeso() : Double
+ setPeso(peso : Double) : void
+ getReceptor() : AgenteEpistemico
+ setReceptor(receptor : AgenteEpistemico) : void
+ equals(obj : Object) : boolean
# antecedente
# consequente
- receptor
- emissor
ParEpistemico
+ ParEpistemico()
+ getAntecedente() : Antecedente
+ setAntecedente(antecedente : Antecedente) : void
+ getConsequente() : Consequente
+ setConsequente(consequente : Consequente) : void
+ calcularDiferencaConsequente(par : ParEpistemico) : Double
AgenteEpistemico
- x : int
- y : int
- raio : int
- numberEvaluations : int = 100
- name : String
- nome : long = 0
- id : long
- qtdParComunicado : int = 0
- limiteVoo : int = 10
- somenteUltimaTeoria : boolean = true
- incrementoPublicacao : int = 100
- pesoRepulsao : Double = null
- vontadeDePublicar : double = 1.0
- msoAlcance : double = 100
- msaDiff : Double = 0.2
+ getRedeEpistemica() : RedeEpistemica
+ setRedeEpistemica(redeEpistemica : RedeEpistemica) : void
+ getFoco() : Foco
+ setFocodoco : Foco) : void
+ getVontadeDePublicar() : double
+ setVontadeDePublicar(vontadeDePublicar : double) : void
+ getColor() : Color
+ setColor(color : Color) : void
+ AgenteEpistemico()
+ getId() : long
+ setId(id : long) : void
+ getCriarNovoPar() : int
+ setCriarNovoMinAclaraNovoMin : int) : void
+ getMaxDiff() : Double
+ setMaxDiff(maxDiff : Double) : void
+ getNome() : String
+ setNome(nome : String) : void
+ criarNovoPar() : ParEpistemico
+ getPesoRepulsao() : double
+ setPesoRepulsao(pesoRepulsao : Double) : void
+ addPesoRepulsacao(: double) : void
+ gerarPar() : ParEpistemico
+ interpretar(parEpistemico : ParEpistemico) : ParEpistemico
+ getNovoEpistemico : ParEpistemico) : ArrayList<Double>
+ treinar(passo : List<ParEpistemico>, qtd : int) : void
+ lembrar(parEpistemicoAprendido : ParEpistemico, qtd : int) : void
+ receberComunicado(parEpistemicoEnviado : ParEpistemico, aresta : Aresta, emissor : AgenteEpistemico) : Double
- distribuirAlcance() : void
- refinarCrencas(delta : double, emissor : AgenteEpistemico, pesoEmissor : double) : void
- procurar(antecedente : Antecedente) : ParEpistemico
+ getArestas() : List<Aresta>
+ setArestas(arestas : List<Aresta>) : void
+ getX() : int
+ getY() : int
+ setX(x : int) : void
+ setY(y : int) : void
+ conectar(agenteNovo : AgenteEpistemico, peso : double) : void
+ getRaio() : int
+ setRaio(raio : int) : void
+ toString() : String
+ ricochete() : void
- removerAgente(agenteEpistemico : AgenteEpistemico) : void
+ equals(obj : Object) : boolean
+ getQtdParComunicado() : int
+ getMomentoPublicacao() : int
+ setMomentoPublicacao(momentoPublicacao : int) : void
+ setSomenteUltimaTeoria(somenteUltimaTeoria : boolean) : void
+ getCrencas() : List<ParEpistemico>
+ addCrencas(pares : List<ParEpistemico>) : void
+ querPublicar() : boolean
Foco
- x : Double
- y : Double
- z : Double
- raio : Double
+ getX() : Double
+ setX(x : Double) : void
+ getY() : Double
+ setY(y : Double) : void
+ getZ() : Double
+ setZ(z : Double) : void
+ getRaio() : Double
+ setRaio(raio : Double) : void
RedeEpistemica
- somenteMaiorPeso : boolean = false
- ligado : boolean = false
- t : Thread
- atajo : int
+ setCicloVidaAgenteListener(cicloVidaAgenteListener : CicloVidaAgenteListener) : void
+ getCicloVidaAgenteListener() : CicloVidaAgenteListener
+ setComunicacaoListener(comunicacaoListener : ComunicacaoListener) : void
+ getListAgentesEpistemicos() : List<AgenteEpistemico>
+ setListAgentesEpistemicos(listAgenteEpistemico : List<AgenteEpistemico>) : void
+ fazUmaMagia() : void
# normalizarPesos() : void
+ matarAgente(agente : AgenteEpistemico) : void
+ inserirAgente(agenteNovo : AgenteEpistemico, pesoAleatorio : boolean) : AgenteEpistemico
+ setAgenteEpistemicoFactory(agenteEpistemicoFactory : AgenteEpistemicoFactory) : void
+ liguei_crepi() : boolean : void
+ setNormalizarPesos(normalizarPesos : boolean) : void
+ isNormalizarPesos() : boolean
+ getStatus() : int
- redeEpistemica
- listAgenteEpistemico
1
*
1
*
- foco

Anexo A.2

Programas

redesepistemicas/ bo/ BackPropagation.java

```
1   package br.unicarioca.redesepistemicas.bo;
2
3   import java.util.ArrayList;
4   import java.util.Iterator;
5   import java.util.ListIterator;
6
7   import org.apache.log4j.Logger;
8
9   import com.rmit.neuralnetwork.NeuralNetwork;
10  import com.rmit.neuralnetwork.activation.Activation;
11  import com.rmit.neuralnetwork.node.Connection;
12  import com.rmit.neuralnetwork.node.Node;
13  import com.rmit.neuralnetwork.node.Weight;
14  import com.rmit.neuralnetwork.training.Training;
15  import com.rmit.neuralnetwork.trainingdata.TrainingExample;
16  import com.rmit.neuralnetwork.trainingdata.TrainingSet;
17
18  /**
19   * @author lachlan
20   *
21   */
22  public class BackPropagation extends Training {
23      private static Logger logger = Logger.getLogger(BackPropagation.class);
24      private double learningRate;
25      private double momentum;
26
27      public BackPropagation(int numberEvaluations, double errorTolerance) {
28
29          super(numberEvaluations, errorTolerance);
30
31          this.learningRate = 0.9;
32          this.momentum = 0.001;
33
34          init();
35      }
36
37      public BackPropagation(int numberEvaluations, double errorTolerance, double learningRate, double
38  momentum) {
39
40          super(numberEvaluations, errorTolerance);
41
42          this.learningRate = learningRate;
43          this.momentum = momentum;
44
45          init();
46      }
47
48      private void init() {
49          setDetails("BACKPROP learningRate " + learningRate + " momentum " + momentum);
50      }
51
52      @Override
53      public ArrayList<Double> train(NeuralNetwork neuralNetwork, TrainingSet trainingSet) {
```

```
54
55              ArrayList<Double> errors = new ArrayList<Double>();
56              ArrayList<TrainingExample> trainingExamples = trainingSet.getTrainingExamples();
57              double error = Double.MAX_VALUE;
58              double errorBest = Double.MAX_VALUE;
59
60              int numberTrainingExamples = trainingExamples.size();
61
62              // reset the weights
63              //neuralNetwork.initWeights();
64
65              for (int current = 0; current < numberEvaluations && error > errorTolerance; current++) {
66
67                  error = 0;
68
69                  // go through each training example
70                  for (TrainingExample trainingExample : trainingExamples) {
71
72                      // pass in the new inputs to the nn
73                      neuralNetwork.setInputs(trainingExample.getInputs());
74
75                      neuralNetwork.getOutputs();
76
77                      // ******************* USE THIS FOR ONLINE ERROR ************************
78                      // get the predicted outputs + calculate outputs error
79                      // error += getError(trainingExample.getOutputs(), neuralNetwork.getOutputs());
80
81                      // update weights by back propagating errors
82                      backPropagate(neuralNetwork, trainingExample.getOutputs());
83                  }
84
85                  // ******************** THIS IS FOR OFFLINE ERROR ****************************
86                  // calculate offline error (real error)
87                  for (TrainingExample trainingExample : trainingExamples) {
88
89                      // pass in the new inputs to the nn
90                      neuralNetwork.setInputs(trainingExample.getInputs());
91
92                      // get the predicted outputs + calculate outputs error
93                      error += getError(trainingExample.getOutputs(), neuralNetwork.getOutputs());
94                  }
95                  // ***********************************************************************
96
97                  // get the mean of the summed squared error of one epoch
98                  error = error/numberTrainingExamples;
99
100                 // get best error
101                 if(error < errorBest) {
102                     errorBest = error;
103                 }
104
105                 // save the errors for output
106                 errors.add(errorBest);
107             }
108
```

```
109                 logger.debug("BACKPROP " + errorBest);
110                 return errors;
111             }
112
113             private void backPropagate(NeuralNetwork neuralNetwork, ArrayList<Double> trainingSetOutputs) {
114
115                 double value = 0.0;
116                 double delta = 0.0;
117                 double error = 0.0;
118                 double errorGradient = 0.0;
119
120                 Iterator<Double> trainingSetOutputsIterator = trainingSetOutputs.iterator();
121                 Activation activation = neuralNetwork.getActivation();
122
123                 // iterator backwards through the neural networks graph
124                 for (ListIterator<Node> graphIterator = neuralNetwork.listIteratorEnd();
125         graphIterator.hasPrevious();) {
126
127                     Node node = graphIterator.previous();
128                     value = node.getValue();
129
130                     if (node.getType() == Node.OUTPUT) {
131
132                         error = getErrorOutput(trainingSetOutputsIterator.next(), value);
133
134                     } else if (node.getType() == Node.HIDDEN) {
135
136                         error = getErrorHidden(value, node.getOutputConnections());
137                     }
138
139                     // error gradient
140                     errorGradient = error * activation.getGradient(value);
141
142                     // update the gradient so the children can use next
143                     node.setErrorGradient(errorGradient);
144
145                     for (Connection inputConnection : node.getInputConnections()) {
146
147                         Weight weight = inputConnection.getWeight();
148
149                         delta = learningRate * inputConnection.getNode().getValue() * errorGradient +
150         momentum * weight.getDelta();
151
152                         // update the weight and save the delta
153                         weight.updateWeight(delta);
154                     }
155                 }
156             }
157
158             private double getErrorOutput(double outputTrainingSet, double outputNeuralNetwork){
159
160                 return (outputTrainingSet - outputNeuralNetwork);
161             }
162
163             private double getErrorHidden(double nodeValue, ArrayList<Connection> outputConnections){
```

```
164
165                 double sum = 0.0;
166
167                 for (Connection outputConnection : outputConnections) {
168
169                       sum += outputConnection.getWeight().getWeight() *
170         outputConnection.getNode().getErrorGradient();
                    }
                    return sum;
              }
         }
```

bo/ Demo.java

```
1  /**
2   * Copyright (c) 2009, lachlan.gregor@gmail.com
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without modification, are
6   * permitted provided that the following conditions are met: Redistributions of source code
7   * must retain the above copyright notice, this list of conditions and the following disclaimer.
8   * Redistributions in binary form must reproduce the above copyright notice, this list of
9   * conditions and the following disclaimer in the documentation and/or other materials
10  * provided with the distribution. Neither the name of the RMIT University Ð Melbourne, Australia
11  * nor the names of its contributors may be used to endorse or promote products derived from this
12  * software without specific prior written permission.
13  *
14  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
15  ANY EXPRESS
16  * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
17  MERCHANTABILITY
18  * AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
19  COPYRIGHT HOLDER
20  * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
21  OR
22  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
23  GOODS
24  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
25  CAUSED AND ON
26  * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
27  NEGLIGENCE
28  * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
29  OF THE
30  * POSSIBILITY OF SUCH DAMAGE.
31  */
32  package br.unicarioca.redesepistemicas.bo;
33
34  import java.util.ArrayList;
35
36  import com.rmit.neuralnetwork.NeuralNetwork;
37  import com.rmit.neuralnetwork.training.BackPropagation;
38  import com.rmit.neuralnetwork.training.Training;
39  import com.rmit.neuralnetwork.trainingdata.TrainingExample;
40  import com.rmit.neuralnetwork.trainingdata.TrainingSet;
41
42  /**
43   * @author lachlan
44   *
45   */
46  public class Demo {
47
48      private NeuralNetwork neuralNetwork;
49
50      public Demo() {
51
52          // define the structure of the neural network
53          int [] neuralNetworkStructure = new int[]{3, 1, 2};
```

```
54
55              // create new neural network with the defined structure
56              neuralNetwork = new NeuralNetwork(neuralNetworkStructure);
57
58              // training settings
59              //int numberEvaluations = 3000;
60              int numberEvaluations = 1000;
61              double errorTolerance = 0.0;
62
63              // pso specific settings
64              int numberParticles = 20;
65              double learningFactor = 1.49618;
66              double inertialWeight = 0.7298;
67
68              // create a instance of a training method
69              //Training training = new ParticleSwarmOptimization(numberEvaluations, errorTolerance,
70  learningFactor, inertialWeight, numberParticles);
71              Training training = new BackPropagation(numberEvaluations,errorTolerance);
72
73              // get the training set
74              //DataLoader trainingData = new DataLoader(DataLoader.CLASSIFICATION,
75  "../neuralnetworkdata/xor.csv", 2, 1, 100, 0, 1);
76
77              ArrayList<TrainingExample> listTraining = new ArrayList<TrainingExample>();
78              {
79                    TrainingExample te = new TrainingExample();
80                    ArrayList<Double> in = new ArrayList<Double>();
81                    in.add(1.0);
82                    in.add(1.0);
83                    in.add(1.0);
84                    ArrayList<Double> out = new ArrayList<Double>();
85                    out.add(0.7);
86                    out.add(0.7);
87                    te.setInputs(in);
88                    te.setOutputs(out);
89                    listTraining.add(te);
90              }
91              {
92                    TrainingExample te = new TrainingExample();
93                    ArrayList<Double> in = new ArrayList<Double>();
94                    in.add(0.5);
95                    in.add(1.0);
96                    in.add(1.0);
97                    ArrayList<Double> out = new ArrayList<Double>();
98                    out.add(0.3);
99                    out.add(0.3);
100                   te.setInputs(in);
101                   te.setOutputs(out);
102                   listTraining.add(te);
103             }
104             {
105                   TrainingExample te = new TrainingExample();
106                   ArrayList<Double> in = new ArrayList<Double>();
107                   in.add(0.5);
108                   in.add(0.5);
```

```
109                     in.add(1.0);
110                     ArrayList<Double> out = new ArrayList<Double>();
111                     out.add(0.9);
112                     out.add(0.9);
113                     te.setInputs(in);
114                     te.setOutputs(out);
115                     listTraining.add(te);
116             }
117             TrainingSet trainingSet = new TrainingSet("MyT",listTraining);
118
119
120             // set the training method and set for the neural network to use
121             neuralNetwork.setTraining(training);
122             //neuralNetwork.setTrainingSet(trainingData.getTrainingSet());
123             neuralNetwork.setTrainingSet(trainingSet);
124
125     }
126
127     private void train() {
128             neuralNetwork.train();
129     }
130
131     private void predict(Double a,Double b) {
132             ArrayList<Double> inputs = new ArrayList<Double>();
133             ArrayList<Double> outputs = new ArrayList<Double>();
134
135             inputs.add(a);
136             inputs.add(b);
137             inputs.add(1.0);
138             System.out.println("inputs "+a+" "+b);
139
140             neuralNetwork.setInputs(inputs);
141             outputs = neuralNetwork.getOutputs();
142
143             for (Double output : outputs) {
144                     System.out.println("predicted output " + output);
145             }
146     }
147
148     public static void main(String[] args) {
149
150             System.out.println("starting ...");
151             Demo main = new Demo();
152
153             System.out.println("training neural network ...");
154             main.train();
155
                System.out.println("predicting neural network ...");
                main.predict(1.0,1.0);
                System.out.println("predicting neural network ...");
                main.predict(0.5,1.0);
                System.out.println("predicting neural network ...");
                main.predict(0.5,0.5);

                System.out.println("finished successfully !");
```

```
    }
}
```

bo/ GenericDao.java

```
1  package br.unicarioca.redesepistemicas.bo;
2
3  import java.io.File;
4  import java.io.FileInputStream;
5  import java.io.FileNotFoundException;
6  import java.io.FileOutputStream;
7  import java.io.IOException;
8  import java.io.ObjectInputStream;
9  import java.io.ObjectOutputStream;
10 import java.text.SimpleDateFormat;
11 import java.util.ArrayList;
12 import java.util.Collections;
13 import java.util.Date;
14
15 public class GenericDao<T> {
16     public void salvar(T obj, String nome) throws IOException{
17         String pasta = obj.getClass().getSimpleName();
18         File dir = new File(pasta);
19         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd-HH-mm-ss");
20         String data = sdf.format(new Date());
21         if(!dir.exists()) dir.mkdir();
22         File arq = new File(dir,data + "-" + nome);
23
24         FileOutputStream fos = null;
25         ObjectOutputStream out = null;
26         try {
27             fos = new FileOutputStream(arq);
28             out = new ObjectOutputStream(fos);
29             out.writeObject(obj);
30             out.close();
31         } catch (IOException ex) {
32             ex.printStackTrace();
33         }
34     }
35
36     @SuppressWarnings("unchecked")
37     public T getLast(Class<?> clazz){
38         String pasta = clazz.getSimpleName();
39         File dir = new File(pasta);
40         if(!dir.exists()){
41             return null;
42         }
43         ArrayList<String> lista = new ArrayList<String>();
44         String list[] = dir.list();
45         if(list==null || list.length==0){
46             return null;
47         }
48         for(int i=0;i<list.length;i++){
49             lista.add(list[i]);
50         }
51         Collections.sort(lista);
52         String fileName = lista.get(lista.size()-1);
53
```

```
54              File file = new File(dir,fileName);
55              FileInputStream fis = null;
56              ObjectInputStream in = null;
57              try {
58                    fis = new FileInputStream(file);
59                    in = new ObjectInputStream(fis);
60                    return (T) in.readObject();
61              }catch(Exception e){
62                    e.printStackTrace();
63                    return null;
64              }finally{
65                    try {
66                          in.close();
67                    } catch (IOException e) {
68                          e.printStackTrace();
69                    }
70              }
71        }
72  }
```

bo/ InfoListener.java

```
1  package br.unicarioca.redesepistemicas.bo;
2
3  /**
4   * Interface para desviar os INFO para a UI
5   */
6  public interface InfoListener {
7          public void info(String info);
8  }
```

bo/ Log4jAppender.java

```
1  package br.unicarioca.redesepistemicas.bo;
2
3  import org.apache.log4j.AppenderSkeleton;
4  import org.apache.log4j.spi.LoggingEvent;
5
6  import br.unicarioca.redesepistemicas.view.MainFrame;
7
8  /**
9   * Responsavel por desviar os INFO para a interface do usuario
10  */
11 public class Log4jAppender extends AppenderSkeleton{
12
13      @Override
14      protected void append(LoggingEvent event) {
15          String message = "";
16          if( this.layout == null ){
17              message = "Erro: Sem layout de mensagem";
18          }else{
19              message = this.layout.format(event);
20          }
21          //Enviar a mensagem para o MainFrame
22          InfoListener mf = MainFrame.getLastInstance();
23          if(mf!=null){
24              mf.info(message);
25          }
26      }
27
28      @Override
29      public void close() {
30
31      }
32
33      @Override
34      public boolean requiresLayout() {
35          return true;
36      }
37
38 }
```

bo/ QuickSort.java

```
1   package br.unicarioca.redesepistemicas.bo;
2
3   import java.util.Random;
4
5   import javax.swing.DefaultListModel;
6
7   import br.unicarioca.redesepistemicas.modelo.AgenteEpistemico;
8
9   public class QuickSort {
10     public static final Random RND = new Random();
11
12     private static  void swap(DefaultListModel array, int i, int j) {
13        Object tmp = array.get(i);
14        array.set(i, array.get(j));
15        array.set(j, tmp);
16     }
17
18     private static int partition(DefaultListModel array, int begin, int end) {
19        int index = begin + RND.nextInt(end - begin + 1);
20        AgenteEpistemico pivot = (AgenteEpistemico) array.get(index);
21        swap(array, index, end);
22        for (int i = index = begin; i < end; ++i) {
23           if
24  (((Double)((AgenteEpistemico)array.get(i)).getPesoReputacao()).compareTo(pivot.getPesoReputacao()) >=
25  0) {
26              swap(array, index++, i);
27           }
28        }
29        swap(array, index, end);
30        return (index);
31     }
32
33     private static void qsort(DefaultListModel array, int begin, int end) {
34        if (end > begin) {
35           int index = partition(array, begin, end);
36           qsort(array, begin, index - 1);
37           qsort(array, index + 1, end);
38        }
39     }
40
41     public static void sort(DefaultListModel array) {
42        qsort(array, 0, array.getSize() - 1);
     }
  }
```

bo/ SalvarSnapShoot.java

```
 1  package br.unicarioca.redesepistemicas.bo;
 2
 3  import java.awt.image.BufferedImage;
 4  import java.io.File;
 5  import java.io.IOException;
 6  import java.lang.reflect.Method;
 7
 8  import javax.imageio.ImageIO;
 9  import javax.swing.JCheckBox;
10  import javax.swing.JLabel;
11  import javax.swing.JSpinner;
12  import javax.swing.JTextField;
13
14  import org.apache.commons.io.FileUtils;
15
16  import br.unicarioca.redesepistemicas.view.ConfiguracoesPanel;
17
18  /**
19   * Responsavel por salvar o estado da aplicacao
20   *
21   */
22  public class SalvarSnapShoot {
23
24      private boolean pastasVerificadas = false;
25      private static SalvarSnapShoot instance;
26      private static int snapshotN=0;
27      private static int snapshotManN=0;
28      private ConfiguracoesPanel configuracoesPanel;
29
30      {
31          criaIndex();
32      }
33
34      private SalvarSnapShoot(){
35          pastasVerificadas = false;
36      }
37
38      public static SalvarSnapShoot getInstance() {
39          if(instance==null){
40              instance = new SalvarSnapShoot();
41          }
42          return instance;
43      }
44
45      /**
46       * Salva um screenshot
47       * @param bi
48       * @param n
49       */
50      public void salvar(BufferedImage bi,long n) {
51          try {
52              snapshotN++;
53              if(!pastasVerificadas){
```

```
54                          File dir = new File("screenshots/automatica");
55                          if(!dir.exists())dir.mkdir();
56                          dir = new File("screenshots/automatica/imagens");
57                          if(!dir.exists())dir.mkdir();
58                  }
59                  //salva o png
60                  ImageIO.write(bi, "png", new File("screenshots/automatica/imagens/f"+snapshotN+".png"));
61                  //salvar o html
62                  StringBuilder html=new StringBuilder();
63                  String configuracao = criarHtmlConf();
64                  html.append("<html><head><title>RedeEpistemica</title><script
65  src=\"../auto.js\"></script></head><body>");
66                  html.append("<div id=\"nav\">");
67
68                  if (snapshotN > 1)
69                          html.append("<a href=\"p").append(snapshotN-1).append(".html\">Anterior</a> - ");
70                  else
71                          html.append("Anterior - ");
72                  html.append("<a href=\"p").append(snapshotN).append(".html#autoplay\">auto</a> - ");
73                  html.append("<a href=\"p").append(snapshotN+1).append(".html\">Pr&oacute;xima</a>");
74                  html.append("</div>");
75                  html.append("<img src=\"imagens/f").append(snapshotN).append(".png\">");
76                  html.append("<div id=\"conf\">");
77                  html.append("Passo: ").append(n).append("<br />");
78                  html.append(configuracao);
79                  html.append("</div>");
80                  html.append("</body></html>");
81                  FileUtils.writeStringToFile(new File("screenshots/automatica/p"+snapshotN+".html"),
82  html.toString(), "iso-8859-1");
83          } catch (IOException e) {
84                  e.printStackTrace();
85          } catch (Exception e) {
86                  e.printStackTrace();
87          }
88      }
89
90      public void salvar(BufferedImage bi) {
91          try {
92                  snapshotManN++;
93                  if(!pastasVerificadas){
94                          File dir = new File("screenshots/manual");
95                          if(!dir.exists())dir.mkdir();
96                          dir = new File("screenshots/manual/imagens");
97                          if(!dir.exists())dir.mkdir();
98                  }
99                  //salva o png
100                 ImageIO.write(bi, "png", new File("screenshots/manual/imagens/m"+snapshotManN+".png"));
101                 //salvar o html
102                 StringBuilder html=new StringBuilder();
103                 String configuracao = criarHtmlConf();
104                 html.append("<html><head><title>RedeEpistemica</title></head><body>");
105                 html.append("<div id=\"nav\">");
106
107                 if (snapshotManN > 1)
108                         html.append("<a href=\"p").append(snapshotManN-1).append(".html\">Anterior</a> - ");
```

```
109                     else
110                         html.append("Anterior - ");
111
112
113                     html.append("<a
114 href=\"p").append(snapshotManN+1).append(".html\">Pr&oacute;xima</a>");
115                     html.append("</div>");
116                     html.append("<img src=\"imagens/m").append(snapshotManN).append(".png\">");
117                     html.append("<div id=\"conf\">");
118                     //html.append("Passo: ").append(n).append("<br />");
119                     html.append(configuracao);
120                     html.append("</div>");
121                     html.append("</body></html>");
122                     FileUtils.writeStringToFile(new File("screenshots/manual/p"+snapshotManN+".html"),
123 html.toString(), "iso-8859-1");
124             } catch (IOException e) {
125                 e.printStackTrace();
126             } catch (Exception e) {
127                 e.printStackTrace();
128             }
129         }
130
131         private String criarHtmlConf() throws Exception{
132             StringBuilder retorno=new StringBuilder();
133             Method metodos[] = configuracoesPanel.getClass().getMethods();
134             for(Method metodo : metodos){
135                 if(metodo.getName().startsWith("getTxt")){
136                     //pegar o label
137                     Method metodoLabel =
138 configuracoesPanel.getClass().getMethod(metodo.getName().replace("getTxt","getLbl"));
139                     JTextField obj = (JTextField)metodo.invoke(configuracoesPanel);
140                     JLabel label = (JLabel)metodoLabel.invoke(configuracoesPanel);
141                     retorno.append(label.getText());
142                     retorno.append(" ");
143                     retorno.append(obj.getText());
144                     retorno.append("<br />");
145                 }else if(metodo.getName().startsWith("getChk")){
146                     //pegar o label
147                     Method metodoLabel =
148 configuracoesPanel.getClass().getMethod(metodo.getName().replace("getChk","getLbl"));
149                     JCheckBox obj = (JCheckBox)metodo.invoke(configuracoesPanel);
150                     JLabel label = (JLabel)metodoLabel.invoke(configuracoesPanel);
151                     retorno.append(label.getText());
152                     retorno.append(" ");
153                     retorno.append(obj.isSelected());
154                     retorno.append("<br />");
155                 }else if(metodo.getName().startsWith("getSpn")){
156                     //pegar o label
157                     Method metodoLabel =
158 configuracoesPanel.getClass().getMethod(metodo.getName().replace("getSpn","getLbl"));
159                     JSpinner obj = (JSpinner)metodo.invoke(configuracoesPanel);
160                     JLabel label = (JLabel)metodoLabel.invoke(configuracoesPanel);
161                     retorno.append(label.getText());
162                     retorno.append(" ");
163                     retorno.append(obj.getValue());
```

```
164                         retorno.append("<br />");
165                     }
166                 }
167                 return retorno.toString();
168         }
169
170         private void criaIndex(){
171                 try{
172                         File dir = new File("screenshots");
173                         if(!dir.exists())dir.mkdir();
174
175                         //salvar o html
176                         StringBuilder html=new StringBuilder();
177
178                         html.append("<html><head><title>RedeEpistemica - ScreenShots</title></head><body>");
179                         html.append("<div id=\"nav\">");
180                         html.append("<h3>Screenshots</h3>");
181                         html.append("<ul>");
182                         html.append("<li /><a href=\"automatica/p1.html\">Screenshots Automáticas</a>");
183                         html.append("<li /><a href=\"manual/p1.html\">Screenshots Manual</a>");
184                         html.append("</ul>");
185                         html.append("</div>");
186                         html.append("</body></html>");
187                         FileUtils.writeStringToFile(new File("screenshots/index.html"), html.toString(), "iso-8859-1");
188
189                 } catch (IOException e) {
190                         e.printStackTrace();
191                 } catch (Exception e) {
192                         e.printStackTrace();
193                 }
194
195         }

        public ConfiguracoesPanel getConfiguracoesPanel() {
                return configuracoesPanel;
        }
        public void setConfiguracoesPanel(ConfiguracoesPanel configuracoesPanel) {
                this.configuracoesPanel = configuracoesPanel;
        }
}
```

dao/ AgenteDao.java

```
1  package br.unicarioca.redesepistemicas.dao;
2
3  import br.unicarioca.redesepistemicas.modelo.AgenteEpistemico;
4  import br.unicarioca.redesepistemicas.modelo.Aresta;
5  import br.unicarioca.redesepistemicas.modelo.ParEpistemico;
6
7  public class AgenteDao {
8      public static final String TEMPLATE = "<agente aid=\"{aid}\" y=\"{y}\" x=\"{x}\" maxDif=\"{maxDif}\"
9  morrerEm=\"{morrerEm}\" somenteUltimo=\"{somenteUltimo}\" criarNovoEm=\"{criarNovoEm}\" freq=\"{freq}\"
10 raio=\"{raio}\">";
11     public static String getXml(AgenteEpistemico agente){
12         String header = TEMPLATE
13             .replace("{aid}", agente.getId()+"")
14             .replace("{x}", agente.getX()+"")
15             .replace("{y}", agente.getY()+"")
16             .replace("{maxDif}", agente.getMaxDiff()+"")
17             .replace("{morrerEm}", agente.getMorrerEmXpublicacoes()+"")
18             .replace("{somenteUltimo}", agente.isSomenteUltimaTeoria()? "1" : "0")
19             .replace("{criarNovoEm}", agente.getCriarNovoEm()+ "")
20             .replace("{freq}", agente.getVontadeDePublicar()+"")
21             .replace("{raio}", agente.getRaio()+"");
22
23         StringBuilder sb = new StringBuilder();
24         sb.append(header);
25         for(ParEpistemico par:agente.getCrencas()){
26             sb.append("<crencaref ref=\"");
27             sb.append(par.getId());
28             sb.append("\" qtd=\"1000\" />");
29         }
30         for(Aresta aresta: agente.getArestas()){
31             sb.append("<aresta aid=\"");
32             sb.append(aresta.getReceptor().getId());
33             sb.append("\" peso=\"");
34             sb.append(aresta.getPeso());
35             sb.append("\" />");
36         }
37         sb.append("</agente>");
38         return sb.toString();
       }
   }
```

dao/ CrencaDao.java

```
1  package br.unicarioca.redesepistemicas.dao;
2
3  import br.unicarioca.redesepistemicas.modelo.ParEpistemico;
4
5  public class CrencaDao {
6      public static String getXml(ParEpistemico parEpistemico){
7          StringBuilder a = new StringBuilder();
8          for(Double d:parEpistemico.getDoubleAntecedentes()){
9              a.append(";").append(d);
10         }
11         StringBuilder c = new StringBuilder();
12         for(Double d:parEpistemico.getDoubleConsequentes()){
13             c.append(";").append(d);
14         }
15         if(c.length()==0) c.append(";");
16         if(a.length()==0) a.append(";");
17         System.out.println("a " + a.toString());
18         System.out.println("c " + c.toString());
19         String color = "FFFFFF";
20         if(parEpistemico.getCor()!=null){
21             color = Integer.toHexString(parEpistemico.getCor().getRGB()).toUpperCase().substring(2);
22         }
23         return "<crenca a=\"{a}\" c=\"{c}\" cid=\"{cid}\" nome=\"{nome}\" color=\"{color}\" />"
24             .replace("{a}", a.toString().substring(1))
25             .replace("{c}", c.toString().substring(1))
26             .replace("{cid}", parEpistemico.getId()+"")
27             .replace("{nome}", parEpistemico.toString())
28             .replace("{color}", color);
29
30     }
31 }
```

dao/ RedeDao.java

```
1   package br.unicarioca.redesepistemicas.dao;
2
3   import java.awt.Color;
4   import java.io.File;
5   import java.util.ArrayList;
6   import java.util.HashMap;
7   import java.util.List;
8   import java.util.Map;
9
10  import javax.xml.parsers.DocumentBuilder;
11  import javax.xml.parsers.DocumentBuilderFactory;
12
13  import org.apache.log4j.Logger;
14  import org.w3c.dom.Document;
15  import org.w3c.dom.Element;
16  import org.w3c.dom.NamedNodeMap;
17  import org.w3c.dom.Node;
18  import org.w3c.dom.NodeList;
19
20  import br.unicarioca.redesepistemicas.modelo.AgenteEpistemico;
21  import br.unicarioca.redesepistemicas.modelo.ParEpistemico;
22  import br.unicarioca.redesepistemicas.modelo.ParEpistemicoOrkut;
23  import br.unicarioca.redesepistemicas.modelo.RedeEpistemica;
24  import br.unicarioca.redesepistemicas.view.ConfiguracoesPanel;
25  import br.unicarioca.redesepistemicas.view.RedeEpistemicaView;
26
27  public class RedeDao {
28      private static final Logger logger = Logger.getLogger(RedeDao.class);
29      public static void loadFromXml(ConfiguracoesPanel configuracoesPanel, RedeEpistemica
30  redeEpistemica, RedeEpistemicaView redeEpistemicaView, File xmlFile) {
31          try {
32              logger.info("Abrindo xml " + xmlFile.getAbsolutePath());
33              DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
34              dbf.setNamespaceAware(false);
35              DocumentBuilder docBuilder;
36              docBuilder = dbf.newDocumentBuilder();
37              Document doc = docBuilder.parse(xmlFile);
38              Element redeTag = doc.getDocumentElement();
39              String nnsStr[] = redeTag.getAttribute("neuralNetworkStructure").split(",");
40              boolean distrAleatoria = redeTag.getAttribute("distrAleatoria").equals("s");
41              int nns[] = new int[nnsStr.length];
42              for(int i=0;i<nns.length;i++){
43                  nns[i] = Integer.parseInt(nnsStr[i].trim());
44              }
45
46  configuracoesPanel.getTxtEstruturaRede().setText(redeTag.getAttribute("neuralNetworkStructure"));
47
48  configuracoesPanel.getSpnPassoMax().setValue(Integer.valueOf(redeTag.getAttribute("passoAgente")));
49              configuracoesPanel.getChkDistribuicaoAleatoria().setSelected(distrAleatoria);
50
51  redeEpistemicaView.setDistanciaMaximaRepulsao(Integer.valueOf(redeTag.getAttribute("distMaxRepulsao")));
52              redeEpistemicaView.setPassoMax(Integer.valueOf(redeTag.getAttribute("passoAgente")));
53              NodeList crencas = redeTag.getElementsByTagName("crenca");
```

```
54                      String parType = redeTag.getAttribute("parEpistemicoType");
55                      Map<Long,ParEpistemico> mapCrencas = new HashMap<Long,ParEpistemico>();
56                      for(int i=0;i<crencas.getLength();i++){
57                          Node crencaTag = crencas.item(i);
58                          ParEpistemico par = (ParEpistemico)Class.forName(parType).newInstance();
59                          String aStr[] = crencaTag.getAttributes().getNamedItem("a").getNodeValue().split(";");
60                          par.setSizeAntecedente(aStr.length);
61                          for(int j=0;j<aStr.length;j++){
62                              par.addAntecedente(Double.valueOf(aStr[j]));
63                          }
64                          String cStr[] = crencaTag.getAttributes().getNamedItem("c").getNodeValue().split(";");
65                          par.setSizeConsequente(cStr.length);
66                          for(int j=0;j<cStr.length;j++){
67                              par.addConsequente(Double.valueOf(cStr[j]));
68                          }
69                          par.setNome(crencaTag.getAttributes().getNamedItem("nome").getNodeValue());
70                          par.setId(Long.valueOf(crencaTag.getAttributes().getNamedItem("cid").getNodeValue()));
71                          Color cor = new
72      Color(Integer.parseInt(crencaTag.getAttributes().getNamedItem("color").getNodeValue(), 16));
73                          par.setCor(cor);
74                          mapCrencas.put(par.getId(), par);
75                      }
76                      Map<Long,AgenteEpistemico> mapAgentes = new HashMap<Long,AgenteEpistemico>();
77                      NodeList agentes = redeTag.getElementsByTagName("agente");
78                      ArrayList<InsertAresta> inserts = new ArrayList<InsertAresta>();
79                      ArrayList<AgenteEpistemico> agenteList = new ArrayList<AgenteEpistemico>();
80                      for(int i=0;i<agentes.getLength();i++){
81                          Node agenteTag = agentes.item(i);
82                          AgenteEpistemico agente = new AgenteEpistemico(nns);
83                          NamedNodeMap att = agenteTag.getAttributes();
84                          agente.setId(Long.valueOf(att.getNamedItem("aid").getNodeValue()));
85                          agente.setMaxDiff(Double.valueOf(att.getNamedItem("maxDif").getNodeValue()));
86
87      agente.setMorrerEmXpublicacoes(Integer.valueOf(att.getNamedItem("morrerEm").getNodeValue()));
88
89      agente.setSomenteUltimaTeoria(att.getNamedItem("somenteUltimo").getNodeValue().equals("1"));
90
91      agente.setCriarNovoEm(Integer.valueOf(att.getNamedItem("criarNovoEm").getNodeValue()));
92                          agente.setVontadeDePublicar(Double.valueOf(att.getNamedItem("freq").getNodeValue()));
93                          agente.setX(Integer.valueOf(att.getNamedItem("x").getNodeValue()));
94                          agente.setY(Integer.valueOf(att.getNamedItem("y").getNodeValue()));
95                          agente.setRaio(Integer.valueOf(att.getNamedItem("raio").getNodeValue()));
96                          NodeList arestasCrencasref = agenteTag.getChildNodes();
97                          ArrayList<ParEpistemico> par2add = new ArrayList<ParEpistemico>();
98                          Integer qtd=1000;
99                          for(int j=0;j<arestasCrencasref.getLength();j++){
100                             Node arestaCrenca = arestasCrencasref.item(j);
101                             if(arestaCrenca.getNodeName().equals("crencaref")){
102                                 Long parId =
103     Long.parseLong(arestaCrenca.getAttributes().getNamedItem("ref").getNodeValue());
104                                 qtd =
105     Integer.parseInt(arestaCrenca.getAttributes().getNamedItem("qtd").getNodeValue());
106                                 ParEpistemico par = mapCrencas.get(parId);
107                                 par2add.add(par);
108                             }else if (arestaCrenca.getNodeName().equals("aresta")){
```

```
109                                     inserts.add(
110                                             new InsertAresta(agente,
111
112     Double.parseDouble(arestaCrenca.getAttributes().getNamedItem("peso").getNodeValue()),
113
114     Long.parseLong(arestaCrenca.getAttributes().getNamedItem("aid").getNodeValue()),
115                                                             mapAgentes
116                                             ));
117                                 }
118                             }
119                             if(qtd>0){
120                                 logger.info("treinando agente " + agente.getId() + " qtd = " + qtd + " crencas " +
121     par2add.size());
122                                 agente.treinar(par2add,qtd);
123                                 agente.addCrencas(par2add);
124                             }
125                             mapAgentes.put(agente.getId(), agente);
126                             agenteList.add(agente);
127                             agente.setRedeEpistemica(redeEpistemica);
128                     }
129                     for(InsertAresta insert:inserts){
130                         insert.execute();
131                     }
132                     //matar os antigos
133                     for(AgenteEpistemico agente: redeEpistemica.getListAgenteEpistemico()){
134                         redeEpistemica.matarAgente(agente);
135                     }
136                     for(AgenteEpistemico agente: agenteList){
137                         redeEpistemica.fireAgenteCriadoEvent(agente);
138                     }
139                     redeEpistemica.setListAgenteEpistemico(agenteList);
140
141                     redeEpistemicaView.refresh();
142                 } catch (Exception e) {
143                     e.printStackTrace();
144                 }
145             }
146
147         public static String getXml(RedeEpistemica redeEpistemica, RedeEpistemicaView redeEpistemicaView,
148     ConfiguracoesPanel configuracoesPanel){
149                 String parType = ParEpistemicoOrkut.class.getCanonicalName();
150                 StringBuilder crencaSb = new StringBuilder();
151                 for(AgenteEpistemico agente : redeEpistemica.getListAgenteEpistemico()){
152                     List<ParEpistemico> crencas = agente.getCrencas();
153                     for(ParEpistemico crenca : crencas){
154                         parType = crenca.getClass().getCanonicalName();
155                         crencaSb.append(CrencaDao.getXml(crenca));
156                     }
157                 }
158                 StringBuilder sb = new StringBuilder();
159                 int a[] = redeEpistemica.getListAgenteEpistemico().get(0).getNeuralNetworkStructure();
160                 String nns = a[0]+","+a[1]+","+a[2];
161                 String template = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<rede
162     neuralNetworkStructure=\"{nns}\" " +
163                     "distMaxRepulsao=\"{distMaxRepulsao}\" " +
```

```
                    "passoAgente=\"{passoAgente}\" " +
                    "distrAleatoria=\"{distrAleatoria}\" " +
                    "parEpistemicoType=\"{parType}\">";
            template = template
                    .replace("{nns}", nns)
                    .replace("{distMaxRepulsao}", redeEpistemicaView.getDistanciaMaxRepulsao()+"")
                    .replace("{passoAgente}", redeEpistemicaView.getPassoMax()+"")
                    .replace("{distrAleatoria}", configuracoesPanel.getChkDistribuicaoAleatoria().isSelected()?
"s":"n")
                    .replace("{parType}", parType);
            sb.append(template);
            sb.append(crencaSb.toString());
            for(AgenteEpistemico agente: redeEpistemica.getListAgenteEpistemico()){
                sb.append(AgenteDao.getXml(agente));
            }

            sb.append("</rede>");
            return sb.toString();
        }
    }
    class InsertAresta{
        AgenteEpistemico agente;
        double peso;
        long idAgente;
        Map<Long, AgenteEpistemico> mapAgentes;
        public InsertAresta(AgenteEpistemico agente, double peso, long idAgente, Map<Long,
    AgenteEpistemico> mapAgentes) {
            super();
            this.agente = agente;
            this.peso = peso;
            this.idAgente = idAgente;
            this.mapAgentes = mapAgentes;
        }
        void execute(){
            AgenteEpistemico receptor = mapAgentes.get(idAgente);
            if(receptor!=null){
                agente.conhecer(receptor, peso);
            }else{
                throw new RuntimeException("Agente id " + idAgente + " not found.");
            }
        }
    }
```

modelo/ AgenteEpistemico.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  import java.awt.Color;
4  import java.text.DecimalFormat;
5  import java.text.DecimalFormatSymbols;
6  import java.util.ArrayList;
7  import java.util.List;
8  import java.util.Locale;
9
10 import org.apache.log4j.Logger;
11
12 import br.unicarioca.redesepistemicas.bo.BackPropagation;
13
14 import com.rmit.neuralnetwork.NeuralNetwork;
15 import com.rmit.neuralnetwork.training.Training;
16 import com.rmit.neuralnetwork.trainingdata.TrainingExample;
17 import com.rmit.neuralnetwork.trainingdata.TrainingSet;
18
19 /**
20  * Um agente epistêmico é um construto capaz de gerar, comunicar e representar
21  * pares epistêmicos.
22  */
23 public class AgenteEpistemico{
24       private static Logger logger = Logger.getLogger(AgenteEpistemico.class);
25       private List<ParEpistemico> crencas = new ArrayList<ParEpistemico>();
26
27       /**
28        * O sistema ficou muito lento ao formatar no toString()
29        * fica aqui para documentar o problema
30        */
31       private static DecimalFormat decimalFormat = new DecimalFormat("#,##0.000", new
32 DecimalFormatSymbols (new Locale ("pt", "BR")));
33
34       /**
35        * Arestas que saem deste agente para outros<br>
36        * arestas usadas para falar
37        */
38       private List<Aresta> arestasSaida = new ArrayList<Aresta>();
39       private List<Aresta> arestasEntrada = new ArrayList<Aresta>();
40       private NeuralNetwork neuralNetwork;
41       private int x,y;
42       private int raio;
43       private int numberEvaluations = 50;
44       private String nome;
45       private static long lastId=0;
46       private long id;
47       private int qtdParComunicado=0;
48       private int criarNovoEm = 10;
49       private boolean somenteUltimaTeoria = true;
50       private int morrerEmXpublicacoes = 100;
51       private Double pesoReputacao = null;
52       private Color color;
53       private double vontadeDePublicar=1.0;
```

```
54      private Foco foco;
55      private double maxAtencao = 100;
56      private RedeEpistemica redeEpistemica;
57      private double errorTolerance = 0.0;
58
59      /**
60       * Rede
61       * @return rede a qual este agente pertence
62       */
63      public RedeEpistemica getRedeEpistemica() {
64            return redeEpistemica;
65      }
66
67      /**
68       * Atribui a rede
69       * @param redeEpistemica rede a qual este agente pertence
70       */
71      public void setRedeEpistemica(RedeEpistemica redeEpistemica) {
72            this.redeEpistemica = redeEpistemica;
73      }
74
75      /**
76       * Foco
77       * @return foco do agente
78       */
79      public Foco getFoco() {
80            return foco;
81      }
82
83      /**
84       * Foco
85       * @param foco do agente
86       */
87      public void setFoco(Foco foco) {
88            this.foco = foco;
89      }
90
91      /**
92       * Representa a vontade de publicar alguma coisa
93       * @return Double normalmente de 0 a 1
94       */
95      public double getVontadeDePublicar() {
96            return vontadeDePublicar;
97      }
98
99      /**
100      * Representa a vontade de publicar alguma coisa
101      * @param vontadeDePublicar normalmente de 0 a 1
102      */
103     public void setVontadeDePublicar(double vontadeDePublicar) {
104           this.vontadeDePublicar = vontadeDePublicar;
105     }
106
107     /**
108      * Cor para desenhar o agente
```

```
109         * @return Color
110         */
111        public Color getColor() {
112                return color;
113        }
114
115        /**
116         * Cor para desenhar o agente
117         * @param color cor para o desenho na interface do usu&aacute;rio
118         */
119        public void setColor(Color color) {
120                this.color = color;
121        }
122
123        /**
124         * O maximo de diferenca para rejeitar o resultado
125         */
126        private Double maxDiff = 0.2;
127
128        private int[] neuralNetworkStructure;
129        public ArrayList<ParEpistemico> crencaMonitorada = new ArrayList<ParEpistemico>();
130
131        public AgenteEpistemico(int[] neuralNetworkStructure) {
132                this.neuralNetworkStructure = neuralNetworkStructure;
133                neuralNetwork = new NeuralNetwork(neuralNetworkStructure);
134                neuralNetwork.initWeights();
135                foco = FocoFactory.criarFoco();
136                //gerarPar();
137                //gera um nome padrao
138                id = lastId++;
139                setNome("A" + id);
140        }
141
142        public int[] getNeuralNetworkStructure() {
143                return neuralNetworkStructure;
144        }
145
146        /**
147         * Identificador do agente
148         * @return Long
149         */
150        public long getId() {
151                return id;
152        }
153
154        /**
155         * Identificador do agente
156         * @param id Long
157         */
158        public void setId(long id) {
159                this.id = id;
160        }
161
162        /**
163         * De quanto em quanto tempo publica um novo par para publicar<br>
```

```
164          * coloque 1 para publicar um novo par epistemico sempre.
165          * @return int
166          */
167         public int getCriarNovoEm() {
168               return criarNovoEm;
169         }
170
171         /**
172          * De quanto em quanto tempo publica um novo par para publicar<br>
173          * coloque 1 para publicar um novo par epistemico sempre.
174          * @param criarNovoEm
175          */
176         public void setCriarNovoEm(int criarNovoEm) {
177               this.criarNovoEm = criarNovoEm;
178         }
179
180         /**
181          * M&aacute;ximo de diferen&ccedil;a de um consequente para o outro
182          * para que este agente aceite o resultado durante o receberComunicado e fa&ccedil;a um treinamento.
183          * @return Double
184          */
185         public Double getMaxDiff() {
186               return maxDiff;
187         }
188
189         /**
190          * M&aacute;ximo de diferen&ccedil;a de um consequente para o outro
191          * para que este agente aceite o resultado durante o receberComunicado e fa&ccedil;a um treinamento.
192          * @param maxDiff Double
193          */
194         public void setMaxDiff(Double maxDiff) {
195               this.maxDiff = maxDiff;
196         }
197
198         /**
199          * Nome deste agente
200          * @return nome
201          */
202         public String getNome() {
203               return nome;
204         }
205
206         /**
207          *
208          * @param nome nome
209          */
210         public void setNome(String nome) {
211               this.nome = nome;
212         }
213
214         /**
215          * Sorteia um novo espaço X,Y e Z e gera um par,
216          * delegado para ParEpistemicoFactory
217          * @return novo par
218          */
```

```
219         private ParEpistemico criarNovoPar(){
220               ParEpistemico parEpistemico;
221               if(neuralNetworkStructure[0]==3 && neuralNetworkStructure[2]==2){
222                     ParEpistemico3x2 parEpistemico2 = ParEpistemicoFactory.criar(foco);
223                     //atualizar o consequente
224                     Consequente consequente = new Consequente();
225                     consequente = new Consequente(neuralNetwork.getOutputs());
226                     parEpistemico2.setConsequente(consequente);
227                     return parEpistemico2;
228               }else{
229                     parEpistemico =
230   ParEpistemicoFactory.criar(neuralNetworkStructure[0],neuralNetworkStructure[2]);
231                     parEpistemico = interpretar(parEpistemico);
232               }
233               return parEpistemico;
234         }
235
236         /**
237          * Se for nulo, calcula o valor de todas as arestas de saida.<br>
238          * Se nao, retorna o valor já calculado
239          *
240          * @return valor somado de todas as arestas
241          */
242         public double getPesoReputacao(){
243               //if(pesoReputacao==null){
244                     double res = 0;
245                     synchronized (arestasSaida) {
246                           for(Aresta aresta:arestasSaida){
247                                 res+=aresta.getPeso();
248                           }
249                     }
250                     pesoReputacao=res;
251               //}
252               return pesoReputacao;
253         }
254
255         /**
256          * Usado para:<br>
257          * 1 - zerar, ou seja, colocar null
258          * quando null o peso é todo recalculado
259          * <br>
260          * 2 - Normalizar o peso em {@link RedeEpistemica#normalizarPesos()}
261          * @param pesoReputacao
262          */
263         public void setPesoReputacao(Double pesoReputacao){
264               this.pesoReputacao = pesoReputacao;
265         }
266
267         /**
268          * Usado para comodidade<br>
269          * pesoReputacao = getPesoTotal()+x;
270          * @param x valor a ser adicionado
271          */
272         public void addPesoReputacao(double x){
273               pesoReputacao = getPesoReputacao()+x;
```

```
274         }
275
276         /**
277          * Gera um par aleatorio,
278          * ou pega um par aleatorio do conhecimento
279          */
280         public ParEpistemico gerarPar(){
281               qtdParComunicado++;
282
283               ParEpistemico parEpistemico;
284               if(crencas.size()==0){
285                     //iniciar
286                     parEpistemico = criarNovoPar();
287                     treinar(parEpistemico,500);
288                     parEpistemico = interpretar(parEpistemico);
289                     crencas.add(parEpistemico);
290               }else if(criarNovoEm!=0 && qtdParComunicado%criarNovoEm==0){
291                     //Nova teoria
292                     parEpistemico = criarNovoPar();
293                     treinar(parEpistemico,500);
294                     parEpistemico = interpretar(parEpistemico);
295                     crencas.add(parEpistemico);
296               }else if(somenteUltimaTeoria){
297                     parEpistemico = crencas.get(crencas.size()-1);
298               }else{
299                     int indice = (int)((crencas.size())*NumeroAleatorio.gerarNumero());
300                     logger.debug("crenca " + indice);
301                     parEpistemico = crencas.get(indice);
302               }
303
304
305               return parEpistemico;
306         }
307
308         /**
309          * Somente interpreta e retorna um novo par de
310          * acordo com a sua visao
311          * @param parEpistemico
312          * @return ParEpistemico
313          */
314         public ParEpistemico interpretar(ParEpistemico parEpistemico){
315               ParEpistemico retorno = null;
316               try {
317                     retorno = (ParEpistemico)parEpistemico.clone();
318                     for(int i=0;i<parEpistemico.getSizeAntecedente();i++){
319                           retorno.addAntecedente(parEpistemico.getDoubleAntecedentes().get(i));
320                     }
321                     neuralNetwork.setInputs(parEpistemico.getDoubleAntecedentes());
322                     for(int i=0;i<parEpistemico.getSizeConsequente();i++){
323                           retorno.addConsequente(neuralNetwork.getOutputs().get(i));
324                     }
325               } catch (CloneNotSupportedException e) {
326                     e.printStackTrace();
327               }
328               return retorno;
```

```
329         }
330 
331         /**
332          * Treina mas nao adiciona nas crencas
333          * @param pares
334          * @param qtd
335          */
336         public void treinar(List<ParEpistemico> pares, int qtd) {
337               ArrayList<TrainingExample> listTraining = new ArrayList<TrainingExample>();
338               for(ParEpistemico parEpistemicoInformado:pares){
339                     TrainingExample te = new TrainingExample();
340                     ArrayList<Double> in = parEpistemicoInformado.getDoubleAntecedentes();
341                     ArrayList<Double> out = parEpistemicoInformado.getDoubleConsequentes();
342                     if(out.size()!=parEpistemicoInformado.getSizeConsequente()){
343                           throw new RuntimeException("Out errado!");
344                     }
345                     if(in.size()!=parEpistemicoInformado.getSizeAntecedente()){
346                           throw new RuntimeException("In errado!");
347                     }
348                     te.setInputs(in);
349                     te.setOutputs(out);
350                     listTraining.add(te);
351               }
352 
353 
354               // pso specific settings
355               int numberParticles = 20;
356               double learningFactor = 1.49618;
357               double inertialWeight = 0.7298;
358 
359               logger.debug("qtd = " + qtd);
360               // create a instance of a training method
361               //Training training = new ParticleSwarmOptimization(numberEvaluations, errorTolerance,
362 learningFactor, inertialWeight, numberParticles);
363               Training training = new BackPropagation(qtd,errorTolerance);
364 
365               // set the training method and set for the neural network to use
366               neuralNetwork.setTraining(training);
367               TrainingSet trainingSet = new TrainingSet("MyT",listTraining);
368               //neuralNetwork.setTrainingSet(trainingData.getTrainingSet());
369               neuralNetwork.setTrainingSet(trainingSet);
370 
371               neuralNetwork.train();
372 
373         }
374 
375         /**
376          * Treina mas nao adiciona nas crencas
377          * @param parEpistemicoInformado
378          * @param qtd
379          */
380         public void treinar(ParEpistemico parEpistemicoInformado,int qtd){
381               List<ParEpistemico> pares = new ArrayList<ParEpistemico>();
382               pares.add(parEpistemicoInformado);
383               treinar(pares, qtd);
```

```
384         }
385
386         /**
387          * Recebe um comunicado, uma publica&ccedil;&atilde;o
388          * @param parEpistemicoInformado par informado
389          * @param aresta relacao com o emissor
390          * @param emissor Agente emissor, publicador
391          * @return Double deltaErro
392          */
393         public Double receberComunicado(ParEpistemico parEpistemicoInformado,Aresta aresta,
394 AgenteEpistemico emissor){
395                 double peso = aresta.getPeso();
396                 double deltaErro;
397                 //guarda a informacao?
398                 ParEpistemico parEpistemicoExistente = procurar(parEpistemicoInformado);
399                 if(parEpistemicoExistente==null){
400                         logger.debug("Aprendendo " + parEpistemicoInformado);
401                         parEpistemicoExistente = interpretar(parEpistemicoInformado);
402                         deltaErro = parEpistemicoExistente.calcularDiferencaConsequente(parEpistemicoInformado);
403                         if(maxDiff>deltaErro){
404                                 treinar(parEpistemicoInformado,(int)(numberEvaluations * peso * deltaErro));
405                         }//else recusa a aprender
406
407                 }else{
408                         //ja existe, ver diferenca
409                         neuralNetwork.train();
410                         ParEpistemico parEpistemicoPessoalDepoisTreino = interpretar(parEpistemicoInformado);
411                         logger.debug("evolucao depois treino = "+
412                         parEpistemicoExistente.calcularDiferencaConsequente(parEpistemicoPessoalDepoisTreino)
413                         );
414                         for(int i=0;i<parEpistemicoPessoalDepoisTreino.getSizeConsequente();i++){
415
416 parEpistemicoExistente.addConsequente(parEpistemicoPessoalDepoisTreino.getDoubleConsequentes().get(i));
417                         }
418
419                         deltaErro =
420 parEpistemicoPessoalDepoisTreino.calcularDiferencaConsequente(parEpistemicoInformado);
421                 }
422                 if(emissor!=null){
423                         double delta = 0.2 * (1.0/(deltaErro+1.0)) * peso; /*regra de Hebb*/
424                         aresta.setPeso(peso + delta);
425                         //emissor.addPesoReputacao(delta);
426                         //retirar o delta dos outros agentes
427                         //retirarDelta(delta,emissor,peso);
428                         distribuirAtencao();
429                 }
430                 return deltaErro;
431         }
432         private void distribuirAtencao(){
433                 //recuperar o maximo
434                 double max = arestasEntrada.get(0).getPeso();
435                 double total = 0;
436                 for(Aresta aresta:arestasEntrada){
437                         max = Math.max(aresta.getPeso(), max);
438                         total +=aresta.getPeso();
```

```
439             }
440             for(Aresta aresta:arestasEntrada){
441                 double peso = aresta.getPeso()/total;
442                 peso = peso*maxAtencao;
443                 aresta.setPeso(peso);
444             }
445         }
446         private void retirarDelta(double delta,AgenteEpistemico emissor,double pesoEmissor) {
447             if(redeEpistemica.isNormalizarPesos()) return;
448             double relacao = delta/(maxAtencao-pesoEmissor);
449             double retirado = 0;
450             for(Aresta aresta:arestasEntrada){
451                 if(!aresta.getEmissor().equals(emissor)){
452                     double peso = aresta.getPeso();
453                     double retirar = peso*relacao;
454                     retirado+=retirar;
455                     aresta.setPeso(peso-retirar);
456                 }
457             }
458             if(retirado!=delta){
459                 //throw new RuntimeException("Erro de precisao?"+(delta-retirado));
460             }
461         }
462         /**
463          * Procura dentro da crenca
464          * @param parAntecedente
465          * @return ParEpistemico
466          */
467         private ParEpistemico procurar(ParEpistemico parAntecedente){
468             for(ParEpistemico parEpistemico:crencas){
469                 if(parEpistemico.antecedenteEquals(parAntecedente)){
470                     return parEpistemico;
471                 }
472             }
473             return null;
474         }
475
476         /**
477          * Arestas para comunicar
478          * @return Arestas de Saida de fala
479          */
480         public List<Aresta> getArestas() {
481             return arestasSaida;
482         }
483
484         /**
485          * Arestas para comunicar
486          * @param arestas Arestas
487          */
488         public void setArestas(List<Aresta> arestas) {
489             this.arestasSaida = arestas;
490         }
491
492         public int getX() {
493             return x;
```

```
494         }
495         public int getY() {
496               return y;
497         }
498         public void setX(int x) {
499               this.x = x;
500         }
501         public void setY(int y) {
502               this.y = y;
503         }
504
505         /**
506          * Coloca na lista de arestas
507          * @param agenteNovo novo agente
508          * @param peso novo peso
509          */
510         public void conhecer(AgenteEpistemico agenteNovo, double peso) {
511               Aresta aresta = new Aresta();
512               aresta.setEmissor(this);
513               aresta.setReceptor(agenteNovo);
514               aresta.setPeso(peso);
515               synchronized (arestasSaida) {
516                     arestasSaida.add(aresta);
517                     agenteNovo.arestasEntrada.add(aresta);
518               }
519               pesoReputacao = null;
520         }
521
522         public int getRaio() {
523               return raio;
524         }
525         public void setRaio(int raio) {
526               this.raio = raio;
527         }
528
529
530         /**
531          * retorna o this.nome
532          */
533         @Override
534         public String toString() {
535               if(color!=null){
536                     return nome + " ["+qtdParComunicado+"] " + getPesoReputacao()+ "*";
537               }else{
538                     return nome + " ["+qtdParComunicado+"] " + getPesoReputacao();
539               }
540         }
541
542         /**
543          * Usado para matar o agente.
544          * Pede para todos que possui aresta que
545          * remova ele
546          */
547         public void morrer() {
548               for(Aresta aresta:arestasSaida){
```

```
549                     aresta.getReceptor().removerAgente(this);
550                 }
551             }
552
553             /**
554              * Remove o agente informado das arestas de saida
555              * @param agenteEpistemico Agente
556              */
557             private void removerAgente(AgenteEpistemico agenteEpistemico) {
558                 Aresta aresta = new Aresta();
559                 aresta.setReceptor(agenteEpistemico);
560                 arestasSaida.remove(aresta);
561             }
562
563             @Override
564             public boolean equals(Object obj) {
565                 if(obj==null) return false;
566                 AgenteEpistemico a = (AgenteEpistemico)obj;
567                 logger.debug("a.id==this.id " + (a.id==this.id));
568                 return a.id==this.id;
569             }
570
571             /**
572              * Quantidade de pares publicados
573              * @return int
574              */
575             public int getQtdParComunicado() {
576                 return qtdParComunicado;
577             }
578
579             /**
580              * Configura um n&uacute;mero para que depois de publicar esta quantidade,
581              * o agente morra
582              * @return int
583              */
584             public int getMorrerEmXpublicacoes() {
585                 return morrerEmXpublicacoes;
586             }
587
588             /**
589              * zero para imortal
590              * @param morrerEmXpublicacoes
591              */
592             public void setMorrerEmXpublicacoes(int morrerEmXpublicacoes) {
593                 this.morrerEmXpublicacoes = morrerEmXpublicacoes;
594             }
595
596             /**
597              * true publicar&aacute; somente a ultima teoria criada
598              * @param somenteUltimaTeoria boolean
599              */
600             public void setSomenteUltimaTeoria(boolean somenteUltimaTeoria) {
601                 this.somenteUltimaTeoria = somenteUltimaTeoria;
602             }
603
```

```
604         public boolean isSomenteUltimaTeoria() {
605               return somenteUltimaTeoria;
606         }
607
608         /**
609          * Cren&amp;ccedil;as deste agente
610          * @return lista de pares epistemicos
611          */
612         public List<ParEpistemico> getCrencas() {
613               return crencas;
614         }
615
616         /**
617          * Adiciona se nao existir
618          * @param pares
619          */
620         public void addCrencas(List<ParEpistemico> pares) {
621               for(ParEpistemico par:pares){
622                     ParEpistemico procurado = procurar(par);
623                     if(procurado==null){
624                           logger.info("adicionando crenca " + par.getId() + " no agente " + this.getId());
625                           crencas.add(par);
626                     }else{
627                           logger.info("ja existe a crenca " + par.getId() + " no agente " + this.getId());
628                     }
629               }
630         }
631
632         /**
633          * Verifica se o agente quer publicar
634          * @return true caso queira publicar
635          */
636         public boolean querPublicar() {
637               return NumeroAleatorio.gerarNumero()<vontadeDePublicar;
638         }
639
640         /**
641          * Usado para treinar a rede interna
642          * @param errorTolerance double para treinar a rede
              */
              public void setErrorTolerance(double errorTolerance) {
                    this.errorTolerance = errorTolerance;
              }

        }
```

modelo/ AgenteEpistemicoFactory.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  /**
4  * Interface para criar agentes
5  */
6  public interface AgenteEpistemicoFactory {
7
8  /**
9  * Criar um agente epistemico com as configuracoes
10 * @return AgenteEpistemico
11 */
12 public AgenteEpistemico criarAgente();
13 }
```

modelo/ Antecedente.java

```
 1  package br.unicarioca.redesepistemicas.modelo;
 2
 3  /**
 4   * Antecedente do par epistemico
 5   */
 6  public class Antecedente {
 7  	private Double x;
 8  	private Double y;
 9  	private Double z;
10
11  	/**
12  	 * @return the x
13  	 */
14  	public Double getX() {
15  		return x;
16  	}
17  	/**
18  	 * @param x the x to set
19  	 */
20  	public void setX(Double x) {
21  		this.x = x;
22  	}
23  	/**
24  	 * @return the y
25  	 */
26  	public Double getY() {
27  		return y;
28  	}
29  	/**
30  	 * @param y the y to set
31  	 */
32  	public void setY(Double y) {
33  		this.y = y;
34  	}
35  	/**
36  	 * @return the z
37  	 */
38  	public Double getZ() {
39  		return z;
40  	}
41  	/**
42  	 * @param z the z to set
43  	 */
44  	public void setZ(Double z) {
45  		this.z = z;
46  	}
47  	public void add(Object valueAt) {
48  		if(x==null){
49  			x = Double.valueOf(valueAt.toString());
50  		}else if(y==null){
51  			y = Double.valueOf(valueAt.toString());
52  		}else{
53  			z = Double.valueOf(valueAt.toString());
```

```
54              }
55          }
56
57          /* (non-Javadoc)
58           * @see java.lang.Object#equals(java.lang.Object)
59           */
60          @Override
61          public boolean equals(Object obj) {
62              if (this == obj)
63                  return true;
64              if (obj == null)
65                  return false;
66              if (!(obj instanceof Antecedente))
67                  return false;
68              Antecedente other = (Antecedente) obj;
69              if (x == null) {
70                  if (other.x != null)
71                      return false;
72              } else if (!x.equals(other.x))
73                  return false;
74              if (y == null) {
75                  if (other.y != null)
76                      return false;
77              } else if (!y.equals(other.y))
78                  return false;
79              if (z == null) {
80                  if (other.z != null)
81                      return false;
82              } else if (!z.equals(other.z))
83                  return false;
84              return true;
85          }
86
87      }
```

modelo/ Aresta.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  /**
4   * Aresta de relacao entre um agente emissor e um receptor
5   */
6  public class Aresta {
7        private Double peso;
8        private AgenteEpistemico receptor;
9        private AgenteEpistemico emissor;
10       public AgenteEpistemico getEmissor() {
11             return emissor;
12       }
13       public void setEmissor(AgenteEpistemico emissor) {
14             this.emissor = emissor;
15       }
16
17       /**
18        * Peso que o receptor dá ao emissor
19        * @return the peso
20        */
21       public Double getPeso() {
22             return peso;
23       }
24       /**
25        * @param peso the peso to set
26        */
27       public void setPeso(Double peso) {
28             this.peso = peso;
29       }
30       /**
31        * @return receptor
32        */
33       public AgenteEpistemico getReceptor() {
34             return receptor;
35       }
36       /**
37        * @param receptor the agenteEpistemico to set
38        */
39       public void setReceptor(AgenteEpistemico receptor) {
40             this.receptor = receptor;
41       }
42       @Override
43       public boolean equals(Object obj) {
44             Aresta a = (Aresta)obj;
45             if(this.getReceptor().equals(a.getReceptor())){
46                   return true;
47             }else{
48                   return false;
49             }
50       }
51 }
```

modelo/ CicloVidaAgenteListener.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2  /**
3   * Listener de ciclo de vida de um agente <br>
4   * que recebe eventos de quando o agente &eacute; criado ou morto
5   */
6  public interface CicloVidaAgenteListener {
7
8          /**
9           * Chamado quando o agente &eacute; criado
10          * @param agente AgenteEpistemico
11          */
12         public void criado(AgenteEpistemico agente);
13
14         /**
15          * Chamado quando o agente &eacute; morto
16          * @param agente AgenteEpistemico
17          */
18         public void morto(AgenteEpistemico agente);
19 }
```

modelo/ ComunicacaoListener.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  /**
4   * Listener para receber eventos de comunicacao
5   */
6  public interface ComunicacaoListener {
7          /**
8           * Informa qual agente foi escolhido como emissor ou comunicador
9           * @param emissor AgenteEpistemico sorteado
10          */
11         public void comunicadorEscolhido(AgenteEpistemico emissor);
12
13         /**
14          * Chamado depois de uma comunicacao ocorrer
15          * @param emissor dono do par transmitido
16          * @param receptor quem recebe o par
17          * @param peso Peso da relacao
18          * @param diff diferenca dos pontos de vista do emissor para o receptor
19          */
20         public void depoisDeComunicar(AgenteEpistemico emissor,AgenteEpistemico receptor, Double
21 peso,Double diff);
   }
```

modelo/ Consequente.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  import java.util.ArrayList;
4
5  /**
6   * Consequente do par epistemico
7   */
8  public class Consequente {
9          private Double x;
10         private Double y;
11
12         public Consequente(){}
13
14         public Consequente(ArrayList<Double> outputs) {
15                 x = outputs.get(0);
16                 y = outputs.get(1);
17         }
18         /**
19          * @return the x
20          */
21         public Double getX() {
22                 return x;
23         }
24         /**
25          * @param x the x to set
26          */
27         public void setX(Double x) {
28                 this.x = x;
29         }
30         /**
31          * @return the y
32          */
33         public Double getY() {
34                 return y;
35         }
36         /**
37          * @param y the y to set
38          */
39         public void setY(Double y) {
40                 this.y = y;
41         }
42
43         public void add(Object valueAt) {
44                 if(x==null){
45                         x = Double.valueOf(valueAt.toString());
46                 }else{
47                         y = Double.valueOf(valueAt.toString());
48                 }
49         }
50
51 }
```

modelo/ Experimento.java

```
 1  package br.unicarioca.redesepistemicas.modelo;
 2
 3  import java.util.HashSet;
 4  import java.util.Set;
 5
 6  /**
 7   * Modelo de experimento, com variaveis definidas
 8   */
 9  public class Experimento {
10      private AgenteEpistemico pivo;
11      private double maxDiff=0.002;
12      private Set<ParEpistemico> crencas;
13      private double minDiff=0.007;
14
15      public Experimento(){
16          crencas = new HashSet<ParEpistemico>();
17      }
18
19      public AgenteEpistemico getPivo() {
20          return pivo;
21      }
22      public void setPivo(AgenteEpistemico pivo) {
23          this.pivo = pivo;
24      }
25
26      public void addCrenca(ParEpistemico par){
27          crencas.add(par);
28      }
29
30      public Set<ParEpistemico> getSetCrencas(){
31          return crencas;
32      }
33
34      public double getMaxDiff() {
35          return maxDiff;
36      }
37      public void setMaxDiff(double maxDiff) {
38          this.maxDiff = maxDiff;
39      }
40
41      public void setMinDiff(double minDiff) {
42          this.minDiff = minDiff;
43      }
44
45      public double getMinDiff() {
46          return minDiff;
47      }
48  }
```

modelo/ Foco.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  /**
4   * Foco do agente epistemico
5   *
6   */
7  public class Foco {
8  	private Double x,y,z,raio;
9
10 	/**
11 	 * @return the x
12 	 */
13 	public Double getX() {
14 		return x;
15 	}
16
17 	/**
18 	 * @param x the x to set
19 	 */
20 	public void setX(Double x) {
21 		this.x = x;
22 	}
23
24 	/**
25 	 * @return the y
26 	 */
27 	public Double getY() {
28 		return y;
29 	}
30
31 	/**
32 	 * @param y the y to set
33 	 */
34 	public void setY(Double y) {
35 		this.y = y;
36 	}
37
38 	/**
39 	 * @return the z
40 	 */
41 	public Double getZ() {
42 		return z;
43 	}
44
45 	/**
46 	 * @param z the z to set
47 	 */
48 	public void setZ(Double z) {
49 		this.z = z;
50 	}
51
52 	/**
53 	 * @return the raio
```

```
54          */
55         public Double getRaio() {
56                 return raio;
57         }
58
59         /**
60          * @param raio the raio to set
61          */
62         public void setRaio(Double raio) {
63                 this.raio = raio;
64         }
65
66   }
```

modelo/ FocoFactory.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  /**
4   * Responsavel por gerar um foco no espa&ccedil;o para o agente epistemico
5   */
6  public class FocoFactory {
7          /**
8           * Cria um foco pseudo aleatoriamente
9           * @return Foco no espa&ccedil;o
10          */
11         public static Foco criarFoco(){
12                 Foco retorno = new Foco();
13                 retorno.setX(NumeroAleatorio.gerarNumero());
14                 retorno.setY(NumeroAleatorio.gerarNumero());
15                 retorno.setZ(NumeroAleatorio.gerarNumero());
16                 retorno.setRaio(NumeroAleatorio.gerarNumero());
17                 return retorno;
18         }
19 }
```

modelo/ NumeroAleatorio.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  import java.util.Random;
4
5  /**
6   * Classe utilitaria para gerar um numero pseudo aleatorio.<br>
7   * Desta forma os numeros sorteados podem ser os mesmos.
8   */
9  public class NumeroAleatorio {
10         private static Random random = new Random(10L);
11
12         /**
13          * Numero
14          * @return numero pseudo aleatorio
15          */
16         public static double gerarNumero(){
17                 return random.nextDouble();
18         }
19
20         /**
21          * Reinicia o sorteio
22          */
23         public static void restart(){
24                 random = new Random(10L);
25         }
26 }
```

modelo/ ParEpistemico.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  import java.awt.Color;
4  import java.util.ArrayList;
5
6  public interface ParEpistemico extends Cloneable{
7        public long getId();
8        public void setId(long id);
9        public void addAntecedente(Double d);
10       public void addConsequente(Double d);
11       public Double calcularDiferencaConsequente(ParEpistemico par);
12       public int getSizeAntecedente();
13       public int getSizeConsequente();
14       public Object clone() throws CloneNotSupportedException;
15       public boolean antecedenteEquals(ParEpistemico antecedente);
16       public ArrayList<Double> getDoubleAntecedentes();
17       public ArrayList<Double> getDoubleConsequentes();
18       public void setNome(String nomeInRow);
19       public void setCor(Color colorInRow);
20       public Color getCor();
21       public void setSizeAntecedente(int length);
22       public void setSizeConsequente(int length);
23 }
```

modelo/ ParEpistemico3x2.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  import java.awt.Color;
4  import java.util.ArrayList;
5
6  /**
7   * Representa um par epistemico composto por um antecedente e um consequente
8   * @author Fabio Issamu Oshiro
9   */
10 public class ParEpistemico3x2 implements ParEpistemico {
11       protected Antecedente antecedente;
12       protected Consequente consequente;
13       private Color cor;
14       private String nome = "";
15       private long id=0;
16       public static long idGenerated=0;
17
18       public ParEpistemico3x2() {
19            id = idGenerated++;
20       }
21
22       @Override
23       public long getId() {
24            return id;
25       }
26       @Override
27       public void setId(long id) {
28            this.id = id;
29       }
30
31       /**
32        * @return the antecedente
33        */
34       public Antecedente getAntecedente() {
35            return antecedente;
36       }
37       /**
38        * @param antecedente the antecedente to set
39        */
40       public void setAntecedente(Antecedente antecedente) {
41            this.antecedente = antecedente;
42       }
43       /**
44        * @return the consequente
45        */
46       public Consequente getConsequente() {
47            return consequente;
48       }
49       /**
50        * @param consequente the consequente to set
51        */
52       public void setConsequente(Consequente consequente) {
53            this.consequente = consequente;
```

```
54          }
55
56          /**
57           * Calcula a diferenca,
58           * pode dar throws em class cast exception
59           * @param par Par a ser confrontado
60           * @return diferenca
61           */
62          public Double calcularDiferencaConsequente(ParEpistemico par2) {
63                ParEpistemico3x2 par = (ParEpistemico3x2)par2;
64                Double diff = 0.0;
65                Consequente con = par.getConsequente();
66
67                diff+=Math.abs(consequente.getX()-con.getX());
68                diff+=Math.abs(consequente.getY()-con.getY());
69                return diff;
70          }
71          @Override
72          public void addAntecedente(Double d) {
73                if(antecedente==null){
74                      antecedente = new Antecedente();
75                }
76                if(antecedente.getX()==null)
77                      antecedente.setX(d);
78                if(antecedente.getY()==null)
79                      antecedente.setY(d);
80                if(antecedente.getZ()==null)
81                      antecedente.setZ(d);
82          }
83          @Override
84          public void addConsequente(Double d) {
85                if(consequente==null){
86                      consequente = new Consequente();
87                }
88                if(consequente.getX()==null)
89                      consequente.setX(d);
90                if(consequente.getY()==null)
91                      consequente.setY(d);
92          }
93          @Override
94          public int getSizeAntecedente() {
95                return 3;
96          }
97          @Override
98          public int getSizeConsequente() {
99                return 2;
100         }
101
102         @Override
103         public boolean antecedenteEquals(ParEpistemico antecedente) {
104               return false;
105         }
106
107         @Override
108         public ArrayList<Double> getDoubleAntecedentes() {
```

```
109             ArrayList<Double> list = new ArrayList<Double>();
110             list.add(antecedente.getX());
111             list.add(antecedente.getY());
112             list.add(antecedente.getZ());
113             return list;
114         }
115         @Override
116         public ArrayList<Double> getDoubleConsequentes() {
117             ArrayList<Double> list = new ArrayList<Double>();
118             list.add(consequente.getX());
119             list.add(consequente.getY());
120             return list;
121         }
122 
123         @Override
124         public Object clone() throws CloneNotSupportedException {
125             return super.clone();
126         }
127         @Override
128         public void setCor(Color colorInRow) {
129             this.cor = colorInRow;
130 
131         }
132         @Override
133         public void setNome(String nomeInRow) {
134             this.nome = nomeInRow;
135         }
136         @Override
137         public Color getCor() {
138             return cor;
139         }
140 
141         @Override
142         public void setSizeAntecedente(int length) {
143         }
144 
145         @Override
146         public void setSizeConsequente(int length) {
147             // TODO Auto-generated method stub
148 
149         }
150 
151     }
```

modelo/ ParEpistemicoDiffHip.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  /**
4   * Calcula a diferenca por hipotenusa
5   */
6  public class ParEpistemicoDiffHip extends ParEpistemico3x2{
7
8          /**
9           * Calcula a diferen&ccedil;a achando o valor da hipotenusa
10          * @param par2 Par para verificar a distancia
11          * @return Double diferenca
12          */
13         public Double calcularDiferencaConsequente(ParEpistemico par2) {
14                 ParEpistemicoDiffHip par = (ParEpistemicoDiffHip) par2;
15                 Double diffX,diffY;
16                 Consequente con = par.getConsequente();
17                 diffX=consequente.getX()-con.getX();
18                 diffY=consequente.getY()-con.getY();
19                 return Math.sqrt(diffX*diffX+diffY*diffY);
20         }
21 }
```

modelo/ ParEpistemicoFactory.java

```
1   package br.unicarioca.redesepistemicas.modelo;
2
3   /**
4    * Classe responsavel por criar os pares epistemicos
5    * @author Fabio Issamu Oshiro
6    *
7    */
8   public class ParEpistemicoFactory {
9       public static void main(String[] args) {
10          for(int i=0;i<100;i++){
11              double
12  x=NumeroAleatorio.gerarNumero(),y=NumeroAleatorio.gerarNumero(),z=NumeroAleatorio.gerarNumero(),r=Numer
13  oAleatorio.gerarNumero();
14              for(int k=0;k<100;k++){
15                  double[][] a = criarPontoEmEsfera(x,y,z,r);
16                  double dist = distancia(x, y, z, a[0][0], a[0][1], a[0][2]);
17                  if(dist>r){
18                      System.out.println("ERRO");
19                  }else{
20                      System.out.println("("+x+", "+y+", "+z+") r"+r + " d"+dist);
21                  }
22              }
23          }
24      }
25
26      /**
27       * Cria um par epistemico dentro de uma esfera
28       * @param foco
29       * @return ParEpistemico
30       */
31      public static ParEpistemico3x2 criar(Foco foco) {
32          return criar(foco.getX(),foco.getY(),foco.getZ(),foco.getRaio());
33      }
34
35      /**
36       * Cria um par epistemico dentro de uma esfera
37       *
38       * @param x
39       *          coordenada
40       * @param y
41       *          coordenada
42       * @param z
43       *          coordenada
44       * @param raio
45       *          medida
46       * @return ParEpistemico
47       */
48      public static ParEpistemico3x2 criar(double x, double y, double z, double raio) {
49          double[][] a = criarPontoEmEsfera(x, y, z, raio);
50          Antecedente antecedente = new Antecedente();
51
52          antecedente.setX(normaliza(a[0][0]));
53          antecedente.setY(normaliza(a[0][1]));
```

```
54              antecedente.setZ(normaliza(a[0][2]));
55              ParEpistemico3x2 parEpistemico = new ParEpistemicoDiffHip();
56              parEpistemico.setAntecedente(antecedente);
57              Consequente consequente = new Consequente();
58              parEpistemico.setConsequente(consequente);
59              return parEpistemico;
60          }
61
62          /**
63           * Garante que o numero esta entre 0 e 1
64           * @param a numero qualquer
65           * @return valor entre 0 e 1
66           */
67          private static double normaliza(double a){
68              return Math.min(1, Math.max(0, a));
69          }
70
71          private static double[][] criarPontoEmEsfera(double x, double y, double z, double raio){
72              double nX = 0, nY = 0, nZ = NumeroAleatorio.gerarNumero() * raio;
73              double radX = Math.toRadians(360.0 * NumeroAleatorio.gerarNumero());
74              double radY = Math.toRadians(360.0 * NumeroAleatorio.gerarNumero());
75              double[][] a = {{ nX, nY, nZ, 1.0 }};
76              double[][] mX =
77                  {
78                      {1, 0, 0, 0},
79                      {0,Math.cos(radX),Math.sin(radX),0},
80                      {0,-Math.sin(radX),Math.cos(radX),0},
81                      {0, 0, 0, 1}
82                  };
83              a = multiplicar(a,mX);
84              double[][] mY =
85              {
86                  {Math.cos(radY), 0, -Math.sin(radY), 0},
87                  {0, 1, 0, 0},
88                  {Math.sin(radX),0,Math.cos(radX),0},
89                  {x, y, z, 1}
90              };
91              return multiplicar(a,mY);
92          }
93
94          /**
95           * Calcula a distancia de Ponto1 para Ponto2
96           * @param x do Ponto
97           * @param y do Ponto
98           * @param z do Ponto
99           * @param x2 do Ponto 2
100          * @param y2 do Ponto 2
101          * @param z2 do Ponto 2
102          * @return distancia
103          */
104         public static double distancia(double x, double y, double z, double x2,double y2, double z2){
105             x=x-x2;
106             y=y-y2;
107             z=z-z2;
108             return Math.sqrt(x*x+y*y+z*z);
```

```
109         }
110         private static double[][] multiplicar(double[][] mat1, double[][] mat2) {
111               int i;
112               double[][] mat3 = new double[mat1.length][mat2[0].length];
113               for (int linha = 0; linha < mat1.length; linha++){
114                     for (int coluna = 0; coluna < mat2[0].length; coluna++) {
115                           double acumula_somaprod = 0;
116                           for (i = 0; i < mat1[0].length; i++){
117                                 acumula_somaprod = acumula_somaprod + mat1[linha][i] * mat2[i][coluna];
118                           }
119                           mat3[linha][coluna] = acumula_somaprod;
120                     }
121               }
122               return mat3;
123         }
124
125         public static ParEpistemico criar(int antecedente, int consequente) {
126               ParEpistemicoOrkut par = new ParEpistemicoOrkut();
127               par.setSizeAntecedente(antecedente);
128               for(int i=0;i<antecedente;i++){
129                     par.addAntecedente(Math.random());
130               }
131               par.setSizeConsequente(consequente);
132               return par;
133         }

      }
```

modelo/ ParEpistemicoOrkut.java

```
 1  package br.unicarioca.redesepistemicas.modelo;
 2
 3  import java.awt.Color;
 4  import java.util.ArrayList;
 5  import java.util.List;
 6
 7  public class ParEpistemicoOrkut implements ParEpistemico{
 8      private ArrayList<Double> antecedente = new ArrayList<Double>();
 9      private ArrayList<Double> consequente = new ArrayList<Double>();
10      private int sizeAntecedente;
11      private int sizeConsequente;
12      private Color cor;
13      private String nome = "";
14      private long id=0;
15      public static long idGenerated=0;
16
17      public ParEpistemicoOrkut() {
18          id = idGenerated++;
19      }
20
21      public long getId() {
22          return id;
23      }
24
25      public void setId(long id) {
26          this.id = id;
27      }
28
29      public ArrayList<Double> getAntecedente() {
30          return antecedente;
31      }
32      public void setAntecedente(ArrayList<Double> antecedente) {
33          this.antecedente = antecedente;
34      }
35      public List<Double> getConsequente() {
36          return consequente;
37      }
38      public void setConsequente(ArrayList<Double> consequente) {
39          this.consequente = consequente;
40      }
41
42
43      @Override
44      public Double calcularDiferencaConsequente(ParEpistemico par2) {
45          ParEpistemicoOrkut par = (ParEpistemicoOrkut) par2;
46          par.getConsequente();
47          Double retorno=0.0;
48          for(int i=0;i<par.getConsequente().size();i++){
49              double a = par.getConsequente().get(i);
50              double b = getConsequente().get(i);
51              retorno+=Math.abs(a-b);
52          }
53          return retorno;
```

```
54      }
55
56      @Override
57      public synchronized void addAntecedente(Double d) {
58          if(antecedente.size()==sizeAntecedente){
59              antecedente.clear();
60          }
61          antecedente.add(d);
62      }
63
64      @Override
65      public synchronized void addConsequente(Double d) {
66          if(consequente.size()==sizeConsequente){
67              consequente.clear();
68          }
69          consequente.add(d);
70      }
71      public void setSizeAntecedente(int i) {
72          sizeAntecedente = i;
73      }
74      public void setSizeConsequente(int i){
75          sizeConsequente = i;
76      }
77      @Override
78      public int getSizeAntecedente() {
79          return sizeAntecedente;
80      }
81      @Override
82      public int getSizeConsequente() {
83          return sizeConsequente;
84      }
85      @Override
86      public boolean antecedenteEquals(ParEpistemico parAnt) {
87          for(int i=0;i<parAnt.getDoubleAntecedentes().size();i++){
88              if(!parAnt.getDoubleAntecedentes().get(i).equals(antecedente.get(i))){
89                  return false;
90              }
91          }
92          return true;
93      }
94      @Override
95      public ArrayList<Double> getDoubleAntecedentes() {
96          return antecedente;
97      }
98      @Override
99      public ArrayList<Double> getDoubleConsequentes() {
100         return consequente;
101     }
102
103     @Override
104     public Object clone() throws CloneNotSupportedException {
105         ParEpistemicoOrkut o = (ParEpistemicoOrkut)super.clone();
106         o.antecedente=new ArrayList<Double>();
107         for(double d:this.antecedente){
108             o.antecedente.add(d);//manual copy, may addAll
```

```
109                 }
110                 o.consequente=new ArrayList<Double>();
111                 for(double d:this.consequente){
112                       o.consequente.add(d);//manual copy, may addAll
113                 }
114                 return o;
115           }
116
117           public Color getCor() {
118                 return cor;
119           }
120           public void setCor(Color cor) {
121                 this.cor = cor;
122           }
123           public String getNome() {
124                 return nome;
125           }
126           public void setNome(String nome) {
127                 this.nome = nome;
128           }
129           public String toString(){
130                 return nome;
131           }
132
133
134
135     }
```

modelo/ RedeEpistemica.java

```
1  package br.unicarioca.redesepistemicas.modelo;
2
3  import java.awt.Color;
4  import java.util.ArrayList;
5  import java.util.HashSet;
6  import java.util.Iterator;
7  import java.util.List;
8  import java.util.Set;
9
10 import org.apache.log4j.Logger;
11
12 /**
13  * Representa a rede epistemica, possui a lista dos N agentes da rede.<br>
14  * Guarda configuracoes da rede, o numero de etapas que o algoritmo executou e o proprio algoritmo.<br>
15  */
16 public class RedeEpistemica {
17       private static Logger logger = Logger.getLogger(RedeEpistemica.class);
18       private List<AgenteEpistemico> listAgenteEpistemico = new ArrayList<AgenteEpistemico>();
19       private Set<ComunicacaoListener> listComunicacaoListeners = new
20 HashSet<ComunicacaoListener>();
21       private CicloVidaAgenteListener cicloVidaAgenteListener;
22       private AgenteEpistemicoFactory agenteEpistemicoFactory;
23       private boolean normalizarPesos = false;
24       private boolean ligado = false;
25       private Experimento experimento;
26       private Thread t;
27       private static int etapa;
28       private double theta = 0.0;
29
30       public Experimento getExperimento() {
31             return experimento;
32       }
33
34       public void setExperimento(Experimento experimento) {
35             this.experimento = experimento;
36       }
37
38       /**
39        * Listener de ciclo de vida de um agente<br>
40        * que recebe eventos de quando o agente &eacute; criado ou morto
41        * @param cicloVidaAgenteListener CicloVidaAgenteListener
42        */
43       public void setCicloVidaAgenteListener(CicloVidaAgenteListener cicloVidaAgenteListener) {
44             this.cicloVidaAgenteListener = cicloVidaAgenteListener;
45       }
46
47       /**
48        * Listener de ciclo de vida de um agente
49        * @return CicloVidaAgenteListener
50        */
51       public CicloVidaAgenteListener getCicloVidaAgenteListener() {
52             return cicloVidaAgenteListener;
53       }
```

```
54
55          /**
56           * Listener de comunicacao
57           * @param comunicacaoListener ComunicacaoListener
58           */
59          public void addComunicacaoListener(ComunicacaoListener comunicacaoListener) {
60                listComunicacaoListeners.add(comunicacaoListener);
61          }
62
63          public void removeComunicacaoListener(ComunicacaoListener comunicacaoListener){
64                listComunicacaoListeners.remove(comunicacaoListener);
65          }
66
67          /**
68           * Lista de agentes
69           * @return List AgenteEpistemico
70           */
71          public List<AgenteEpistemico> getListAgenteEpistemico() {
72                return listAgenteEpistemico;
73          }
74
75          public void setListAgenteEpistemico(List<AgenteEpistemico> listAgenteEpistemico) {
76                this.listAgenteEpistemico = listAgenteEpistemico;
77          }
78
79          /**
80           * Faz um passo do algoritmo que consiste em sortear um agente. <br>
81           * Verificar se ele deseja publicar.<br>
82           * pedir ao agente que gere o par a ser publicado.<br>
83           * transmitir o par para todos os outros agentes.<br>
84           */
85          public void fazUmaEtapa(){
86                synchronized (listAgenteEpistemico) {
87                      if(listAgenteEpistemico.size()>1){
88                            int thetaSkipped = 0;
89                            AgenteEpistemico agenteEmissor;
90                            int tentativa=0;
91                            while(true){
92                                  int agente =
93    (int)((double)listAgenteEpistemico.size()*NumeroAleatorio.gerarNumero());
94                                  logger.debug("agente = "  + (agente+1) + " de " + listAgenteEpistemico.size());
95                                  agenteEmissor = listAgenteEpistemico.get(agente);
96                                  //ver se ele quer publicar
97                                  if(agenteEmissor.querPublicar()) break;
98                                  //ver se passou do maximo de tentativas
99                                  if(tentativa>100) return;
100                                 tentativa++;
101                           }
102                           if(agenteEmissor.getMorrerEmXpublicacoes()!=0 &&
103   agenteEmissor.getQtdParComunicado()>agenteEmissor.getMorrerEmXpublicacoes()){
104                                 matarAgente(agenteEmissor);
105                                 if(agenteEpistemicoFactory!=null){
106                                       agenteEpistemicoFactory.criarAgente();
107                                 }
108                           }else{
```

```
109                             //comunicar o par para todos os agentes
110                             ParEpistemico parEpistemico = agenteEmissor.gerarPar();
111                             logger.debug("Agente " + agenteEmissor.getId() + " comunicando...");
112                             for(ComunicacaoListener comunicacaoListener: listComunicacaoListeners){
113                                 comunicacaoListener.comunicadorEscolhido(agenteEmissor);
114                             }
115                             int totAresta = agenteEmissor.getArestas().size();
116                             double lim = (agenteEmissor.getPesoReputacao()/(double)totAresta) * theta;
117                             for(int i=0;i<totAresta;i++){
118                                 Aresta aresta = agenteEmissor.getArestas().get(i);
119                                 AgenteEpistemico receptor = aresta.getReceptor();
120                                 if(receptor == agenteEmissor) continue;
121                                 Double peso = aresta.getPeso();
122                                 if(peso<lim){
123                                     thetaSkipped++;
124                                     continue;
125                                 }
126                                 Double diff = receptor.receberComunicado(parEpistemico, aresta,
127 agenteEmissor);
128                                 logger.debug("diff = "  + diff + " peso " + aresta.getPeso());
129                                 for(ComunicacaoListener comunicacaoListener: listComunicacaoListeners){
130                                     comunicacaoListener.depoisDeComunicar(agenteEmissor, receptor,
131 peso, diff);
132                                 }
133                             }
134                             normalizarPesos();
135                             colorirAgentesDoExperimento(experimento);
136                             etapa++;
137                             logger.info("Etapa " + etapa + ": menor que \u0398 (" + theta + ") = " +
138 thetaSkipped);
139                         }
140                     }//fim do if(listAgenteEpistemico.size()>1){
141             }
142         }
143 
144         public void colorirAgentesDoExperimento(Experimento experimento){
145             //lance do experimento de cores
146             if(experimento!=null){
147                 Set<ParEpistemico> p = experimento.getSetCrencas();
148                 for(AgenteEpistemico agente: listAgenteEpistemico){
149                     agente.crencaMonitorada.clear();
150                     for(ParEpistemico crenca:p){
151                         ParEpistemico opiniao = agente.interpretar(crenca);
152                         double diff = opiniao.calcularDiferencaConsequente(crenca);
153                         if(diff <= experimento.getMaxDiff()){//acredita
154                             agente.crencaMonitorada.add(crenca);
155                             agente.crencaMonitorada.add(crenca);
156                         }else if(diff <= experimento.getMinDiff()){//quase acredita
157                             try {
158                                 ParEpistemico par2 = (ParEpistemico)crenca.clone();
159                                 par2.setCor(new Color(0xEEEEEE));
160                                 agente.crencaMonitorada.add(par2);
161                                 agente.crencaMonitorada.add(crenca);
162                             } catch (CloneNotSupportedException e) {
163                                 e.printStackTrace();
```

```
164                                 }
165                             }else{//Nao acredita
166                                 try {
167                                     ParEpistemico par2 = (ParEpistemico)crenca.clone();
168                                     par2.setCor(new Color(0xEEEEEE));
169                                     agente.crencaMonitorada.add(par2);
170                                     agente.crencaMonitorada.add(par2);
171                                 } catch (CloneNotSupportedException e) {
172                                     e.printStackTrace();
173                                 }
174                             }
175                         }
176                     }
177                 }
178         }
179
180         /**
181          * Normaliza os pesos deixando todos em um valor de 0 ate 1
182          */
183         protected void normalizarPesos(){
184                 {
185                     //validacoes de dados
186                     if(!normalizarPesos) return;
187                     if(listAgenteEpistemico.size()==0) return;
188                 }
189                 logger.info("Normalizando pesos");
190                 //recuperar o maximo
191                 double max = listAgenteEpistemico.get(0).getPesoReputacao();
192                 for(AgenteEpistemico agente:listAgenteEpistemico)
193                     max = Math.max(agente.getPesoReputacao(), max);
194                 logger.debug("max="+max);
195                 //normalizar
196                 for(AgenteEpistemico agente:listAgenteEpistemico){
197                     double newValue = agente.getPesoReputacao()/max;
198                     agente.setPesoReputacao(newValue);
199                 }
200
201         }
202
203         public void setTheta(double theta) {
204                 this.theta = theta;
205         }
206         /**
207          * Inicio da morte,
208          * o processo de morte inicia por este metodo
209          * @param agente
210          */
211         public void matarAgente(AgenteEpistemico agente){
212                 //retirar ele da lista
213                 if(cicloVidaAgenteListener!=null){
214                     cicloVidaAgenteListener.morto(agente);
215                 }
216                 listAgenteEpistemico.remove(agente);
217                 agente.morrer();
218
```

```
219         }
220
221         /**
222          * Insere um agente na rede e apresenta ele a todos os outros agentes
223          * criando as arestas necessarias
224          * @param agenteNovo agente novo
225          * @param pesoAleatorio true para peso aleatorio
226          * @return AgenteEpistemico
227          */
228         public AgenteEpistemico inserirAgente(AgenteEpistemico agenteNovo,boolean pesoAleatorio) {
229             agenteNovo.setRedeEpistemica(this);
230             agenteNovo.setRaio(50);
231             //apresentar a todos
232             if(pesoAleatorio){
233                 for(AgenteEpistemico agenteEpistemico:listAgenteEpistemico){
234                     agenteEpistemico.conhecer(agenteNovo, NumeroAleatorio.gerarNumero());
235                     agenteNovo.conhecer(agenteEpistemico, NumeroAleatorio.gerarNumero());
236                 }
237             }else{
238                 for(AgenteEpistemico agenteEpistemico:listAgenteEpistemico){
239                     agenteEpistemico.conhecer(agenteNovo, 1.0);
240                     agenteNovo.conhecer(agenteEpistemico, 1.0);
241                 }
242             }
243             synchronized (listAgenteEpistemico) {
244                 listAgenteEpistemico.add(agenteNovo);
245             }
246             fireAgenteCriadoEvent(agenteNovo);
247             return agenteNovo;
248         }
249
250         public void fireAgenteCriadoEvent(AgenteEpistemico agenteNovo){
251             if(cicloVidaAgenteListener!=null){
252                 cicloVidaAgenteListener.criado(agenteNovo);
253             }
254         }
255         /**
256          * Quem providencia um agente criado inteiramente
257          * chama o metodo criarAgente
258          * @param agenteEpistemicoFactory
259          */
260         public void setAgenteEpistemicoFactory(AgenteEpistemicoFactory agenteEpistemicoFactory) {
261             this.agenteEpistemicoFactory = agenteEpistemicoFactory;
262         }
263
264         /**
265          * Loop independente da view
266          * @param b
267          */
268         public void ligarLoop(boolean b) {
269             if(true)return;
270             ligado = b;
271             if(t==null){
272                 t = new Thread(){
273                     @Override
```

```
274                         public void run() {
275                             while(true){
276                                 if(ligado){
277                                     try{
278                                         fazUmaEtapa();
279                                     }catch(Exception e){
280
281                                     }
282                                 }
283                             }
284                         }
285                     };
286                     t.start();
287                 }
288         }
289
290         /**
291          * Flag para saber se normalizamos ou nao os pesos
292          * @param normalizarPesos
293          */
294         public void setNormalizarPesos(boolean normalizarPesos) {
295             this.normalizarPesos = normalizarPesos;
296         }
297
298         /**
299          * Flag para saber se normalizamos ou nao os pesos
300          * @return boolean
301          */
302         public boolean isNormalizarPesos() {
303             return normalizarPesos;
304         }
305
306         /**
307          * Retorna quantas vezes o algoritmo foi executado
308          * @return int
309          */
310         public int getEtapa() {
311             return etapa;
            }

            public static void setEtapa(int etapa) {
                RedeEpistemica.etapa = etapa;
            }
        }
```

view/ AgenteListMouseMenu.java

```
1  package br.unicarioca.redesepistemicas.view;
2
3  import java.awt.BorderLayout;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.awt.event.MouseAdapter;
7  import java.awt.event.MouseEvent;
8  import java.awt.event.MouseListener;
9
10 import javax.swing.JFrame;
11 import javax.swing.JList;
12 import javax.swing.JMenuItem;
13 import javax.swing.JOptionPane;
14 import javax.swing.JPopupMenu;
15
16 import br.unicarioca.redesepistemicas.modelo.AgenteEpistemico;
17 import br.unicarioca.redesepistemicas.modelo.RedeEpistemica;
18
19 public class AgenteListMouseMenu extends JPopupMenu {
20       private static final long serialVersionUID = 1765661788449368756L;
21       private JMenuItem verCrencasMenuItem;
22       private JList jList;
23       private AgenteEpistemico agente;
24       public AgenteListMouseMenu(JList jList, final RedeEpistemica redeEpistemica, final
25 RedeEpistemicaView redeEpistemicaView) {
26             this.jList = jList;
27             verCrencasMenuItem = new JMenuItem("Ver Crenças");
28             verCrencasMenuItem.addActionListener(new ActionListener() {
29                   public void actionPerformed(ActionEvent e) {
30                         if(agente.getCrencas().size()==0){
31                               JOptionPane.showMessageDialog(AgenteListMouseMenu.this,"Nenhuma crença");
32                         }else{
33                               JFrame jFrame = new JFrame("Crenças " + agente.getNome());
34                               jFrame.setLayout(new BorderLayout());
35                               jFrame.add(new CrencaView(agente,redeEpistemica,redeEpistemicaView));
36                               jFrame.pack();
37                               jFrame.setVisible(true);
38                         }
39                   }
40             });
41             this.add(verCrencasMenuItem);
42
43             MouseListener popupListener = new PopupListener(this);
44             jList.addMouseListener(popupListener);
45       }
46
47       protected void mostrarMenu(MouseEvent e) {
48             int indexSelecionado = jList.locationToIndex(e.getPoint());
49             agente = (AgenteEpistemico)jList.getModel().getElementAt(indexSelecionado);
50             jList.setSelectedIndex(indexSelecionado);
51             this.show(e.getComponent(), e.getX(), e.getY());
52       }
53
```

```
54          class PopupListener extends MouseAdapter {
55                JPopupMenu popup;
56
57                PopupListener(JPopupMenu popupMenu) {
58                      popup = popupMenu;
59                }
60
61                public void mousePressed(MouseEvent e) {
62                      maybeShowPopup(e);
63                }
64
65                public void mouseReleased(MouseEvent e) {
66                      maybeShowPopup(e);
67                }
68
69                private void maybeShowPopup(MouseEvent e) {
70                      if (e.isPopupTrigger()) {
71                            mostrarMenu(e);
72                      }
73                }
74          }
    }
```

view/ AgenteListPanel.java

```
1  package br.unicarioca.redesepistemicas.view;
2
3  import java.awt.BorderLayout;
4  import java.awt.event.KeyListener;
5  import java.util.ArrayList;
6  import java.util.List;
7
8  import javax.swing.DefaultListModel;
9  import javax.swing.JLabel;
10 import javax.swing.JList;
11 import javax.swing.JPanel;
12 import javax.swing.event.ListSelectionListener;
13
14 import org.apache.log4j.Logger;
15
16 import br.unicarioca.redesepistemicas.bo.InfoListener;
17 import br.unicarioca.redesepistemicas.modelo.AgenteEpistemico;
18 import br.unicarioca.redesepistemicas.modelo.CicloVidaAgenteListener;
19 import br.unicarioca.redesepistemicas.modelo.RedeEpistemica;
20 import br.unicarioca.redesepistemicas.bo.QuickSort;
21
22 public class AgenteListPanel extends JPanel implements CicloVidaAgenteListener, ControladoListener {
23     private static final long serialVersionUID = 1L;
24
25     private static Logger logger = Logger.getLogger(AgenteListPanel.class);
26     private JList jList;
27     private DefaultListModel listModel;
28     private JLabel lblAgentes;
29
30     private RedeEpistemica redeEpistemica;
31     private ArrayList<AgenteEpistemico> listAgentes = new ArrayList<AgenteEpistemico>();
32
33
34     public AgenteListPanel() {
35         this.setLayout(new BorderLayout());
36         lblAgentes = new JLabel("Agentes:        ");
37         listModel = new DefaultListModel();
38         jList = new JList(listModel);
39         this.add(lblAgentes, BorderLayout.NORTH);
40         this.add(jList, BorderLayout.CENTER);
41         Thread t = new Thread() {
42             public void run() {
43                 while (isVisible()) {
44                     try {
45                         Thread.sleep(5000);
46                         refresh();
47                     } catch (Exception e) {
48                     }
49
50                 }
51             }
52         };
53         //t.start();
```

```
54      }
55
56      public synchronized void refresh() {
57          logger.info("Ordenando " + listModel.getSize() + " agentes...");
58          int tot = listModel.getSize() - 1;
59          jList.setIgnoreRepaint(true);
60          if(!redeEpistemica.isNormalizarPesos()){
61              for(int j=0;j<tot;j++){
62                  AgenteEpistemico a = (AgenteEpistemico)listModel.get(j);
63                  a.setPesoReputacao(null);
64              }
65          }
66
67          /*
68          for (int j = tot; j >0; j--) {
69              for (int i = 0; i < j; i++) {
70                  AgenteEpistemico a = (AgenteEpistemico)listModel.get(i);
71                  AgenteEpistemico b = (AgenteEpistemico)listModel.get(i+1);
72                  if(a.getPesoReputacao()<b.getPesoReputacao()){
73                      listModel.set(i, b);
74                      listModel.set(i+1, a);
75                  }
76              }
77          }*/
78          QuickSort.sort(listModel);
79          logger.debug("Calculado " + listAgentes.size());
80
81          //list.revalidate();
82          //list.repaint();
83      }
84
85      @Override
86      public synchronized void addKeyListener(KeyListener l) {
87          jList.addKeyListener(l);
88      }
89
90      public void addListSelectionListener(ListSelectionListener l){
91          jList.addListSelectionListener(l);
92      }
93
94      @Override
95      public void criado(AgenteEpistemico agente) {
96          boolean ok = listAgentes.add(agente);
97          if (!ok)
98              logger.error("Ja existe agente " + agente.getId());
99          listModel.addElement(agente);
100     }
101
102     @Override
103     public void morto(AgenteEpistemico agente) {
104         listAgentes.remove(agente);
105         listModel.removeElement(agente);
106     }
107
108     public List<AgenteEpistemico> getSelecionados(){
```

```
109             int[] arr = jList.getSelectedIndices();
110             List<AgenteEpistemico> res = new ArrayList<AgenteEpistemico>();
111             for(int i=0;i<arr.length;i++){
112                 res.add((AgenteEpistemico)listModel.get(arr[i]));
113             }
114             return res;
115         }
116
117         public JList getJList() {
118             return jList;
119         }
120
121         public int indexOf(AgenteEpistemico agente) {
122             return listModel.indexOf(agente);
123         }
124         public void setRedeEpistemica(RedeEpistemica redeEpistemica) {
125             this.redeEpistemica = redeEpistemica;
126         }
127
128         public void reiniciar() {
129             listModel.clear();
130             listAgentes.clear();
131         }
132
133         @Override
134         public void continuar() {
135             // TODO Auto-generated method stub
136
137         }
138
139         @Override
140         public void criarFotografia() {
141             // TODO Auto-generated method stub
142
143         }
144
145         @Override
146         public void pause() {
147             // TODO Auto-generated method stub
148
149         }
150
151         @Override
152         public void setVelocidade(int i) {
153             // TODO Auto-generated method stub
154
155         }
156
157     }
```

view/ ConfiguracoesPanel.java

```
1  package br.unicarioca.redesepistemicas.view;
2
3  import java.awt.BorderLayout;
4  import java.awt.FlowLayout;
5  import java.awt.GridLayout;
6  import java.io.IOException;
7  import java.io.ObjectInputStream;
8  import java.io.ObjectOutputStream;
9
10 import javax.swing.BorderFactory;
11 import javax.swing.JButton;
12 import javax.swing.JCheckBox;
13 import javax.swing.JLabel;
14 import javax.swing.JPanel;
15 import javax.swing.JSpinner;
16 import javax.swing.JTabbedPane;
17 import javax.swing.JTextField;
18 import javax.swing.SpinnerNumberModel;
19 import javax.swing.SwingConstants;
20
21 /**
22  * TODO criar a configuração interna da rede<br>
23  * se quisermos 5 neuronios na entrada 3 na camada intermediária e 1 na saída<br>
24  * colocaremos no campo o seguinte: 5,3,1
25  * @author Fabio Issamu Oshiro, Leandro Freire
26  *
27  */
28 public class ConfiguracoesPanel extends JTabbedPane {
29       private static final long serialVersionUID = 2L;
30       private JLabel lblQtdAgentes;
31       private JTextField txtQtdAgentes;
32       private JLabel lblMaxDiff;
33       private JTextField txtMaxDiff;
34       private JLabel lblMorrerEmXpublicacoes;
35       private JTextField txtMorrerEmXpublicacoes;
36       private JSpinner spnPassoMax;
37       private JLabel lblPassoMax;
38       private JLabel lblDistanciaMaxRepulsao;
39       private JTextField txtDistanciaMaxRepulsao;
40       private JLabel lblCriarNovoEm;
41       private JTextField txtCriarNovoEm;
42       private JLabel lblSomenteUltimaTeoria;
43       private JCheckBox chkSomenteUltimaTeoria;
44       private JLabel lblFrequencia;
45       private JTextField txtFrequencia;
46       private JLabel lblSnapShot;
47       private JTextField txtSnapShot;
48
49       private JLabel lblPesoAleatorio;
50       private JCheckBox chkPesoAleatorio;
51
52       private JLabel lblDistribuicaoAleatoria;
53       private JCheckBox chkDistribuicaoAleatoria;
```

```
54
55      private JLabel lblEstruturaRede;
56      private JTextField txtEstruturaRede;
57
58      private JLabel lblErrorTolerance;
59      private JTextField txtErrorTolerance;
60
61      private JLabel lblTheta;
62      private JSpinner txtTheta;
63
64      private JButton btnOk;
65
66      private void writeObject(ObjectOutputStream out) throws IOException{
67          out.defaultWriteObject();
68      }
69
70      private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException{
71          in.defaultReadObject();
72          spnPassoMax.setModel(new SpinnerNumberModel());
73      }
74
75      public ConfiguracoesPanel() {
76
77          lblTheta = new JLabel("\u0398:");
78          lblTheta.setHorizontalAlignment(SwingConstants.RIGHT);
79          txtTheta = new JSpinner(new SpinnerNumberModel(0.0, 0.0, 5.0, 0.005));
80
81          lblErrorTolerance = new JLabel("Error Tolerance:");
82          lblErrorTolerance.setHorizontalAlignment(SwingConstants.RIGHT);
83          txtErrorTolerance = new JTextField("0.0001");
84
85          lblEstruturaRede = new JLabel("Estrutura da Rede");
86          txtEstruturaRede = new JTextField("5,3,1");
87
88          lblDistribuicaoAleatoria = new JLabel("Distr. Aleatória:");
89          chkDistribuicaoAleatoria = new JCheckBox();
90          chkDistribuicaoAleatoria.setSelected(true);
91
92          lblSnapShot = new JLabel("PrintScreen:");
93          lblSnapShot.setHorizontalAlignment(SwingConstants.RIGHT);
94          lblSnapShot.setToolTipText("Print a cada x etapas");
95
96          txtSnapShot = new JTextField("0");
97
98          lblQtdAgentes = new JLabel("Qtd. Agentes:");
99          lblQtdAgentes.setHorizontalAlignment(SwingConstants.RIGHT);
100
101         lblMaxDiff = new JLabel("Max Diff:");
102         lblMaxDiff.setToolTipText("Tolerância");
103         lblMaxDiff.setHorizontalAlignment(SwingConstants.RIGHT);
104
105         txtQtdAgentes = new JTextField("256");
106         txtMaxDiff = new JTextField("2.0");
107
108         lblMorrerEmXpublicacoes = new JLabel("Morrer em:");
```

```
109             lblMorrerEmXpublicacoes.setHorizontalAlignment(SwingConstants.RIGHT);
110             lblMorrerEmXpublicacoes.setToolTipText("Morre ao comunicar X vezes");
111             txtMorrerEmXpublicacoes = new JTextField("0");
112
113             lblSomenteUltimaTeoria = new JLabel("Somente último:");
114             lblSomenteUltimaTeoria.setToolTipText("Publica somente o último par gerado");
115             lblSomenteUltimaTeoria.setHorizontalAlignment(SwingConstants.RIGHT);
116
117             lblCriarNovoEm = new JLabel("Criar novo em:");
118             lblCriarNovoEm.setToolTipText("Cria um novo par ao comunicar a cada X vezes");
119             lblCriarNovoEm.setHorizontalAlignment(SwingConstants.RIGHT);
120             txtCriarNovoEm = new JTextField("1");
121
122             chkSomenteUltimaTeoria = new JCheckBox();
123             chkSomenteUltimaTeoria.setSelected(true);
124
125             btnOk = new JButton("Ok");
126
127             lblDistanciaMaxRepulsao = new JLabel("Dist. Max. Repulsão:");
128             lblDistanciaMaxRepulsao.setToolTipText("Limite de distância para a repulsão");
129             txtDistanciaMaxRepulsao = new JTextField("1000");
130
131             spnPassoMax = new JSpinner();
132             spnPassoMax.setValue(15);
133             lblPassoMax = new JLabel("Passo Agente:");
134             lblPassoMax.setHorizontalAlignment(SwingConstants.RIGHT);
135
136             lblPesoAleatorio = new JLabel("Peso aleatório:");
137             lblPesoAleatorio.setHorizontalAlignment(SwingConstants.RIGHT);
138             chkPesoAleatorio = new JCheckBox();
139             chkPesoAleatorio.setSelected(true);
140
141             lblFrequencia = new JLabel("Freq.:");
142             lblFrequencia.setHorizontalAlignment(SwingConstants.RIGHT);
143             txtFrequencia = new JTextField("1.0");
144
145             //layout
146             this.addTab("Rede E.",criarPainelConfRede());
147             this.addTab("Agente E.",criarPainelConfAgente());
148             this.addTab("View",criarPainelConfView());
149
150         }
151
152         public int[] getEstruturaRede(){
153             //Configurar a rede neural
154             String arr[] = getTxtEstruturaRede().getText().split(",");
155             int estruturaRede[] = new int[arr.length];
156             for(int i=0;i<arr.length;i++){
157                 estruturaRede[i] = Integer.parseInt(arr[i]);
158             }
159             return estruturaRede;
160         }
161
162
163         private JPanel criarPainelConfRede(){
```

```
164             JPanel retorno = new JPanel();
165             retorno.setLayout(new BorderLayout());
166             JPanel tabela = new JPanel(new GridLayout(0, 2));
167             tabela.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
168             tabela.add(lblQtdAgentes);
169             tabela.add(txtQtdAgentes);
170             tabela.add(lblSnapShot);
171             tabela.add(txtSnapShot);
172             tabela.add(lblEstruturaRede);
173             tabela.add(txtEstruturaRede);
174             tabela.add(lblErrorTolerance);
175             tabela.add(txtErrorTolerance);
176             tabela.add(lblTheta);
177             tabela.add(txtTheta);
178
179             JPanel sul = new JPanel(new FlowLayout());
180             sul.add(btnOk);
181             retorno.add(tabela, BorderLayout.CENTER);
182             retorno.add(sul, BorderLayout.SOUTH);
183
184             JPanel retornoLayout = new JPanel();
185             retornoLayout.add(retorno, BorderLayout.NORTH);
186             retornoLayout.add(new JPanel(),BorderLayout.CENTER);
187             return retornoLayout;
188         }
189
190         private JPanel criarPainelConfView(){
191             JPanel retorno = new JPanel();
192             retorno.setLayout(new BorderLayout());
193             JPanel tabela = new JPanel(new GridLayout(0, 2));
194             tabela.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
195             tabela.add(lblDistanciaMaxRepulsao);
196             tabela.add(txtDistanciaMaxRepulsao);
197             tabela.add(lblPassoMax);
198             tabela.add(spnPassoMax);
199             tabela.add(lblDistribuicaoAleatoria);
200             tabela.add(chkDistribuicaoAleatoria);
201             retorno.add(tabela, BorderLayout.NORTH);
202             retorno.add(new JPanel(),BorderLayout.CENTER);
203             return retorno;
204         }
205         private JPanel criarPainelConfAgente(){
206             JPanel retorno = new JPanel();
207             retorno.setLayout(new BorderLayout());
208             JPanel tabela = new JPanel(new GridLayout(0, 2));
209             tabela.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
210             tabela.add(lblMaxDiff);
211             tabela.add(txtMaxDiff);
212             tabela.add(lblMorrerEmXpublicacoes);
213             tabela.add(txtMorrerEmXpublicacoes);
214             tabela.add(lblSomenteUltimaTeoria);
215             tabela.add(chkSomenteUltimaTeoria);
216             tabela.add(lblCriarNovoEm);
217             tabela.add(txtCriarNovoEm);
218             tabela.add(lblPesoAleatorio);
```

```
219             tabela.add(chkPesoAleatorio);
220             tabela.add(lblFrequencia);
221             tabela.add(txtFrequencia);
222             retorno.add(tabela, BorderLayout.NORTH);
223             retorno.add(new JPanel(),BorderLayout.CENTER);
224             return retorno;
225         }
226 
227         public JButton getBtnOk() {
228             return btnOk;
229         }
230 
231         public JTextField getTxtQtdAgentes() {
232             return txtQtdAgentes;
233         }
234 
235         public JTextField getTxtMaxDiff() {
236             return txtMaxDiff;
237         }
238 
239         public JTextField getTxtMorrerEmXpublicacoes() {
240             return txtMorrerEmXpublicacoes;
241         }
242 
243         public JCheckBox getChkSomenteUltimaTeoria() {
244             return chkSomenteUltimaTeoria;
245         }
246 
247         public JTextField getTxtDistanciaMaxRepulsao() {
248             return txtDistanciaMaxRepulsao;
249         }
250 
251         public JTextField getTxtCriarNovoEm() {
252             return txtCriarNovoEm;
253         }
254         public JSpinner getSpnPassoMax() {
255             return spnPassoMax;
256         }
257         public JCheckBox getChkPesoAleatorio() {
258             return chkPesoAleatorio;
259         }
260         public JTextField getTxtFrequencia() {
261             return txtFrequencia;
262         }
263         public JTextField getTxtSnapShot() {
264             return txtSnapShot;
265         }
266 
267         /**
268          * @return the lblQtdAgentes
269          */
270         public JLabel getLblQtdAgentes() {
271             return lblQtdAgentes;
272         }
273 
```

```
274         /**
275          * @param lblQtdAgentes the lblQtdAgentes to set
276          */
277         public void setLblQtdAgentes(JLabel lblQtdAgentes) {
278                 this.lblQtdAgentes = lblQtdAgentes;
279         }
280 
281         /**
282          * @return the lblMaxDiff
283          */
284         public JLabel getLblMaxDiff() {
285                 return lblMaxDiff;
286         }
287 
288         /**
289          * @param lblMaxDiff the lblMaxDiff to set
290          */
291         public void setLblMaxDiff(JLabel lblMaxDiff) {
292                 this.lblMaxDiff = lblMaxDiff;
293         }
294 
295         /**
296          * @return the lblMorrerEmXpublicacoes
297          */
298         public JLabel getLblMorrerEmXpublicacoes() {
299                 return lblMorrerEmXpublicacoes;
300         }
301 
302         /**
303          * @param lblMorrerEmXpublicacoes the lblMorrerEmXpublicacoes to set
304          */
305         public void setLblMorrerEmXpublicacoes(JLabel lblMorrerEmXpublicacoes) {
306                 this.lblMorrerEmXpublicacoes = lblMorrerEmXpublicacoes;
307         }
308 
309         /**
310          * @return the lblPassoMax
311          */
312         public JLabel getLblPassoMax() {
313                 return lblPassoMax;
314         }
315 
316         /**
317          * @param lblPassoMax the lblPassoMax to set
318          */
319         public void setLblPassoMax(JLabel lblPassoMax) {
320                 this.lblPassoMax = lblPassoMax;
321         }
322 
323         /**
324          * @return the lblDistanciaMaxRepulsao
325          */
326         public JLabel getLblDistanciaMaxRepulsao() {
327                 return lblDistanciaMaxRepulsao;
328         }
```

```
329
330         /**
331          * @param lblDistanciaMaxRepulsao the lblDistanciaMaxRepulsao to set
332          */
333         public void setLblDistanciaMaxRepulsao(JLabel lblDistanciaMaxRepulsao) {
334                 this.lblDistanciaMaxRepulsao = lblDistanciaMaxRepulsao;
335         }
336
337         /**
338          * @return the lblCriarNovoEm
339          */
340         public JLabel getLblCriarNovoEm() {
341                 return lblCriarNovoEm;
342         }
343
344         /**
345          * @param lblCriarNovoEm the lblCriarNovoEm to set
346          */
347         public void setLblCriarNovoEm(JLabel lblCriarNovoEm) {
348                 this.lblCriarNovoEm = lblCriarNovoEm;
349         }
350
351         /**
352          * @return the lblSomenteUltimaTeoria
353          */
354         public JLabel getLblSomenteUltimaTeoria() {
355                 return lblSomenteUltimaTeoria;
356         }
357
358         /**
359          * @param lblSomenteUltimaTeoria the lblSomenteUltimaTeoria to set
360          */
361         public void setLblSomenteUltimaTeoria(JLabel lblSomenteUltimaTeoria) {
362                 this.lblSomenteUltimaTeoria = lblSomenteUltimaTeoria;
363         }
364
365         /**
366          * @return the lblFrequencia
367          */
368         public JLabel getLblFrequencia() {
369                 return lblFrequencia;
370         }
371
372         /**
373          * @param lblFrequencia the lblFrequencia to set
374          */
375         public void setLblFrequencia(JLabel lblFrequencia) {
376                 this.lblFrequencia = lblFrequencia;
377         }
378
379         /**
380          * @return the lblSnapShot
381          */
382         public JLabel getLblSnapShot() {
383                 return lblSnapShot;
```

```
384         }
385
386         /**
387          * @param lblSnapShot the lblSnapShot to set
388          */
389         public void setLblSnapShot(JLabel lblSnapShot) {
390                 this.lblSnapShot = lblSnapShot;
391         }
392
393         /**
394          * @return the lblPesoAleatorio
395          */
396         public JLabel getLblPesoAleatorio() {
397                 return lblPesoAleatorio;
398         }
399
400         /**
401          * @param lblPesoAleatorio the lblPesoAleatorio to set
402          */
403         public void setLblPesoAleatorio(JLabel lblPesoAleatorio) {
404                 this.lblPesoAleatorio = lblPesoAleatorio;
405         }
406
407         /**
408          * @param txtQtdAgentes the txtQtdAgentes to set
409          */
410         public void setTxtQtdAgentes(JTextField txtQtdAgentes) {
411                 this.txtQtdAgentes = txtQtdAgentes;
412         }
413
414         /**
415          * @param txtMaxDiff the txtMaxDiff to set
416          */
417         public void setTxtMaxDiff(JTextField txtMaxDiff) {
418                 this.txtMaxDiff = txtMaxDiff;
419         }
420
421         /**
422          * @param txtMorrerEmXpublicacoes the txtMorrerEmXpublicacoes to set
423          */
424         public void setTxtMorrerEmXpublicacoes(JTextField txtMorrerEmXpublicacoes) {
425                 this.txtMorrerEmXpublicacoes = txtMorrerEmXpublicacoes;
426         }
427
428         /**
429          * @param spnPassoMax the spnPassoMax to set
430          */
431         public void setSpnPassoMax(JSpinner spnPassoMax) {
432                 this.spnPassoMax = spnPassoMax;
433         }
434
435         /**
436          * @param txtDistanciaMaxRepulsao the txtDistanciaMaxRepulsao to set
437          */
438         public void setTxtDistanciaMaxRepulsao(JTextField txtDistanciaMaxRepulsao) {
```

```
439             this.txtDistanciaMaxRepulsao = txtDistanciaMaxRepulsao;
440         }
441 
442         /**
443          * @param txtCriarNovoEm the txtCriarNovoEm to set
444          */
445         public void setTxtCriarNovoEm(JTextField txtCriarNovoEm) {
446             this.txtCriarNovoEm = txtCriarNovoEm;
447         }
448 
449         /**
450          * @param chkSomenteUltimaTeoria the chkSomenteUltimaTeoria to set
451          */
452         public void setChkSomenteUltimaTeoria(JCheckBox chkSomenteUltimaTeoria) {
453             this.chkSomenteUltimaTeoria = chkSomenteUltimaTeoria;
454         }
455 
456         /**
457          * @param txtFrequencia the txtFrequencia to set
458          */
459         public void setTxtFrequencia(JTextField txtFrequencia) {
460             this.txtFrequencia = txtFrequencia;
461         }
462 
463         /**
464          * @param txtSnapShot the txtSnapShot to set
465          */
466         public void setTxtSnapShot(JTextField txtSnapShot) {
467             this.txtSnapShot = txtSnapShot;
468         }
469 
470         /**
471          * @param chkPesoAleatorio the chkPesoAleatorio to set
472          */
473         public void setChkPesoAleatorio(JCheckBox chkPesoAleatorio) {
474             this.chkPesoAleatorio = chkPesoAleatorio;
475         }
476 
477         /**
478          * @param btnOk the btnOk to set
479          */
480         public void setBtnOk(JButton btnOk) {
481             this.btnOk = btnOk;
482         }
483 
484 
485         public JCheckBox getChkDistribuicaoAleatoria() {
486             return chkDistribuicaoAleatoria;
487         }
488 
489         public void setChkDistribuicaoAleatoria(JCheckBox chkDistribuicaoAleatoria) {
490             this.chkDistribuicaoAleatoria = chkDistribuicaoAleatoria;
491         }
492 
493         public JLabel getLblDistribuicaoAleatoria() {
```

```
494             return lblDistribuicaoAleatoria;
495         }
496
497         public void setLblDistribuicaoAleatoria(JLabel lblDistribuicaoAleatoria) {
498             this.lblDistribuicaoAleatoria = lblDistribuicaoAleatoria;
499         }
500
501         public JLabel getLblEstruturaRede() {
502             return lblEstruturaRede;
503         }
504
505         public void setLblEstruturaRede(JLabel lblEstruturaRede) {
506             this.lblEstruturaRede = lblEstruturaRede;
507         }
508
509         public JTextField getTxtEstruturaRede() {
510             return txtEstruturaRede;
511         }
512
513         public void setTxtEstruturaRede(JTextField txtEstruturaRede) {
514             this.txtEstruturaRede = txtEstruturaRede;
515         }
516
517         public JLabel getLblErrorTolerance() {
518             return lblErrorTolerance;
519         }
520
521         public void setLblErrorTolerance(JLabel lblErrorTolerance) {
522             this.lblErrorTolerance = lblErrorTolerance;
523         }
524
525         public JTextField getTxtErrorTolerance() {
526             return txtErrorTolerance;
527         }
528
529         public void setTxtErrorTolerance(JTextField txtErrorTolerance) {
530             this.txtErrorTolerance = txtErrorTolerance;
531         }
532
533         public void setSpnTheta(JSpinner txtTheta) {
534             this.txtTheta = txtTheta;
535         }
536
537         public JSpinner getSpnTheta() {
538             return txtTheta;
539         }
540
541         public JLabel getLblTheta() {
542             return lblTheta;
543         }
544
545         public void setLblTheta(JLabel lblTheta) {
546             this.lblTheta = lblTheta;
547         }
548     }
```

view/ ControladoListener.java

```
1  package br.unicarioca.redesepistemicas.view;
2
3  public interface ControladoListener {
4
5          void reiniciar();
6
7          void pause();
8
9          void continuar();
10
11         void setVelocidade(int i);
12
13         void criarFotografia();
14
15 }
```

view/ ControlePanel.java

```
1  package br.unicarioca.redesepistemicas.view;
2
3  import java.awt.FlowLayout;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.util.ArrayList;
7  import java.util.List;
8
9  import javax.swing.JButton;
10 import javax.swing.JPanel;
11 import javax.swing.JSlider;
12 import javax.swing.event.ChangeEvent;
13 import javax.swing.event.ChangeListener;
14 /**
15  * Objetos controlados por este controle:
16  * AgenteListPanel e RedeEpistemeicaView
17  */
18 public class ControlePanel extends JPanel{
19       private static final long serialVersionUID = 1L;
20       private JButton pause;
21       private JButton play;
22       private JButton zerar;
23       private JButton fotografar;
24       /**
25        * Coisas controladas por este painel,
26        * imaginava que outros objetos seriam pausados por este unico controlador
27        */
28       private List<ControladoListener> listControladoListener = new ArrayList<ControladoListener>();
29       private JSlider slider;
30       public ControlePanel() {
31             this.setLayout(new FlowLayout());
32             fotografar = new JButton("Fotografar");
33             zerar = new JButton("Zerar");
34             pause = new JButton("Pause");
35             play = new JButton("Play");
36
37             slider = new JSlider();
38             slider.setValue(100);
39             slider.setName("Vel.");
40             this.add(fotografar);
41             this.add(zerar);
42             this.add(pause);
43             this.add(play);
44             this.add(slider);
45
46             fotografar.addActionListener(new ActionListener(){
47                   public void actionPerformed(ActionEvent e){
48                         for(ControladoListener controladoListener:listControladoListener)
49                               controladoListener.criarFotografia();
50                   }
51             });
52
53             zerar.addActionListener(new ActionListener(){
```

```
54                      public void actionPerformed(ActionEvent e) {
55                          for(ControladoListener controladoListener:listControladoListener)
56                              controladoListener.reiniciar();
57                      }
58                  });
59                  pause.addActionListener(new ActionListener(){
60                      public void actionPerformed(ActionEvent e) {
61                          for(ControladoListener controladoListener:listControladoListener)
62                              controladoListener.pause();
63                      }
64                  });
65                  play.addActionListener(new ActionListener(){
66                      public void actionPerformed(ActionEvent e) {
67                          for(ControladoListener controladoListener:listControladoListener)
68                              controladoListener.continuar();
69                      }
70                  });
71                  slider.addChangeListener(new ChangeListener(){
72                      @Override
73                      public void stateChanged(ChangeEvent e) {
74                          for(ControladoListener controladoListener:listControladoListener)
75                              controladoListener.setVelocidade(slider.getMaximum() - slider.getValue());
76
77                      }
78                  });
79          }
80
81          /**
82           * Objeto que sera controlado por este controle
83           * @param controladoListener
84           */
85          public void addControlado(ControladoListener controladoListener) {
86              listControladoListener.add(controladoListener);
87          }
88
    }
```

view/ CrencaJTable.java

```
1  package br.unicarioca.redesepistemicas.view;
2
3  import java.awt.Color;
4  import java.awt.Component;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.util.ArrayList;
8  import java.util.List;
9
10 import javax.swing.AbstractCellEditor;
11 import javax.swing.BorderFactory;
12 import javax.swing.JButton;
13 import javax.swing.JColorChooser;
14 import javax.swing.JDialog;
15 import javax.swing.JLabel;
16 import javax.swing.JOptionPane;
17 import javax.swing.JTable;
18 import javax.swing.event.TableModelEvent;
19 import javax.swing.event.TableModelListener;
20 import javax.swing.table.TableCellEditor;
21 import javax.swing.table.TableCellRenderer;
22
23 import org.apache.log4j.Logger;
24
25 import br.unicarioca.redesepistemicas.modelo.ParEpistemico;
26
27 public class CrencaJTable extends JTable{
28     private static final long serialVersionUID = 1L;
29     private static final int PAR_COLUMN=2;
30     private static final int COLOR_COLUMN=1;
31     private static final int MONITORA_COLUMN=0;
32     private static Logger logger = Logger.getLogger(CrencaJTable.class);
33     private static int shiftCol2Right = 3;
34     private CrencaTableModel crencaTableModel;
35     private ParEpistemico parModelo;
36     public CrencaJTable(ParEpistemico parModelo, boolean consequenteAtual) {
37         this.parModelo = parModelo;
38         crencaTableModel = new CrencaTableModel(parModelo, consequenteAtual);
39         this.setModel(crencaTableModel);
40         this.getColumnModel().getColumn(COLOR_COLUMN).setCellRenderer(new ColorCellRenderer());
41         this.getColumnModel().getColumn(COLOR_COLUMN).setCellEditor(new ColorCellEditor());
42         this.getColumnModel().getColumn(COLOR_COLUMN).setMaxWidth(20);//cor
43         this.getColumnModel().getColumn(PAR_COLUMN).setMinWidth(200);//nome
44         this.getColumnModel().getColumn(MONITORA_COLUMN).setMaxWidth(20);//checkbox?
45         crencaTableModel.addTableModelListener(new TableModelListener() {
46             @Override
47             public void tableChanged(TableModelEvent e) {
48                 if(e.getColumn()==COLOR_COLUMN){
49                     logger.info("TableChanged coluna da cor");
50                     ParEpistemico par = getParInRow(e.getFirstRow());
51                     if(par!=null){
52                         logger.info("atualizando a cor do par...");
53                         par.setCor(getColorInRow(e.getFirstRow()));
```

```
54                                  }
55                              }
56                          }
57                      });
58          }
59
60          public String getNomeInRow(int row) {
61                  Object obj = crencaTableModel.getValueAt(row,PAR_COLUMN);
62                  return obj.toString();
63          }
64
65          public ParEpistemico getParInRow(int row) {
66                  Object obj = crencaTableModel.getValueAt(row,PAR_COLUMN);
67                  if(obj!=null){
68                          return (ParEpistemico)obj;
69                  }else{
70                          return null;
71                  }
72          }
73
74          public List<ParEpistemico> getPares() {
75                  List<ParEpistemico> retorno = new ArrayList<ParEpistemico>();
76                  try {
77                          int linhas = crencaTableModel.getRowCount();
78                          for (int i = 0; i < linhas; i++) {
79                                  ParEpistemico par = (ParEpistemico) crencaTableModel.getValueAt(i,PAR_COLUMN);
80                                  retorno.add(par);
81                          }
82                  } catch (Exception e) {
83                          e.printStackTrace();
84                          JOptionPane.showMessageDialog(this,"Erro ao gerar pares: " + e.getMessage());
85                  }
86                  return retorno;
87          }
88          public boolean isChecked(int row) {
89
90                  return (Boolean) crencaTableModel.getValueAt(row,MONITORA_COLUMN);
91          }
92
93          public Color getColorInRow(int row) {
94                  Object obj = crencaTableModel.getValueAt(row,COLOR_COLUMN);
95                  if(obj==null || !(obj instanceof Color)){
96                          logger.info("sem cor " + row);
97                          return Color.BLACK;
98                  }else{
99                          logger.info("com cor " + row);
100                 }
101                 return (Color)obj;
102         }
103
104         /**
105          * Remove todos os caras
106          */
107         public void clear() {
108                 int r = crencaTableModel.getRowCount();
```

```
109             for(int i=0;i<r;i++){
110                 crencaTableModel.removeRow(0);
111             }
112         }
113 
114         public void addRow(ParEpistemico parR, Color color, Boolean boolean1, ParEpistemico parR2) {
115             Object objects[];
116             if(parR2!=null){
117                 objects = new
118 Object[shiftCol2Right+parR.getSizeAntecedente()+parR.getSizeConsequente()+parR2.getSizeConsequente()];
119             }else{
120                 objects = new Object[shiftCol2Right+parR.getSizeAntecedente()+parR.getSizeConsequente()];
121             }
122             logger.info("Criando uma linha com "+objects.length+" colunas");
123             objects[MONITORA_COLUMN] = boolean1;
124             objects[COLOR_COLUMN]=color;
125             objects[PAR_COLUMN]=parR;
126             int i=0;
127             for(int j=0;j<parR.getSizeAntecedente();j++){
128                 objects[i+shiftCol2Right] = parR.getDoubleAntecedentes().get(j);
129                 i++;
130             }
131             for(int j=0;j<parR.getSizeConsequente();j++){
132                 objects[i+shiftCol2Right] = parR.getDoubleConsequentes().get(j);
133                 i++;
134             }
135             if(parR2!=null){
136                 for(int j=0;j<parR2.getSizeConsequente();j++){
137                     objects[i+shiftCol2Right] = parR2.getDoubleConsequentes().get(j);
138                     i++;
139                 }
140             }
141             crencaTableModel.addRow(objects);
142         }
143 
144         public void addRow(ParEpistemico par) {
145             addRow(par, par.getCor(), false, null);
146         }
147 
148         public void addRow(ParEpistemico par, ParEpistemico parR2) {
149             addRow(par, par.getCor(), false, parR2);
150         }
151 
152         public void removeRow(int row) {
153             crencaTableModel.removeRow(row);
154         }
155 
156         public void addRow() {
157             Object obj[] = new
158 Object[parModelo.getSizeAntecedente()+parModelo.getSizeConsequente()+shiftCol2Right];
159             obj[0] = new Boolean(false);
160             obj[1] = new Color(33,23,54);
161             obj[2] = "";
162             logger.info("Criando linha com " + obj.length + " colunas");
163             for (int i = shiftCol2Right; i < obj.length; i++) {
```

```
164                     obj[i] = Math.random();
165                 }
166                 crencaTableModel.addRow(obj);
167         }
168 
169         class ColorCellRenderer extends JLabel implements TableCellRenderer{
170             private static final long serialVersionUID = 1L;
171             @Override
172             public Component getTableCellRendererComponent(JTable table, Object color, boolean isSelectd,
173 boolean hasFocus, int row, int col) {
174                     Color newColor =(Color) color;
175                     setBackground(newColor);
176                     setOpaque(true);
177                     setBorder(BorderFactory.createMatteBorder(2,5,2,5,table.getBackground()));
178                     table.setCellSelectionEnabled(false);
179                     table.setColumnSelectionAllowed(false);
180                     return this;
181                 }
182         }
183         class ColorCellEditor extends AbstractCellEditor implements TableCellEditor, ActionListener{
184             private static final long serialVersionUID = 1L;
185             Color currentColor;
186           JButton button;
187           JColorChooser colorChooser;
188           JDialog dialog;
189           protected static final String EDIT = "edit";
190 
191           public ColorCellEditor() {
192             button = new JButton();
193             button.setActionCommand(EDIT);
194             button.addActionListener(this);
195             button.setBorderPainted(false);
196             colorChooser = new JColorChooser();
197             colorChooser.remove(1);
198             dialog = JColorChooser.createDialog(button, "Escolha uma cor para a crença", true,  //modal
199                                         colorChooser,
200                                         this,  //OK button handler
201                                         null); //no CANCEL button handler
202           }
203 
204           /**
205            * Handles events from the editor button and from
206            * the dialog's OK button.
207            */
208           public void actionPerformed(ActionEvent e) {
209             if (EDIT.equals(e.getActionCommand())) {
210               //The user has clicked the cell, so
211               //bring up the dialog.
212               button.setBackground(currentColor);
213               colorChooser.setColor(currentColor);
214               dialog.setVisible(true);
215 
216               //Make the renderer reappear.
217               fireEditingStopped();
218 
```

```
219                 } else { //User pressed dialog's "OK" button.
220                     currentColor = colorChooser.getColor();
221                 }
222             }
223 
224             //Implement the one CellEditor method that AbstractCellEditor doesn't.
225             public Object getCellEditorValue() {
226                 return currentColor;
227             }
228 
229             //Implement the one method defined by TableCellEditor.
230             public Component getTableCellEditorComponent(JTable table, Object value, boolean isSelected, int
231     row, int column) {
232                 currentColor = (Color)value;
233                 return button;
234             }
235 
236         }
237 
238         public Class<?> getColumnClass(int c) {
239             Object obj = getValueAt(0, c);
240             if(obj==null){
241                 logger.warn("Colum class == null " + c);
242                 if(COLOR_COLUMN==c){
243                     return Color.class;
244                 }else if(MONITORA_COLUMN==c){
245                     return Boolean.class;
246                 }else{
247                     return String.class;
248                 }
249             }
250             return obj.getClass();
251         }
```

```
    }
```

view/ CrencaTreinarView.java

```
1  package br.unicarioca.redesepistemicas.view;
2
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.FlowLayout;
6  import java.awt.event.ActionEvent;
7  import java.awt.event.ActionListener;
8  import java.awt.event.KeyEvent;
9  import java.awt.event.KeyListener;
10 import java.util.List;
11
12 import javax.swing.JButton;
13 import javax.swing.JLabel;
14 import javax.swing.JPanel;
15 import javax.swing.JScrollPane;
16 import javax.swing.JTextField;
17
18 import org.apache.log4j.Logger;
19
20 import br.unicarioca.redesepistemicas.modelo.AgenteEpistemico;
21 import br.unicarioca.redesepistemicas.modelo.ParEpistemico;
22 import br.unicarioca.redesepistemicas.modelo.ParEpistemicoFactory;
23
24 public class CrencaTreinarView extends JPanel {
25     private static final long serialVersionUID = 1L;
26     private static final Logger logger = Logger.getLogger(CrencaTreinarView.class);
27     static class Holder {
28         static CrencaTreinarView instance = new CrencaTreinarView();
29     }
30     public static CrencaTreinarView getInstance() {
31         return Holder.instance;
32     }
33
34     private CrencaJTable jTable;
35     private JButton btnNovaCrenca;
36     private JButton btnTreinarSelecionados;
37     private JTextField txtQtdTreino;
38     private JLabel lblQtdTreino;
39     private AgenteListPanel agenteListPanel;
40
41     public void setAgenteListPanel(AgenteListPanel agenteListPanel) {
42         this.agenteListPanel = agenteListPanel;
43     }
44
45     private CrencaTreinarView() {
46         btnNovaCrenca = new JButton("Nova Crença");
47         txtQtdTreino = new JTextField("1000");
48         lblQtdTreino = new JLabel("Qtd. Treino: ");
49         btnTreinarSelecionados = new JButton("Treinar Selecionados");
50         jTable = new CrencaJTable(MainFrame.parModelo, false);
51
52         jTable.addKeyListener(new KeyListener() {
53             public void keyPressed(KeyEvent e) {
```

```
54                          int row = jTable.getSelectedRow();
55                          if (e.getKeyCode() == KeyEvent.VK_DELETE && row > -1) {
56                                jTable.removeRow(row);
57                          }
58                    }
59
60                    // Dont work over there
61                    public void keyReleased(KeyEvent e) {
62                    }
63
64                    // Dont work over there
65                    public void keyTyped(KeyEvent e) {
66                    }
67              });
68
69              btnNovaCrenca.addActionListener(new ActionListener() {
70                    @Override
71                    public void actionPerformed(ActionEvent e) {
72                          criarNovaLinha();
73                    }
74              });
75
76              btnTreinarSelecionados.addActionListener(new ActionListener() {
77                    @Override
78                    public void actionPerformed(ActionEvent e) {
79                          treinarAgentesSelecionados();
80                    }
81              });
82
83              JPanel sul = new JPanel(new FlowLayout());
84              sul.add(lblQtdTreino);
85              sul.add(txtQtdTreino);
86              sul.add(btnNovaCrenca);
87              sul.add(btnTreinarSelecionados);
88
89              this.setLayout(new BorderLayout());
90              this.add(sul, BorderLayout.SOUTH);
91              this.add(new JScrollPane(jTable), BorderLayout.CENTER);
92        }
93
94        private void criarNovaLinha() {
95              ParEpistemico par = ParEpistemicoFactory.criar(MainFrame.parModelo.getSizeAntecedente(),
96  MainFrame.parModelo.getSizeConsequente());
97              par.setCor(Color.WHITE);
98              int i;
99              for(i=0;i<par.getSizeAntecedente();i++){
100                   par.addAntecedente(Math.random());
101             }
102             for(i=0;i<par.getSizeConsequente();i++){
103                   par.addConsequente(Math.random());
104             }
105             jTable.addRow(par);
106       }
107
108       private void treinarAgentesSelecionados() {
```

```
109             List<ParEpistemico> pares = getParesEpistemicos();
110             List<AgenteEpistemico> agentes = agenteListPanel.getSelecionados();
111             int qtd = Integer.parseInt(txtQtdTreino.getText());
112             for (AgenteEpistemico agente : agentes) {
113                 agente.treinar(pares, qtd);
114                 agente.addCrencas(pares);
115             }
116         }
117
118         private List<ParEpistemico> getParesEpistemicos() {
119             return jTable.getPares();
120         }
121
    }
```

view/ CrencaView.java

```
1  package br.unicarioca.redesepistemicas.view;
2
3  import java.awt.BorderLayout;
4  import java.awt.FlowLayout;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.util.Iterator;
8  import java.util.List;
9  import java.util.Set;
10
11 import javax.swing.JButton;
12 import javax.swing.JPanel;
13 import javax.swing.JScrollPane;
14 import javax.swing.JTextField;
15
16 import org.apache.log4j.Logger;
17
18 import br.unicarioca.redesepistemicas.modelo.AgenteEpistemico;
19 import br.unicarioca.redesepistemicas.modelo.Experimento;
20 import br.unicarioca.redesepistemicas.modelo.ParEpistemico;
21 import br.unicarioca.redesepistemicas.modelo.RedeEpistemica;
22
23 /**
24  * Visualiza a Crenca
25  * @author Fabio Issamu Oshiro, Leandro Freire
26  */
27 public class CrencaView extends JPanel{
28       private static final long serialVersionUID = 1L;
29       private static final Logger logger = Logger.getLogger(CrencaView.class);
30       private CrencaJTable jTable;
31       private JButton btnAtualizar;
32       private AgenteEpistemico agente;
33
34       private Experimento experimento;
35
36       public CrencaView(Set<ParEpistemico> crencas, RedeEpistemica redeEpistemica,
37 RedeEpistemicaView redeEpistemicaView){
38             logger.info("CrencaView abrindo monitor...");
39             experimento = new Experimento();
40             jTable = new CrencaJTable(crencas.iterator().next(), false);
41
42             JPanel sul = criarPainelMonitorar(redeEpistemica, redeEpistemicaView);
43
44             this.setLayout(new BorderLayout());
45             popularTabela(crencas);
46             this.add(sul, BorderLayout.SOUTH);
47             this.add(new JScrollPane(jTable),BorderLayout.CENTER);
48       }
49
50       public JPanel criarPainelMonitorar(final RedeEpistemica redeEpistemica, final RedeEpistemicaView
51 redeEpistemicaView){
52             JButton btnMonitorarCrencas;
53             final JTextField txtMaxDiff;
```

```
54              final JTextField txtMinDiff;
55              txtMaxDiff = new JTextField("0.002");
56              txtMinDiff = new JTextField("0.007");
57              btnMonitorarCrencas = new JButton("Monitorar");
58              btnMonitorarCrencas.addActionListener(new ActionListener(){
59                  @Override
60                  public void actionPerformed(ActionEvent e) {
61                      Experimento experimento = new Experimento();
62                      double maxDiff = Double.parseDouble(txtMaxDiff.getText());
63                      double minDiff = Double.parseDouble(txtMinDiff.getText());
64                      experimento.setMaxDiff(maxDiff);
65                      experimento.setMinDiff(minDiff);
66                      for(int row= 0; row < jTable.getRowCount(); row++){
67                          if(jTable.isChecked(row)){
68                              ParEpistemico par =(ParEpistemico) jTable.getParInRow(row);
69                              par.setNome(jTable.getNomeInRow(row));
70                              par.setCor(jTable.getColorInRow(row));
71                              experimento.addCrenca(par);
72                          }
73                      }
74                      redeEpistemica.setExperimento(experimento);
75                      redeEpistemica.colorirAgentesDoExperimento(experimento);
76                      redeEpistemicaView.refresh();
77                  }
78              });
79              JPanel sul = new JPanel(new FlowLayout());
80              sul.add(txtMaxDiff);
81              sul.add(txtMinDiff);
82              sul.add(btnMonitorarCrencas);
83              return sul;
84          }
85
86          public Experimento getExperimento(){
87              return experimento;
88          }
89
90          public void setExperimento(Experimento experimento){
91              this.experimento = experimento;
92          }
93
94          public CrencaView(AgenteEpistemico agente, RedeEpistemica redeEpistemica, RedeEpistemicaView
95      redeEpistemicaView) {
96              this.agente = agente;
97              jTable = new CrencaJTable(agente.getCrencas().get(0), true);
98              btnAtualizar = new JButton("Atualizar");
99              popularTabela();
100
101             btnAtualizar.addActionListener(new ActionListener(){
102                 @Override
103                 public void actionPerformed(ActionEvent e) {
104                     atualizar();
105                 }
106             });
107
108             this.setLayout(new BorderLayout());
```

```
109             JPanel sul = criarPainelMonitorar(redeEpistemica, redeEpistemicaView);
110             sul.add(btnAtualizar);
111
112             this.add(new JScrollPane(jTable),BorderLayout.CENTER);
113             this.add(sul,BorderLayout.SOUTH);
114         }
115
116         private void popularTabela(){
117             List<ParEpistemico> pares = agente.getCrencas();
118             for(ParEpistemico par:pares){
119                 ParEpistemico parR = agente.interpretar(par);
120                 jTable.addRow(par, parR);
121             }
122         }
123
124         private void popularTabela(Set<ParEpistemico> crencas){
125             if(crencas == null) return;
126             Iterator<ParEpistemico> it = crencas.iterator();
127             while(it.hasNext()){
128                 ParEpistemico par = it.next();
129                 jTable.addRow(par,par.getCor(),new Boolean(false), null);
130             }
131         }
132
133         private void atualizar(){
134             jTable.clear();
            popularTabela();
        }
    }
```

view/ GraficoPotenciaFactory.java

```
1  package br.unicarioca.redesepistemicas.view;
2
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import java.awt.image.BufferedImage;
6  import java.util.ArrayList;
7  import java.util.List;
8
9  import br.unicarioca.redesepistemicas.modelo.AgenteEpistemico;
10 import br.unicarioca.redesepistemicas.modelo.RedeEpistemica;
11
12 public class GraficoPotenciaFactory{
13     public static BufferedImage create(RedeEpistemica redeEpistemica, int totFaixas, int w, int h){
14         List<AgenteEpistemico> listAgenteEpistemicos = redeEpistemica.getListAgenteEpistemico();
15         if(listAgenteEpistemicos.size()==0) return null;
16         double max, min;
17         max = min=listAgenteEpistemicos.get(0).getPesoReputacao();
18         for(AgenteEpistemico agente: listAgenteEpistemicos){
19             max = Math.max(max, agente.getPesoReputacao()+0.00001);
20             min = Math.min(min, agente.getPesoReputacao());
21         }
22         double intervalo = (max-min) / (double)totFaixas;
23         double last = min;
24         List<Faixa> faixas = new ArrayList<Faixa>();
25         //System.out.println("intervalo " + intervalo);
26         for(int i=0;i<totFaixas;i++){
27             Faixa faixa = new Faixa();
28             faixa.min = last;
29             faixa.max = last = min + intervalo + intervalo*i;
30             faixas.add(faixa);
31             //System.out.println(" min " + faixa.min + " max " + faixa.max);
32         }
33         int maxTotal = 0;
34         for(AgenteEpistemico agente: listAgenteEpistemicos){// O(a*f)
35             for(Faixa faixa: faixas){
36                 if(faixa.min<=agente.getPesoReputacao() && agente.getPesoReputacao() < faixa.max){
37                     faixa.total++;
38                     maxTotal = Math.max(maxTotal, faixa.total);
39                     break;
40                 }
41             }
42         }
43
44         BufferedImage bi = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
45         intervalo = w/(float)totFaixas;
46         float uy = (h-20)/(float)maxTotal;
47         Graphics gra = bi.getGraphics();
48         gra.setColor(new Color(0xEEEEEE));
49         gra.fillRect(0, 0, w, h);
50         int x1 = 0;
51         int x2 = (int)intervalo/2;
52         int y1 = (int)(faixas.get(0).total * uy);
53         //System.out.println("uy = " + uy);
```

```
54              for(Faixa faixa:faixas){
55                  //System.out.println( faixa.total + "\t"  + maxTotal + " faixa [" + faixa.min + " " + faixa.max +
56  "]");
57                  int y2 = (int)(faixa.total * uy);
58                  gra.setColor(Color.BLACK);
59                  if(x1!=0)
60                  gra.drawLine(x1, h - y1 - 8, x2, h - y2 - 8);
61                  y1 = y2;
62                  x1 = x2;
63                  x2+=(int)intervalo;
64                  gra.setColor(Color.LIGHT_GRAY);
65                  gra.drawLine(x1, 0, x1, h);
66                  gra.setColor(Color.BLACK);
67                  gra.drawString("" + (int)faixa.min, x1, h);
68                  gra.drawString("" + (int)faixa.total, x1-8, h - y1 - 10);
69              }
70              return bi;
71          }
72  }
73  class Faixa{
74          int total = 0;
75          double max, min;
    }
```

view/ MainFrame.java

```
1  package br.unicarioca.redesepistemicas.view;
2
3  import java.awt.BorderLayout;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.awt.event.FocusEvent;
7  import java.awt.event.FocusListener;
8  import java.awt.event.KeyEvent;
9  import java.awt.event.KeyListener;
10 import java.awt.event.WindowEvent;
11 import java.awt.event.WindowListener;
12 import java.io.File;
13 import java.util.HashSet;
14 import java.util.List;
15 import java.util.Set;
16
17 import javax.swing.BorderFactory;
18 import javax.swing.JFileChooser;
19 import javax.swing.JFrame;
20 import javax.swing.JLabel;
21 import javax.swing.JOptionPane;
22 import javax.swing.JPanel;
23 import javax.swing.JScrollPane;
24 import javax.swing.SwingConstants;
25 import javax.swing.event.ChangeEvent;
26 import javax.swing.event.ChangeListener;
27 import javax.swing.event.ListSelectionEvent;
28 import javax.swing.event.ListSelectionListener;
29
30 import org.apache.commons.io.FileUtils;
31 import org.apache.log4j.Logger;
32
33 import br.unicarioca.redesepistemicas.bo.GenericDao;
34 import br.unicarioca.redesepistemicas.bo.InfoListener;
35 import br.unicarioca.redesepistemicas.bo.SalvarSnapShoot;
36 import br.unicarioca.redesepistemicas.dao.RedeDao;
37 import br.unicarioca.redesepistemicas.modelo.AgenteEpistemico;
38 import br.unicarioca.redesepistemicas.modelo.AgenteEpistemicoFactory;
39 import br.unicarioca.redesepistemicas.modelo.CicloVidaAgenteListener;
40 import br.unicarioca.redesepistemicas.modelo.NumeroAleatorio;
41 import br.unicarioca.redesepistemicas.modelo.ParEpistemico;
42 import br.unicarioca.redesepistemicas.modelo.ParEpistemicoFactory;
43 import br.unicarioca.redesepistemicas.modelo.RedeEpistemica;
44
45 /**
46  * Janela principal do sistema
47  */
48 public class MainFrame extends JFrame implements InfoListener,WindowListener,
49 CicloVidaAgenteListener,AgenteEpistemicoFactory{
50       private static final long serialVersionUID = 1L;
51       private static Logger logger = Logger.getLogger(MainFrame.class);
52       private static MainFrame instance;
53       private RedeEpistemicaView redeEpistemicaView;
```

```
54      private RedeEpistemica redeEpistemica;
55      private AgenteListPanel agenteListPanel;
56      private ControlePanel controlePanel;
57      private MenuPrincipal menuPrincipal;
58      private GenericDao<ConfiguracoesPanel> configuracoesPanelDao= new
59  GenericDao<ConfiguracoesPanel>();
60      private ConfiguracoesPanel configuracoesPanel = null;
61      private JLabel sysInfo = new JLabel("Redes Epistêmicas");
62      public static ParEpistemico parModelo = ParEpistemicoFactory.criar(5,1);
63      private CrencaView crencaView;
64      public MainFrame() {
65          this.setTitle("IndraNet 1.2 - Simulador de Redes Epistêmicas");
66          //instancias
67          redeEpistemica = new RedeEpistemica();
68          redeEpistemicaView = new RedeEpistemicaView(redeEpistemica);
69          controlePanel = new ControlePanel();
70          agenteListPanel = new AgenteListPanel();
71          menuPrincipal = new MenuPrincipal();
72          configuracoesPanel = configuracoesPanelDao.getLast(ConfiguracoesPanel.class);
73          if(configuracoesPanel==null){
74              configuracoesPanel = new ConfiguracoesPanel();
75          }
76          AgenteListMouseMenu agenteListMouseMenu = new
77  AgenteListMouseMenu(agenteListPanel.getJList(),redeEpistemica,redeEpistemicaView);
78          //configuracoes
79          SalvarSnapShoot.getInstance().setConfiguracoesPanel(configuracoesPanel);
80          CrencaTreinarView crencaTreinarView = CrencaTreinarView.getInstance();
81          crencaTreinarView.setAgenteListPanel(agenteListPanel);
82
83          redeEpistemica.setCicloVidaAgenteListener(this);
84          redeEpistemica.setAgenteEpistemicoFactory(this);
85          redeEpistemicaView.setAgenteEpistemicoFactory(this);
86          redeEpistemicaView.setAgenteListPanel(agenteListPanel);
87          agenteListPanel.setRedeEpistemica(redeEpistemica);
88          controlePanel.addControlado(redeEpistemicaView);
89          controlePanel.addControlado(agenteListPanel);
90          controlePanel.addControlado(new ControladoListener() {
91              public void setVelocidade(int i) {}
92              public void reiniciar() {
93                  clean();
94              }
95              public void pause() {}
96              public void criarFotografia() {}
97              public void continuar() {}
98          });
99          menuPrincipal.getGraficoPotencia().addActionListener(new ActionListener() {
100             public void actionPerformed(ActionEvent e) {
101                 redeEpistemicaView.ligaDesligaGraficoPotencia();
102             }
103         });
104         menuPrincipal.getSalvar().addActionListener(new ActionListener() {
105             public void actionPerformed(ActionEvent ev) {
106                 try{
107                     redeEpistemicaView.pause();
108                     JFileChooser fileChooser = new JFileChooser(new File("experimentos"));
```

```
109                             int choose = fileChooser.showOpenDialog(MainFrame.this);
110                             if(choose==JFileChooser.APPROVE_OPTION){
111                                 File file = fileChooser.getSelectedFile();
112                                 String xml = RedeDao.getXml(redeEpistemica, redeEpistemicaView,
113 configuracoesPanel);
114                                 FileUtils.writeStringToFile(file, xml, "utf-8");
115                             }
116                         }catch(Exception e){
117                             logger.error("Erro ao salvar configurações: ",e);
118                             JOptionPane.showMessageDialog(MainFrame.this,"Erro ao salvar configurações: " +
119 e.getMessage());
120                         }
121                     }
122                 });
123 
124                 menuPrincipal.getAbrir().addActionListener(new ActionListener() {
125                     public void actionPerformed(ActionEvent ev) {
126                         clean();
127                         JFileChooser fileChooser = new JFileChooser(new File("experimentos"));
128                         int choose = fileChooser.showOpenDialog(MainFrame.this);
129                         if(choose==JFileChooser.APPROVE_OPTION){
130                             File file = fileChooser.getSelectedFile();
131                             RedeDao.loadFromXml(configuracoesPanel, redeEpistemica, redeEpistemicaView, file);
132                         }
133                         refreshConf();
134                     }
135                 });
136 
137                 menuPrincipal.getNovo().addActionListener(new ActionListener(){
138                     public void actionPerformed(ActionEvent e) {
139                         novo();
140                     }
141                 });
142 
143                 menuPrincipal.getVerLinhasVermelhas().setSelected(redeEpistemicaView.isVerLinhasVermelhas());
144                 menuPrincipal.getVerLinhasVermelhas().addChangeListener(new ChangeListener(){
145                     public void stateChanged(ChangeEvent e) {
146 
147 redeEpistemicaView.setVerLinhasVermelhas(menuPrincipal.getVerLinhasVermelhas().isSelected());
148                     }
149                 });
150                 menuPrincipal.getVerLinhasAzuis().setSelected(redeEpistemicaView.isVerLinhasAzuis());
151                 menuPrincipal.getVerLinhasAzuis().addChangeListener(new ChangeListener(){
152                     public void stateChanged(ChangeEvent e) {
153                         redeEpistemicaView.setVerLinhasAzuis(menuPrincipal.getVerLinhasAzuis().isSelected());
154                     }
155                 });
156                 menuPrincipal.getZoomMais().addActionListener(new ActionListener(){
157                     public void actionPerformed(ActionEvent e) {
158                         redeEpistemicaView.zoomMais();
159                     }
160                 });
161                 menuPrincipal.getZoomMenos().addActionListener(new ActionListener(){
162                     public void actionPerformed(ActionEvent e) {
163                         redeEpistemicaView.zoomMenos();
```

```
164                 }
165             });
166             menuPrincipal.getTreinamentoPrevio().addActionListener(new ActionListener(){
167                 public void actionPerformed(ActionEvent e) {
168                     treinar();
169                 }
170             });
171             menuPrincipal.getVerPesosDoSelecionado().addActionListener(new ActionListener(){
172                 public void actionPerformed(ActionEvent e) {
173 
174 redeEpistemicaView.setVerPesosDoSelecionado(menuPrincipal.getVerPesosDoSelecionado().isSelected());
175                 }
176             });
177 
178             menuPrincipal.getColorir().addActionListener(new ActionListener(){
179                 public void actionPerformed(ActionEvent e) {
180                     List<AgenteEpistemico> agentes = redeEpistemica.getListAgenteEpistemico();
181                     Set<ParEpistemico> crencas = new HashSet<ParEpistemico>();
182                     for(AgenteEpistemico agente : agentes){
183                         List<ParEpistemico> crencaS = agente.getCrencas();
184                         for(ParEpistemico crenca : crencaS){
185                             crencas.add(crenca);
186                         }
187                     }
188                     if(crencas.size()==0){
189                         JOptionPane.showMessageDialog(MainFrame.this, "Nenhuma crença");
190                     }else{
191                         JFrame jFrame = new JFrame("Crenças ");
192                         jFrame.setLayout(new BorderLayout());
193                         crencaView = new CrencaView(crencas, redeEpistemica, redeEpistemicaView);
194                         jFrame.add(crencaView);
195                         jFrame.pack();
196                         jFrame.setVisible(true);
197                     }
198                 }
199             });
200             configuracoesPanel.getTxtSnapShot().addFocusListener(new FocusListener() {
201                 public void focusLost(FocusEvent focusEvent) {
202                     try{
203 
204 redeEpistemicaView.setSnapshot(Integer.valueOf(configuracoesPanel.getTxtSnapShot().getText()));
205                     }catch(Exception e){}
206                 }
207                 public void focusGained(FocusEvent e) {}
208             });
209             configuracoesPanel.getSpnTheta().addChangeListener(new ChangeListener(){
210                 public void stateChanged(ChangeEvent e) {
211                     try{
212 
213 redeEpistemica.setTheta(Double.valueOf(configuracoesPanel.getSpnTheta().getValue().toString()));
214                     }catch(Exception ex){}
215                 }
216             });
217             configuracoesPanel.getBtnOk().addActionListener(new ActionListener(){
218                 public void actionPerformed(ActionEvent e) {
```

```
219                     novo();
220                 }
221             });
222             configuracoesPanel.getSpnPassoMax().addChangeListener(new ChangeListener(){
223                 public void stateChanged(ChangeEvent e) {
224                     int passoMax = Integer.valueOf(configuracoesPanel.getSpnPassoMax().getValue().toString());
225                     logger.info("Passo Máximo = " + passoMax);
226                     redeEpistemicaView.setPassoMax(passoMax);
227                 }
228             });
229             agenteListPanel.addKeyListener(new KeyListener(){
230                 public void keyPressed(KeyEvent e) {}
231                 public void keyReleased(KeyEvent e) {
232                     logger.debug("keyPressed");
233                     if(e.getKeyCode()==KeyEvent.VK_DELETE){
234                         boolean paused = redeEpistemicaView.isPaused();
235                         if(!paused){
236                             redeEpistemicaView.pause();
237                         }
238 
239                         List<AgenteEpistemico> agentes = agenteListPanel.getSelecionados();
240                         logger.debug("Matando "+agentes.size());
241                         for(AgenteEpistemico agente:agentes){
242                             redeEpistemica.matarAgente(agente);
243                         }
244                         if(!paused) redeEpistemicaView.continuar();
245                     }
246                 }
247                 public void keyTyped(KeyEvent e) {}
248             });
249             agenteListPanel.addListSelectionListener(new ListSelectionListener(){
250                 @Override
251                 public void valueChanged(ListSelectionEvent e) {
252                     redeEpistemicaView.selecionarAgentes(agenteListPanel.getSelecionados());
253                 }
254             });
255             //layout
256             this.setJMenuBar(menuPrincipal);
257             JPanel rightPanel = new JPanel(new BorderLayout());
258             JPanel leftPanel = new JPanel(new BorderLayout());
259             leftPanel.add(configuracoesPanel,BorderLayout.NORTH);
260 
261             this.setLayout(new BorderLayout());
262             rightPanel.add(redeEpistemicaView,BorderLayout.CENTER);
263             rightPanel.add(controlePanel,BorderLayout.NORTH);
264 
265             final JScrollPane scrollPane = new
266 JScrollPane(agenteListPanel,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,JScrollPane.HORIZONTAL_SCRO
267 LLBAR_NEVER);
268 
269             leftPanel.add(scrollPane, BorderLayout.CENTER);
270             this.addWindowListener(this);
271             this.add(leftPanel,BorderLayout.WEST);
272             this.add(rightPanel, BorderLayout.CENTER);
273             sysInfo.setBorder(BorderFactory.createEmptyBorder(3, 3,3, 3));
```

```
274             sysInfo.setHorizontalAlignment(SwingConstants.RIGHT);
275             this.add(sysInfo,BorderLayout.SOUTH);
276             this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
277             this.setSize(800,600);
278             this.setVisible(true);
279             instance = this;
280         }
281
282         public void treinar(){
283             JFrame jFrame = new JFrame("Treinamento Prévio");
284             jFrame.setLayout(new BorderLayout());
285             jFrame.add(CrencaTreinarView.getInstance());
286             jFrame.pack();
287             jFrame.setVisible(true);
288         }
289
290         public void clean(){
291             File dir = new File("screenshots/automatica");
292             File files[] = dir.listFiles();
293             for(File f: files){
294                 if(f.getName().endsWith(".html")){
295                     f.delete();
296                 }
297             }
298             dir = new File("screenshots/automatica/imagens");
299             files = dir.listFiles();
300             for(File f: files){
301                 if(f.getName().endsWith(".png")){
302                     f.delete();
303                 }
304             }
305             NumeroAleatorio.restart();
306             RedeEpistemica.setEtapa(0);
307             agenteListPanel.reiniciar();
308             redeEpistemicaView.pause();
309         }
310
311         public void refreshConf(){
312             redeEpistemicaView.setSnapshot(Integer.valueOf(configuracoesPanel.getTxtSnapShot().getText()));
313
314 redeEpistemicaView.setPassoMax(Integer.valueOf(configuracoesPanel.getSpnPassoMax().getValue().toString()));
315             try{
316
317 redeEpistemica.setTheta(Double.valueOf(configuracoesPanel.getSpnTheta().getValue().toString()));
318             }catch(Throwable e){}
319             int estr[] = configuracoesPanel.getEstruturaRede();
320             parModelo = ParEpistemicoFactory.criar(estr[0], estr[2]);
321         }
322
323         /**
324          * Nova simulacao
325          */
326         public void novo(){
327             clean();
328             refreshConf();
```

```
329             int qtd = Integer.valueOf(configuracoesPanel.getTxtQtdAgentes().getText());
330             for(int i=0;i<qtd;i++){
331                     criarAgente();
332             }
333     }
334
335     @Override
336     public void windowActivated(WindowEvent e) {
337     }
338
339     @Override
340     public void windowClosed(WindowEvent e) {
341     }
342
343     @Override
344     public void windowClosing(WindowEvent e) {
345             redeEpistemicaView.finalizar();
346             System.exit(0);
347     }
348
349     @Override
350     public void windowDeactivated(WindowEvent e) {
351
352     }
353
354     @Override
355     public void windowDeiconified(WindowEvent e) {
356
357     }
358
359     @Override
360     public void windowIconified(WindowEvent e) {
361
362     }
363
364     @Override
365     public void windowOpened(WindowEvent e) {
366
367     }
368
369     @Override
370     public void criado(AgenteEpistemico agente) {
371             agenteListPanel.criado(agente);
372     }
373
374     @Override
375     public void morto(AgenteEpistemico agente) {
376             agenteListPanel.morto(agente);
377     }
378
379     /**
380      * Cria um agente de acordo com as configuracoes
381      */
382     @Override
383     public AgenteEpistemico criarAgente() {
```

```
384             //TODO talvez devesse existir um factory implementado em outro local
385             int qtd = Integer.valueOf(configuracoesPanel.getTxtQtdAgentes().getText());
386             int morrerEm = Integer.valueOf(configuracoesPanel.getTxtMorrerEmXpublicacoes().getText());
387             double maxDiff = Double.valueOf(configuracoesPanel.getTxtMaxDiff().getText());
388             double errorTolerance = Double.valueOf(configuracoesPanel.getTxtErrorTolerance().getText());
389             boolean chkSomenteUltimaTeoria = configuracoesPanel.getChkSomenteUltimaTeoria().isSelected();
390             boolean pesoAleatorio = configuracoesPanel.getChkPesoAleatorio().isSelected();
391             boolean distribuicaoAleatoria = configuracoesPanel.getChkDistribuicaoAleatoria().isSelected();
392             int distanciaMaxRepulsao =
393 Integer.valueOf(configuracoesPanel.getTxtDistanciaMaxRepulsao().getText());
394             int criarNovoEm = Integer.valueOf(configuracoesPanel.getTxtCriarNovoEm().getText());
395             int w = redeEpistemicaView.getWidth();
396             int h = redeEpistemicaView.getHeight();
397             int i = redeEpistemica.getListAgenteEpistemico().size();
398             logger.debug("chkSomenteUltimaTeoria = " + chkSomenteUltimaTeoria);
399             int x = 0,y = 0;
400             if(distribuicaoAleatoria){//distr aleatoria
401                 x = (int)(w*NumeroAleatorio.gerarNumero());
402                 y = (int)(h*NumeroAleatorio.gerarNumero());
403             }else{//distribuicao organizada
404                 double d = Math.sqrt(qtd);
405                 int linhas = (int)Math.floor(d);
406                 int colunas = (int)Math.ceil(d);
407                 int col = i%linhas;
408                 int lin = i/colunas;
409                 x = col * (w/colunas) + (w/colunas)/2;
410                 y = lin * (w/linhas) + (w/linhas)/2;
411             }
412             double freq = Double.valueOf(configuracoesPanel.getTxtFrequencia().getText());
413             int estr[] = configuracoesPanel.getEstruturaRede();
414             AgenteEpistemico agente = new AgenteEpistemico(estr);
415             agente.setVontadeDePublicar(freq);
416             redeEpistemica.inserirAgente(agente,pesoAleatorio);
417             redeEpistemicaView.addAgente(agente, x, y);
418             agente.setMaxDiff(maxDiff);
419             agente.setMorrerEmXpublicacoes(morrerEm);
420             agente.setSomenteUltimaTeoria(chkSomenteUltimaTeoria);
421             agente.setCriarNovoEm(criarNovoEm);
                agente.setErrorTolerance(errorTolerance);
                redeEpistemicaView.setDistanciaMaximaRepulsao(distanciaMaxRepulsao);
                return agente;
        }

        @Override
        public void info(String info) {
            sysInfo.setText(info);
        }

        public static MainFrame getLastInstance() {
            return instance;
        }
    }
```

view/ MenuPrincipal.java

```
1   package br.unicarioca.redesepistemicas.view;
2
3   import java.awt.event.InputEvent;
4   import java.awt.event.KeyEvent;
5
6   import javax.swing.JCheckBoxMenuItem;
7   import javax.swing.JMenu;
8   import javax.swing.JMenuBar;
9   import javax.swing.JMenuItem;
10  import javax.swing.KeyStroke;
11
12  public class MenuPrincipal extends JMenuBar{
13      private static final long serialVersionUID = 1L;
14      private JMenu arquivo;
15      private JMenu visualizar;
16      private JMenu treinar;
17      private JMenu crencas;
18      private JMenuItem graficoPotencia;
19      private JCheckBoxMenuItem verLinhasVermelhas;
20      private JCheckBoxMenuItem verLinhasAzuis;
21      private JCheckBoxMenuItem verPesosDoSelecionado;
22      private JMenuItem salvar;
23      private JMenuItem abrir;
24      private JMenuItem novo;
25      private JMenuItem colorir;
26      private JMenuItem zoomMais;
27      private JMenuItem zoomMenos;
28      private JMenuItem treinametoPrevio;
29      public MenuPrincipal() {
30          arquivo = new JMenu("Arquivo");
31          salvar = new JMenuItem("Salvar");
32          abrir = new JMenuItem("Abrir");
33          novo = new JMenuItem("Novo");
34
35          visualizar = new JMenu("Visualizar");
36          verLinhasVermelhas = new JCheckBoxMenuItem("Ver Linhas Vermelhas");
37          verLinhasAzuis = new JCheckBoxMenuItem("Ver Linhas Azuis");
38          verPesosDoSelecionado = new JCheckBoxMenuItem("Ver Pesos do(s) Selecionado(s)");
39          zoomMais = new JMenuItem("Zoom +");
40          zoomMenos = new JMenuItem("Zoom -");
41          graficoPotencia = new JMenuItem("Grafico Potencia");
42          treinar = new JMenu("Treinamento");
43          treinametoPrevio = new JMenuItem("Treinamento Prévio");
44
45          crencas = new JMenu("Crenças");
46          colorir = new JMenuItem("Colorir");
47
48
49          salvar.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,InputEvent.CTRL_DOWN_MASK
50  ));
51          abrir.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,InputEvent.CTRL_DOWN_MASK ));
52          novo.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,InputEvent.CTRL_DOWN_MASK));
53
```

```
54      zoomMais.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_PLUS,InputEvent.CTRL_DOWN_MASK ));
55
56      zoomMenos.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_MINUS,InputEvent.CTRL_DOWN_MASK
57      ));
58
59              this.add(arquivo);
60              arquivo.add(salvar);
61              arquivo.add(abrir);
62              arquivo.add(novo);
63
64              this.add(visualizar);
65              visualizar.add(graficoPotencia);
66              visualizar.add(verLinhasVermelhas);
67              visualizar.add(verLinhasAzuis);
68              visualizar.add(verPesosDoSelecionado);
69              visualizar.add(zoomMais);
70              visualizar.add(zoomMenos);
71
72              this.add(treinar);
73              treinar.add(treinametoPrevio);
74
75              this.add(crencas);
76              crencas.add(colorir);
77      }
78
79      public JMenuItem getGraficoPotencia() {
80              return graficoPotencia;
81      }
82
83      public JMenuItem getAbrir() {
84              return abrir;
85      }
86      public JMenuItem getSalvar() {
87              return salvar;
88      }
89      public JMenuItem getNovo() {
90              return novo;
91      }
92      public JCheckBoxMenuItem getVerLinhasVermelhas() {
93              return verLinhasVermelhas;
94      }
95      public JCheckBoxMenuItem getVerLinhasAzuis() {
96              return verLinhasAzuis;
97      }
98      public JMenuItem getZoomMais() {
99              return zoomMais;
100     }
101     public JMenuItem getZoomMenos() {
102             return zoomMenos;
103     }
104     public JMenuItem getTreinamentoPrevio() {
105             return treinametoPrevio;
106     }
107     public JCheckBoxMenuItem getVerPesosDoSelecionado() {
108             return verPesosDoSelecionado;
```

```
109             }
                public JMenuItem getColorir(){
                    return colorir;
                }
            }
```

view/ RedeEpistemicaView.java

```
1  package br.unicarioca.redesepistemicas.view;
2
3  import java.awt.BasicStroke;
4  import java.awt.Color;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7  import java.awt.Point;
8  import java.awt.Stroke;
9  import java.awt.event.ComponentEvent;
10 import java.awt.event.ComponentListener;
11 import java.awt.event.MouseEvent;
12 import java.awt.event.MouseListener;
13 import java.awt.event.MouseWheelEvent;
14 import java.awt.event.MouseWheelListener;
15 import java.awt.image.BufferedImage;
16 import java.util.ArrayList;
17 import java.util.List;
18
19 import javax.swing.ImageIcon;
20 import javax.swing.JButton;
21
22 import org.apache.log4j.Logger;
23
24 import br.unicarioca.redesepistemicas.bo.SalvarSnapShoot;
25 import br.unicarioca.redesepistemicas.modelo.AgenteEpistemico;
26 import br.unicarioca.redesepistemicas.modelo.AgenteEpistemicoFactory;
27 import br.unicarioca.redesepistemicas.modelo.Aresta;
28 import br.unicarioca.redesepistemicas.modelo.ComunicacaoListener;
29 import br.unicarioca.redesepistemicas.modelo.NumeroAleatorio;
30 import br.unicarioca.redesepistemicas.modelo.ParEpistemico;
31 import br.unicarioca.redesepistemicas.modelo.RedeEpistemica;
32
33 /**
34  * Extends JButton
35  */
36 public class RedeEpistemicaView extends JButton implements ComunicacaoListener, ControladoListener,
37 MouseWheelListener, MouseListener {
38       private static final long serialVersionUID = 1L;
39       private static Logger logger = Logger.getLogger(RedeEpistemicaView.class);
40       private RedeEpistemica redeEpistemica;
41       private Thread t;
42       private boolean run = true;
43       private BufferedImage bi = new BufferedImage(800, 600, BufferedImage.TYPE_INT_ARGB);
44       private Graphics graphics = bi.getGraphics();
45       private int delay = 0;
46       private boolean pause = false;
47       private boolean paused = false;
48       private float escala = 0.25f;
49       private int arrastarX = 0;
50       private int arrastarY = 0;
51       private boolean reiniciarBi = false;
52       private boolean showGraficoPotencia = false;
53       private AgenteEpistemicoFactory agenteEpistemicoFactory;
```

```
54      private int startDragX = 0;
55      private int startDragY = 0;
56      private int distanciaMaxRepulsao;
57      private int passoMax = 15;
58      private boolean verLinhasVermelhas = false;
59      private boolean verLinhasAzuis = false;
60      private boolean verPesosDoSelecionado = false;
61      private int algoritimoRepulsao = 2;
62      private AgenteListPanel agenteListPanel;
63      private int ordenarAgentesEmNEtapas = 10;
64      private int snapshot = 1000;
65      private BasicStroke comunicadorStroke = new BasicStroke(3.0f);
66      public RedeEpistemicaView(RedeEpistemica redeEpistemica) {
67          this.redeEpistemica = redeEpistemica;
68          this.redeEpistemica.addComunicacaoListener(this);
69          this.redeEpistemica.ligarLoop(false);
70          t = new Thread() {
71              @Override
72              public void run() {
73                  while (run) {
74                      try {
75                          if (!pause) {
76                              paused = false;
77                              initDesenho();
78                              desenharAgentes();
79                              atualizar();
80                              logger.debug("Atualizando imagem delay " + delay*10);
81                              setIcon(new ImageIcon(bi));
82
83                              //30 is a magic number, just to not flick the image
84                              Thread.sleep((delay*10)+30);
85                          } else {
86                              paused = true;
87                              Thread.sleep(84);
88                          }
89                      } catch (Exception e) {
90                          if (run)
91                              e.printStackTrace();
92                      }
93                  }
94              }
95          };
96          t.start();
97          this.addMouseListener(this);
98          this.addMouseWheelListener(this);
99          this.addComponentListener(new ComponentListener() {
100             public void componentHidden(ComponentEvent e) {}
101             public void componentMoved(ComponentEvent e) {}
102             public void componentResized(ComponentEvent e) {
103                 reiniciarBi = true;
104                 refresh();
105             }
106             public void componentShown(ComponentEvent e) {}
107         });
108     }
```

```
109
110         /**
111          * Redesenha os agentes
112          */
113         public void refresh(){
114               initDesenho();
115               desenharAgentes();
116               drawSelection();
117               setIcon(new ImageIcon(bi));
118         }
119
120         /**
121          * redeEpistemica.fazUmaEtapa();
122          */
123         private void atualizar() {
124               if(redeEpistemica.getListAgenteEpistemico().size()!=0){
125                     int etapa = redeEpistemica.getEtapa();
126                     if(etapa%ordenarAgentesEmNEtapas==0 && agenteListPanel!=null){
127                           agenteListPanel.refresh();
128                     }
129                     // pede para fazer uma etapa
130                     redeEpistemica.fazUmaEtapa();
131
132                     if(snapshot!=0 && etapa%snapshot==0){
133                           //salvar o estado do programa
134                           SalvarSnapShoot.getInstance().salvar(bi, etapa);
135                     }
136               }
137         }
138
139         private void initDesenho() {
140               if (reiniciarBi) {
141                     try{
142                           bi = new BufferedImage(this.getWidth(), this.getHeight(),
143   BufferedImage.TYPE_INT_ARGB);
144                           graphics = bi.getGraphics();
145                     }catch(Exception e){}
146                           reiniciarBi = false;
147               }
148               graphics.setColor(Color.WHITE);
149               graphics.fillRect(0, 0, this.getWidth(), this.getHeight());
150         }
151
152         /**
153          * Desenha todos os agentes em cor BLACK
154          */
155         private void desenharAgentes() {
156               graphics.setColor(Color.BLACK);
157               synchronized (redeEpistemica.getListAgenteEpistemico()) {
158                     if(showGraficoPotencia){
159                           BufferedImage bi = GraficoPotenciaFactory.create(redeEpistemica, 7, 300, 100);
160                           graphics.drawImage(bi, 5, 5,null);
161                     }
162                     logger.debug("desenharAgentes()");
163                     for (AgenteEpistemico agente : redeEpistemica.getListAgenteEpistemico()) {
```

```
164                         desenharAgente(agente);
165                     }
166                 }
167                 drawSelection();
168         }
169 
170         /**
171          * Desenha um agente com a cor corrente
172          *
173          * @param agente
174          * @param color
175          */
176         private AgenteEpistemico desenharAgente(AgenteEpistemico agente, boolean fill) {
177                 int raioCalc;// = (int) agente.getPesoReputacao();//
178                 raioCalc = agente.getRaio();
179                 float x = (agente.getX() - raioCalc) * escala;
180                 float y = (agente.getY() - raioCalc) * escala;
181                 int dim = Math.round(raioCalc * 2 * escala);
182                 int raio = Math.round(raioCalc * escala);
183                 //int raio = Math.round((int)agente.getPesoReputacao() * escala);
184                 //desenhar o que ele acredita
185                 if(agente.crencaMonitorada.size()>0){
186                         int parte = 360/(agente.crencaMonitorada.size()/2);
187                         int startAngle=0;
188                         Color old = graphics.getColor();
189                         int i=0;
190                         for(ParEpistemico par: agente.crencaMonitorada){
191                                 graphics.setColor(par.getCor());
192                                 if(i%2==0){
193                                         graphics.fillArc(arrastarX + Math.round(x), arrastarY + Math.round(y), dim, dim,
194 startAngle, parte);
195                                 }else{
196                                         graphics.fillArc(arrastarX + Math.round(x+(raio/2)), arrastarY +
197 Math.round(y+(raio/2)), raio, raio, startAngle, parte);
198                                         startAngle+=parte;
199                                 }
200                                 i++;
201                         }
202                         graphics.setColor(old);
203                 }
204                 if(fill){
205                         Graphics2D g2d = (Graphics2D) graphics;
206                         Stroke oldS = g2d.getStroke();
207                         g2d.setStroke(comunicadorStroke);
208                         g2d.drawOval(arrastarX + Math.round(x), arrastarY + Math.round(y), dim, dim);
209                         g2d.setStroke(oldS);
210                 }else if(agente.getColor()!=null){
211                         logger.info("Color = " + agente.getColor());
212                         Color old = graphics.getColor();
213                         graphics.setColor(agente.getColor());
214                         graphics.fillOval(arrastarX + Math.round(x), arrastarY + Math.round(y), dim, dim);
215                         graphics.setColor(old);
216                         desenharArestasComPesos(agente);
217                 }else{
218                         graphics.drawOval(arrastarX + Math.round(x), arrastarY + Math.round(y), dim, dim);
```

```
219             }
220
221             return agente;
222         }
223
224         /**
225          * Desenha um agente com a cor corrente
226          *
227          * @param agente
228          */
229         private AgenteEpistemico desenharAgente(AgenteEpistemico agente) {
230             return desenharAgente(agente,false);
231         }
232
233         /**
234          * Coloca o agente na posicao X e Y
235          *
236          * @param agente
237          * @param x
238          *          coordenada x
239          * @param y
240          *          coordenada y
241          * @return agenteNovo
242          */
243         public AgenteEpistemico addAgente(AgenteEpistemico agente, int x, int y) {
244             agente.setX(Math.round((x - arrastarX) / escala));
245             agente.setY(Math.round((y - arrastarY) / escala));
246             return desenharAgente(agente);
247         }
248
249         public void setRedeEpistemica(RedeEpistemica redeEpistemica) {
250             this.redeEpistemica = redeEpistemica;
251         }
252
253         public RedeEpistemica getRedeEpistemica() {
254             return redeEpistemica;
255         }
256
257         public void finalizar() {
258             // flag do loop infinito
259             run = false;
260             // thread do loop infinito
261             t.interrupt();
262         }
263
264         private void drawString(String string, int x, int y) {
265             graphics.drawString(string,arrastarX + Math.round(x * escala), arrastarY + Math.round(y * escala));
266         }
267
268         private void drawLine(int x,int y, int x2, int y2){
269             graphics.drawLine(arrastarX + Math.round(x * escala), arrastarY + Math.round(y * escala),
270     arrastarX + Math.round(x2 * escala), arrastarY + Math.round(y2 * escala));
271         }
272
273
```

```
274         boolean checkColisao(AgenteEpistemico agente){
275             int ex = agente.getX();
276             int ey = agente.getY();
277             for(AgenteEpistemico receptor:redeEpistemica.getListAgenteEpistemico()){
278                 if(agente==receptor) continue;
279                 int rx = receptor.getX();
280                 int ry = receptor.getY();
281                 double dx = ex - rx;
282                 double dy = ey - ry;
283                 double hip = Math.sqrt(dx * dx + dy * dy);
284                 double diam = agente.getRaio() + receptor.getRaio();
285                 if(hip < diam){//colidiu
286                     double relacao = diam/hip + 0.05;
287                     double mx = dx * relacao;
288                     double my = dy * relacao;
289                     agente.setX(rx + (int) mx);
290                     agente.setY(ry + (int) my);
291                     return true;
292                 }
293             }
294             return false;
295         }
296 
297         /**
298          * Faz o agente dar um passo
299          * TODO calcular colisao para nao deixar sobrepor
300          */
301         public void depoisDeComunicar(AgenteEpistemico emissor, AgenteEpistemico receptor, Double peso,
302     Double diff) {
303 
304             int ex = emissor.getX();
305             int ey = emissor.getY();
306             int rx = receptor.getX();
307             int ry = receptor.getY();
308             // graphics.setColor(Color.BLUE);
309 
310             boolean colidiu = checkColisao(receptor);
311             // andar
312             double ref = 1.0;
313 
314             double passo = (ref - diff) * passoMax;
315             double dx = ex - rx;
316             double dy = ey - ry;
317             double hip = Math.sqrt(dx * dx + dy * dy);
318             double relacao = Math.min(passo / hip, 1.0);
319             double mx = dx * relacao;
320             double my = dy * relacao;
321             if (receptor.getMaxDiff() - diff > 0) {
322                 if(!colidiu){
323                     receptor.setX(rx + (int) mx);
324                     receptor.setY(ry + (int) my);
325                 }
326                 graphics.setColor(Color.BLUE);
327                 if(verLinhasAzuis){
328                     drawLine(ex,ey,rx,ry);
```

```
329                     }
330                 } else {
331                     // repulsao
332                     if(algoritimoRepulsao==1){
333                         if (hip < distanciaMaxRepulsao) {// max
334                             if (mx + my < 1.0) {
335                                 if (NumeroAleatorio.gerarNumero() < .5) {
336                                     if (NumeroAleatorio.gerarNumero() < .5)
337                                         mx = 5.0;
338                                     else
339                                         mx = -5.0;
340                                 } else {
341                                     if (NumeroAleatorio.gerarNumero() < .5)
342                                         my = 5.0;
343                                     else
344                                         my = -5.0;
345                                 }
346                             }
347                             mx *= .5f;
348                             my *= .5f;
349                             if(!colidiu){
350                                 receptor.setX(rx - (int) mx);
351                                 receptor.setY(ry - (int) my);
352                             }
353                         }
354                     }else{
355                         if (hip < distanciaMaxRepulsao) {
356                             mx *= hip/distanciaMaxRepulsao;
357                             my *= hip/distanciaMaxRepulsao;
358                             if(!colidiu){
359                                 receptor.setX(rx - (int) mx);
360                                 receptor.setY(ry - (int) my);
361                             }
362                         }
363                     }
364                     graphics.setColor(Color.RED);
365                     if (verLinhasVermelhas) {
366                         drawLine(ex,ey,rx,ry);
367                     }
368                 }
369                 desenharAgente(receptor);
370                 //this.setIcon(new ImageIcon(bi));
371             }
372 
373             @Override
374             public void comunicadorEscolhido(AgenteEpistemico emissor) {
375                 graphics.setColor(Color.GREEN);
376                 desenharAgente(emissor,true);
377                 //this.setIcon(new ImageIcon(bi));
378             }
379 
380             /**
381              * Metodo agora sincrono,
382              * ao sair dele garante que pausou.
383              */
```

```
384         public void pause() {
385             pause = true;
386             redeEpistemica.ligarLoop(false);
387             //espera pausar
388             while (!paused)
389                 try {
390                     Thread.sleep(10);
391                 } catch (Exception e) {}
392             ;
393         }
394 
395         public void continuar() {
396             pause = false;
397             redeEpistemica.ligarLoop(true);
398         }
399 
400         public void setVelocidade(int value) {
401             delay = value;
402         }
403 
404         public void reiniciar() {
405             pause = true;
406 
407             this.redeEpistemica.getListAgenteEpistemico().clear();
408             pause = false;
409         }
410 
411         @Override
412         public void mouseWheelMoved(MouseWheelEvent e) {
413             String message;
414             int notches = e.getWheelRotation();
415             if (notches < 0) {
416                 message = "Mouse wheel moved UP " + -notches + " notch(es)";
417                 float escalaAnt = escala;
418                 zoomMais();
419                 int x = e.getX();
420                 x = Math.round((x - arrastarX) / escalaAnt);
421                 x = arrastarX + Math.round(x * escala);
422                 arrastarX += e.getX() - x;
423                 int y = e.getY();
424                 y = Math.round((y - arrastarY) / escalaAnt);
425                 y = arrastarY + Math.round(y * escala);
426                 arrastarY += e.getY() - y;
427             } else {
428                 message = "Mouse wheel moved DOWN " + notches + " notch(es)";
429                 float escalaAnt = escala;
430                 zoomMenos();
431                 int x = e.getX();
432                 x = Math.round((x - arrastarX) / escalaAnt);
433                 x = arrastarX + Math.round(x * escala);
434                 arrastarX += e.getX() - x;
435                 int y = e.getY();
436                 y = Math.round((y - arrastarY) / escalaAnt);
437                 y = arrastarY + Math.round(y * escala);
438                 arrastarY += e.getY() - y;
```

```
439                 }
440                 logger.debug(message);
441                 if(pause){
442                       refresh();
443                 }
444           }
445           public void zoomMais(){
446                 escala *= 1.2f;
447           }
448           public void zoomMenos(){
449                 escala *= 0.8f;
450           }
451           @Override
452           public void mouseClicked(MouseEvent e) {
453                 addAgente(agenteEpistemicoFactory.criarAgente(), e.getX(), e.getY());
454           }
455
456           @Override
457           public void mouseEntered(MouseEvent e) {
458           }
459
460           @Override
461           public void mouseExited(MouseEvent e) {
462           }
463
464           private Point startSelection = null;
465           private Point endSelection = null;
466           @Override
467           public void mousePressed(MouseEvent e) {
468                 if (e.getButton() == MouseEvent.BUTTON3) {
469                       startDragX = e.getX();
470                       startDragY = e.getY();
471                 }else if(e.getButton()== MouseEvent.BUTTON1){
472                       startSelection = e.getPoint();
473                 }
474           }
475
476           @Override
477           public void mouseReleased(MouseEvent e) {
478                 if (e.getButton() == MouseEvent.BUTTON3) {
479                       arrastarX += e.getX() - startDragX;
480                       arrastarY += e.getY() - startDragY;
481                       refresh();
482                 }else if(e.getButton()== MouseEvent.BUTTON1){
483                       endSelection = e.getPoint();
484                       initDesenho();
485                       desenharAgentes();
486                       int x=0,y=0,x2=0,y2=0;
487                       if(startSelection!=null && endSelection!=null){
488
489                             x2 = Math.max(startSelection.x, endSelection.x);
490                             y2 = Math.max(startSelection.y, endSelection.y);
491                             x = Math.min(startSelection.x, endSelection.x);
492                             y = Math.min(startSelection.y, endSelection.y);
493                             x = Math.round((x - arrastarX) / escala);
```

```
494                         y = Math.round((y - arrastarY) / escala);
495                         x2 = Math.round((x2 - arrastarX) / escala);
496                         y2 = Math.round((y2 - arrastarY) / escala);
497                     }
498                     synchronized (redeEpistemica.getListAgenteEpistemico()) {
499                         logger.debug("desenharAgentes()");
500                         List<Integer> listaSelecionados = new ArrayList<Integer>();
501                         for (AgenteEpistemico agente : redeEpistemica.getListAgenteEpistemico()) {
502                             desenharAgente(agente);
503                             boolean selecionado = false;
504                             logger.debug(x + "< (ax="+agente.getX() + ")<"+x2);
505                             if(x<agente.getX() && agente.getX()<x2){
506                                 if(y<agente.getY() && agente.getY()<y2){
507                                     agente.setColor(new Color(55,55,255));
508                                     selecionado = true;
509                                     logger.debug("Selecionando " + agente.getNome());
510                                     int ic=agenteListPanel.indexOf(agente);
511                                     if(ic!=-1){
512                                         listaSelecionados.add(ic);
513                                     }
514                                 }
515                             }
516                             if(!selecionado){
517                                 logger.debug("Nao selecionado " + agente.getNome());
518                             }
519                         }
520                         if(agenteListPanel!=null){
521                             int[] arr = new int[listaSelecionados.size()];
522                             for(int i=0;i<listaSelecionados.size();i++)arr[i]=listaSelecionados.get(i);
523                             agenteListPanel.getJList().setSelectedIndices(arr);
524                         }
525                     }
526                     drawSelection();
527                 }
528             }
529
530             private void drawSelection(){
531                 logger.debug("drawSelection()");
532                 if(startSelection == null || endSelection==null) return;
533                 int width = Math.abs(startSelection.x-endSelection.x);
534                 int height = Math.abs(startSelection.y-endSelection.y);
535                 if(width<1) return;
536                 int x = Math.min(startSelection.x, endSelection.x);
537                 int y = Math.min(startSelection.y, endSelection.y);
538                 graphics.setColor(Color.GRAY);
539                 graphics.drawRect(x, y, width, height);
540             }
541
542             public void setAgenteEpistemicoFactory(AgenteEpistemicoFactory agenteEpistemicoFactory) {
543                 this.agenteEpistemicoFactory = agenteEpistemicoFactory;
544             }
545
546             public int getDistanciaMaxRepulsao() {
547                 return distanciaMaxRepulsao;
548             }
```

```
549
550             public void setDistanciaMaximaRepulsao(int distanciaMaxRepulsao) {
551                 this.distanciaMaxRepulsao = distanciaMaxRepulsao;
552             }
553
554             public int getPassoMax() {
555                 return passoMax;
556             }
557
558             public void setPassoMax(int passoMax) {
559                 this.passoMax = passoMax;
560             }
561
562             public void setVerLinhasVermelhas(boolean mostrarLinhasVermelhas) {
563                 this.verLinhasVermelhas = mostrarLinhasVermelhas;
564             }
565
566             public boolean isVerLinhasVermelhas(){
567                 return verLinhasVermelhas;
568             }
569
570             public void setVerLinhasAzuis(boolean verLinhasAzuis) {
571                 this.verLinhasAzuis = verLinhasAzuis;
572             }
573
574             public boolean isVerLinhasAzuis() {
575                 return verLinhasAzuis;
576             }
577
578             /**
579              * Chamado quando existe um evento vindo do ListSelectionListener
580              * @param agentes
581              */
582             public void selecionarAgentes(List<AgenteEpistemico> agentes) {
583                 List<AgenteEpistemico> listAgentes=redeEpistemica.getListAgenteEpistemico();
584                 for(AgenteEpistemico agente:listAgentes)
585                     agente.setColor(null);
586                 if(pause){
587                     initDesenho();
588                     desenharAgentes();
589                     graphics.setColor(new Color(55,55,255));
590                     for(AgenteEpistemico agente:agentes){
591                         agente.setColor(new Color(55,55,255));
592                         desenharAgente(agente,true);
593                         desenharArestasComPesos(agente);
594                     }
595                     this.setIcon(new ImageIcon(bi));
596                 }else{
597                     for(AgenteEpistemico agente:agentes)
598                         agente.setColor(new Color(55,55,255));
599                 }
600             }
601
602             private void desenharArestasComPesos(AgenteEpistemico agente){
603                 if(verPesosDoSelecionado){
```

```
604                         synchronized (agente.getArestas()) {
605                             for(Aresta aresta:agente.getArestas()){
606                                 //achar o meio
607                                 int x = (agente.getX() - aresta.getReceptor().getX())/2;
608                                 int y = (agente.getY()-aresta.getReceptor().getY())/2;
609                                 drawString(aresta.getPeso().toString(), aresta.getReceptor().getX()+x,
610     aresta.getReceptor().getY()+y);
611
612     drawLine(agente.getX(),agente.getY(),aresta.getReceptor().getX(),aresta.getReceptor().getY());
613                             }
614                         }
615                     }
616             }
617
618             public boolean isPaused() {
619                 return paused;
620             }
621
622             public void setVerPesosDoSelecionado(boolean verPesosDoSelecionado) {
623                 this.verPesosDoSelecionado = verPesosDoSelecionado;
624             }
625
626             public void setAgenteListPanel(AgenteListPanel agenteListPanel) {
627                 this.agenteListPanel = agenteListPanel;
628             }
629             public int getSnapshot() {
630                 return snapshot;
631             }
632             public void setSnapshot(int snapshot) {
633                 this.snapshot = snapshot;
634             }
635
636             public BufferedImage getCurrentScreen(){
637                 return bi;
638             }
639
640             /**
641              * Gera uma foto do momento
642              */
643             public void criarFotografia() {
644                 if(bi!=null){
645                     boolean estavaPausado = pause;
646                     if(!estavaPausado) pause();
647                     SalvarSnapShoot.getInstance().salvar(bi);
                        if(!estavaPausado) continuar();
                    }
                }

                public void ligaDesligaGraficoPotencia() {
                    showGraficoPotencia=!showGraficoPotencia;
                }
            }
```