ROGÉRIO PERINO DE OLIVEIRA NEVES

A.L.I.V.E.
VIDA ARTIFICIAL EM AMBIENTES VIRTUAIS: UMA PLATAFORMA EXPERIMENTAL EM REALIDADE VIRTUAL PARA ESTUDOS DOS SERES VIVOS E DA DINÂMICA DA VIDA

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

São Paulo
2003

ROGÉRIO PERINO DE OLIVEIRA NEVES

A.L.I.V.E.
VIDA ARTIFICIAL EM AMBIENTES VIRTUAIS: UMA PLATAFORMA EXPERIMENTAL EM REALIDADE VIRTUAL PARA ESTUDOS DOS SERES VIVOS E DA DINÂMICA DA VIDA

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

Área de Concentração:
Engenharia de Sistemas Eletrônicos

Orientador:
Prof. Dr. Marcio Lobo Netto

São Paulo
2003

FICHA CATALOGRÁFICA

Neves, Rogério Perino de Oliveira

Vida Artificial em Ambientes Virtuais: Implementando uma plataforma computacional para estudos dos seres vivos e da dinâmica da vida, 2003.

166p.

Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Vida Artificial 2.Realidade Virtual 3.Ciência da Computação I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t

Aos cientistas e filósofos que trabalham incessantemente na busca pela resposta à inquietante pergunta “o que é vida”.

RESUMO

Este trabalho descreve os estudos realizados sobre o tema Vida Artificial, que levaram à criação do projeto A.L.I.V.E. (*Artificial Life in Virtual Environments*) desenvolvido no Laboratório de Sistemas Integráveis da Escola Politécnica da USP. O projeto tem como objetivo a aplicação de tecnologias de realidade virtual na visualização de experimentos em Vida Artificial. A plataforma implementada utiliza-se do paradigma de orientação a objetos, fazendo uso dos recursos disponíveis através da linguagem Java e do API Java3D, o que garante portabilidade e compatibilidade com diversos dispositivos, também possibilitando a execução baseada em *browser*, permitindo a visualização de experimentos em páginas interativas na Internet. O programa foi desenvolvido num contexto multi-agentes e prevê sua utilização em arquiteturas distribuídas e multi-processadas, fazendo uso da capacidade de multi-threading da linguagem Java.

ABSTRACT

This work describes the studies conducted under the theme Artificial Life, which leaded to the creation of the A.L.I.V.E. (Artificial Life in Virtual Environments) project, developed on the Laboratory of Integrated Systems of the Polytechnic School at USP. The project aims to apply virtual reality technologies to the visualization of Artificial Life experiments. The developed platform employs an object oriented paradigm, making use of resources available trough Java language and the Java3D API, what grants portability and device compatibility, also making possible browser-based execution of experiments in interactive web pages through Internet. The program was developed in a multi-agent context, allowing the execution in distributed and multi-processed architectures, making use of the multi-threading capability of the Java language.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ALIVE	–	*Artificial Life in Virtual Environments*
ATP	–	Adenosina tri-fosfato
DNA	–	*Dioxirribonucleic Acid* ou Ácido Dioxirribonucléico
EPUSP	–	Escola Politécnica da Universidade de São Paulo
ID	–	Identificação de agente
IP	–	*Internet Protocol*
LSI	–	Laboratório de Sistema Integráveis
POO	–	Programação Orientada a Objetos
RN	–	Redes Neurais Artificiais
RNA	–	*Ribonucleic Acid* ou Ácido ribonucléico
RV	–	Realidade Virtual
SMA	–	Sistemas Multi-Agentes
UML	–	*Unified Modeling Language*
USP	–	Universidade de São Paulo
VA	–	Vida Artificial

PREFÁCIO

Durante o programa de mestrado no Laboratório de Sistemas Integráveis da Escola Politécnica da Universidade de São Paulo (EPUSP), foi proposta a exploração do tema “Vida Artificial”, a princípio, visando considerar sua abrangência, seus envolvimentos e suas possibilidades.

“Vida Artificial” refere-se a um tema interdisciplinar, com muitas ramificações em áreas não só das ciências físicas ou biológicas, como, também, das humanidades, entendendo conhecimentos e aplicações que abrangem desde os estudos clássicos, entre os quais Física, Química e Biologia, até áreas como Sociologia, Antropologia, Genética, Inteligência Artificial e Sistemas de Informação, passando por uma larga gama de engenharias e áreas de tecnologia que podem se beneficiar com seu estudo.

Prontamente, comecei a estudar tudo que pudesse ter uma tênue ligação com o assunto, cursando matérias com temas como Inteligência Artificial, Teoria do Caos, Computação Gráfica, Visualização Científica, Sistemas Nebulosos, Robótica e Sistemas Multi-Agentes. A tarefa se complicava, gradualmente, com o aumento do conhecimento a respeito do tema, e, conforme o domínio se elevava, o foco se perdia, demonstrando ser impossível para um só indivíduo dominar completamente todos os conceitos envolvidos. Com uma visão mais realista, soube que deveria me concentrar em estudos voltados para aplicação em uma área específica e bem definida, desenvolvendo um projeto direcionado ao estudo da área escolhida.

Portanto, possuindo uma formação em Física, com um bom domínio de matemática e um histórico em programação, decidi empenhar meus esforços na implementação de simulações envolvendo organismos virtuais. No processo, foram criados modelos para estudo de sistemas nervosos, propagação de informação, aplicações em robótica e para busca evolutiva em espaços multi-dimensionais.

Dada a série de dificuldades técnicas encontradas na implementação dos experimentos, muitas que consegui superar apenas graças a uma maior intimidade

com linguagens de programação, imaginei quanto tais dificuldades poderiam frustrar pesquisadores com menos intimidade com linguagens de programação. Assim, resolvi tentar facilitar-lhes a tarefa, implementando uma plataforma de estudos em Vida Artificial com código aberto, visando, também, facilitar a implementação de meus próprios experimentos. Pouco a pouco o projeto tomava forma.

Durante os meses que se seguiram, modelos foram propostos e protótipos implementados, incorporando flexibilidade e facilidades de uso. Não se espera que o protótipo apresentado seja perfeito, nem que seja de fato “final”. Uma vez que segue a doutrina do “código aberto”, permite que usuários e pesquisadores que venham a utilizá-lo dêem sua contribuição para seu aprimoramento, adicionando recursos e ferramentas, dando continuidade ao desenvolvimento do projeto A.L.I.V.E.

Esta dissertação sintetiza o trabalho desenvolvido, os assuntos estudados e os experimentos implementados durante o período de duração do projeto de pesquisa.

Rogério Neves

1. INTRODUÇÃO

> *CIÊNCIA: "Conjunto de conhecimentos socialmente adquiridos ou produzidos, historicamente acumulados, dotados de universalidade e objetividade que permitem sua transmissão, e estruturados com métodos, teorias e linguagens próprias, que visam compreender e possibilitam orientar a natureza e as atividades humanas".*
> *ENGENHARIA: "Arte de aplicar conhecimentos científicos e empíricos e certas habilitações específicas à criação de estruturas, dispositivos e processos que se utilizam para converter recursos naturais em formas adequadas ao atendimento das necessidades humanas". [Dicionário Aurélio - Século XXI]*

Este trabalho apresenta os estudos realizados sobre o tema Vida Artificial, compreendendo teoria e aplicação, realizados durante a especificação e desenvolvimento do projeto A.L.I.V.E. (*Artificial Life in Virtual Environments*). O projeto contemplou uma fase de pesquisa teórica, cujo objetivo foi relacionar muito do que já foi feito e está em atual desenvolvimento em pesquisa relacionada ao tema, e uma fase prática que descreve uma implementação computacional dos conceitos estudados, na forma de uma plataforma experimental de desenvolvimento de simulações.

Vida Artificial, ou VA, é um campo de estudos relativamente recente e ainda pouco explorado. Historicamente o termo data de 1987, quando foi usado para descrever uma conferência realizada em Los Alamos, Novo México, sobre "*Síntese e simulação de sistemas vivos*" [LANGTON, 1989].

Como disciplina, Vida Artificial está vinculada ao uso de modelos computacionais, envolvendo diretamente estudos de Engenharia de Sistemas, Ciência da Computação

e Cibernética, sendo conceitualmente similar a um campo mais antigo de estudo chamado Inteligência Artificial que visa extrair a estrutura lógica elementar da inteligência, construindo programas de computador que pensam, mesmo que de maneira rudimentar. Vida Artificial por sua vez, espera criar programas de computador não mais espertos que uma abelha, porém tão vivos quanto em uma colméia.

Vida Artificial usa o computador como instrumento principal. Como foram os telescópios para a astronomia e os microscópios para a biologia, o computador visa tornar alguns aspectos espaço-temporais visíveis ao olho humano, possibilitando a observação de eventos que ocorrem em escalas arbitrárias de espaço e tempo, tornando também visíveis aspectos abstratos, ou que excedem a capacidade de visualização humana. O computador torna possível a observação de milhões de anos de evolução em apenas alguns minutos de simulação, ou a condução da interação entre microorganismos pelo usuário, interativamente, em tempo real.

Como parte do projeto, foi proposto o desenvolvimento de uma plataforma computacional para experimentação, cujo objetivo é tornar o processo de implementação de experimentos em Vida Artificial simples e rápido, fazendo uso do estado de arte dos sistemas de visualização disponíveis, como placas aceleradoras gráficas 3D e dispositivos de Realidade Virtual, deixando a interface aberta à possibilidade de aplicação de tecnologias vindouras, visando tornar a experiência realizada no Ambiente Virtual visualmente rica e interativa.

Técnicas de visualização científica são empregadas para ilustrar variáveis dinâmicas ou características genéticas através da coloração, transparência e textura, tornando a exibição da informação mais natural e reduzindo a quantidade de informação numérica, exibida regularmente através de tabelas ou gráficos.

O capítulo 2 tem um caráter introdutório, tendo como objetivo esclarecer as motivações que levam ao estudo de VA e a implementação da plataforma e apresenta uma definição geral do que é VA e o que se espera de experimentos desenvolvidos

utilizando-se conceitos fundamentais de VA. Apresenta, também, em um contexto histórico, os trabalhos que precederam as modernas pesquisas sobre as origens da vida, remetendo a antiga questão fundamental “O Que é Vida?”. O capítulo também traz alguns problemas fundamentais considerados atualmente na simulação de seres artificiais, e ressalta alguns trabalhos que obtiveram destaque utilizando-se computadores como ferramenta de estudo na área.

O capítulo 3 apresenta alguns dogmas da simulação de agentes em ambientes virtuais. Traz uma comparação entre o quadro proposto e enfoques tradicionais em VA, como máquinas de estado e autômatos. O capítulo apresenta um pouco da teoria de multi-agentes, onde programas independentes interagem entre si produzindo uma complexidade que excede, em muitas vezes, a complexidade individual do programa de cada agente.

O capítulo 4 apresenta a parte prática do projeto, a especificação e implementação da plataforma de simulação proposta. Traz, além da motivação que levou a especificação da plataforma com as características descritas, uma descrição funcional de sua utilização e alguns diagramas representando as classes do programa e sua hierarquia.

O capítulo 5 traz alguns estudos de casos sobre o assunto, descrevendo a utilização da plataforma experimentalmente. Alguns temas chave foram explorados com finalidade demonstrativa, e alguns experimentos mais complexos foram implementados visando observar algumas características emergentes em sistemas multi-agentes inteligentes, dos quais já é possível extrair regras fundamentais das formas de vida.

O capítulo 6 encerra o trabalho sintetizando algumas conclusões gerais a respeito da pesquisa e sobre os trabalhos realizados. São, também, exploradas possibilidades de aplicação dos conceitos estudados aqui e extrapoladas futuras linhas de pesquisa que podem ser desenvolvidas, em continuidade ao projeto atual.

2. O QUE É VIDA ARTIFICIAL?

O novo campo de pesquisa científica, chamado de “Vida Artificial”, pode ser definido como “uma tentativa de abstrair a forma lógica da vida de sua manifestação material”. A idéia básica parece bem simples, porém, o estudo de Vida Artificial estende uma discussão muito mais abrangente que tem ocupado as mentes de cientistas e filósofos desde o início dos tempos: “O que é vida?”.

Por milênios, “Vida” tem sido um dos maiores mistérios metafísicos e científicos da humanidade. Recentemente, o advento de sistemas eletrônicos de monitoração, microscopia, tecnologias em genética e microbiologia, trouxeram uma nova luz para o tema. Agora, um novo instrumento é apresentado para o estudo da complexidade dos sistemas vivos, o computador. Simulações envolvendo conceitos fundamentais sobre o tema “o que é vida” têm sido desenvolvidas em laboratórios ao redor do mundo, já apresentando resultados muito intrigantes.

Atualmente, os computadores lançam luz no estudo de sistemas complexos e em novos princípios físicos como “Comportamento Emergente”, “Caos” e “Auto-Organização”, sendo largamente empregados em simulações abrangendo praticamente todas as áreas do conhecimento humano. Fazendo uso de sua dinâmica discreta, a implementação de regras simples muitas vezes leva a resultados extremamente complexos e até imprevisíveis, como no caso de máquinas de estado conhecidas como Autômatos Celulares [WOLFRAM, 2002].

A tese base da “Vida Artificial” é que a “Vida” é mais bem compreendida como um processo sistemático complexo, constituído de relações, regras e interações. Em suma, “Vida” é uma propriedade, mais que um fenômeno, e que, potencialmente, pode ser decomposta e separada das criaturas vivas atualmente conhecidas. “Vida” como processo possui propriedades como reprodução, variação genética, hereditariedade, comportamento, aprendizado, parâmetros como em um código genético e expressões que descrevem o caminho do código até o corpo físico final. Neste contexto, Vida é então descrita pelo que ‘faz’, mais do que pelo que ‘é’. A

Vida extrai energia do ambiente, cresce, se regenera, se reproduz, e este conjunto de processos descrito como ‘Vida’ pode ser considerado separadamente, para que possa ser estudado, modelado matematicamente, simulado com o uso de computadores e experimentado digitalmente sem o envolvimento de criaturas vivas.

Pesquisadores da área de VA despendem um grande esforço no sentido de convencer as pessoas da importância do seu trabalho. Além disso, estudos em VA levantam questões filosóficas intrigantes. Tais questões têm sido discutidas por filósofos e cientistas da atualidade, como constatado na matéria publicada pelo jornal “A Folha de São Paulo”, caderno “Mais” em 27 de outubro de 2002, com o título “Vida Digital”, onde questões éticas são relevadas quanto ao experimento com organismos digitais. “Pode um robô ou um programa realmente estar vivo?”, “Pode um organismo que exista puramente em um ambiente virtual estar vivo?”. Mesmo que este se mova como se estivesse vivo, se reproduza, se comunique, mesmo que tenha a forma de um “*pixel*” na tela de um computador? O que exatamente algo deve fazer para que seja caracterizado como vivo?

2.1. O que é Vida?

A pergunta é ao mesmo tempo uma das mais novas e das mais antigas que toma a mente do *homo sapiens*. Desde os primórdios da história registrada, o homem tem se perguntado de onde veio e para onde vai. Mitos, superstições e intrincadas teorias surgiram e se foram, desmentidas, incansavelmente tentando responder a inquietante pergunta. Muitos segredos da natureza foram desvendados na busca pela resposta, porém, muitos mitos perduram. Mas a cada passo da descoberta, a humanidade caminha, em passos lentos, em direção à verdade da criação.

Discussões sobre a vida podem tomar rumos sofisticados e filosóficos, embora do ponto de vista científico, devamos nos concentrar na caracterização dos aspectos que definem um organismo como vivo. Inicialmente, parece se tratar de uma tarefa simples, alguém pode pragmaticamente redigir uma lista definindo o que é necessário para algo “ser vivo”: crescer, reproduzir-se, reagir ao ambiente, assimilar

energia, excretar dejetos, morrer, etc. Porém, muitos desses conceitos são subjetivos, paradoxais e muitas vezes contraditórios.

Consideremos, por exemplo, organismos simples como vírus, bactérias, algas e amebas. Vírus não atendem qualquer uma das exigências da lista, se reproduzem com auxílio do mecanismo de outras células, e, portanto, é considerado “não vivo” por muitos pesquisadores. Pode permanecer inanimado por uma eternidade, até ter outra chance de infectar, e assim voltar à atividade. Indagando filosoficamente, trata-se de uma forma de vida extremamente prática, ou apenas de uma seqüência de algum programa genético?

Bactérias são organismos simples, agem de forma semelhante e possuem complexidade equivalente a muitas entidades que não são consideradas vivas. Bactérias podem existir indefinidamente, se não submetidas a condições danosas, assim como embriões humanos congelados, que podem permanecer dormentes como os vírus, sem a necessidade de um hospedeiro, podendo, a qualquer momento, se tornar um ser humano vivo e ativo. Um embrião é uma vida em potencial, como um programa interrompido no meio da execução. Podemos considerar bactérias, ou embriões congelados, formas de vida?

Algas podem ser facilmente confundidas com partículas em suspensão nos líquidos ou com poeira sobre os sólidos. Algas aquáticas permanecem imóveis, flutuando ao sabor das marés, apenas processando material circulante, com ajuda da luz solar, excretando as substâncias resultantes. Em algum momento, como em um estalo, o que era uma torna-se duas, sem que olhos distraídos percebam o processo. Tipos de algas conhecidas como “Chlamydomonas” [MARGULIS, 1998] ou “Algas da Neve” passaram desapercebidas por séculos, acreditando tratar-se de micro-partículas de ferro e enxofre, dando um tom avermelhado à neve, conhecida como “neve de melancia”.

Amebas são móveis, se reproduzem por meiose, reagem ao ambiente, assimilam energia e liberam dejetos, e, embora sejam imortais ninguém duvida que elas estejam

"vivas". Talvez isto signifique que todas as amebas existentes sejam partes móveis de um único organismo com três bilhões de anos de idade, uma espécie de "Super-Ameba" distribuída por todo o globo.

Questionar sobre "o que é vida" de forma simplista como apresentada acima pode ser extremamente arriscado do ponto de vista científico, e, embora qualquer pessoa possa dizer facilmente a diferença entre uma coisa "viva" e uma "não viva" apenas olhando para ela, nenhuma afirmação deve ser feita sem se basear em resultados experimentais e observações sistemáticas.

Christoph Adami, pesquisador da Caltech e um dos pioneiros no estudo de VA, em seu livro "Introduction to Artificial Life" [ADAMI, 1998] define "vida" num contexto termodinâmico como sendo:

> *"Vida é uma propriedade de uma amostra de unidades que compartilham informação codificada em um substrato físico e que, na presença de ruído, esforça-se para manter sua entropia significativamente abaixo da entropia máxima da amostra, em escalas de tempo que excedem a escala 'normal' de decaimento do substrato (que contém a informação) por muitas ordens de magnitude".*

O substrato em questão refere-se ao ambiente em que a vida se desenvolve e o tipo 'físico' foi, até muito recentemente, o único disponível para observação. Os estudos da origem da vida, até o momento, têm se concentrado no estudo desta manifestação específica da vida, que se desenvolveu num ambiente físico rico em carbono, nitrogênio e água, utilizando demais substâncias químicas presentes para se adaptar e reproduzir. No entanto, com o advento de novas tecnologias, sobretudo a de sistemas computacionais, temos a oportunidade de estudar a vida por outro ângulo, extraindo "características gerais" da vida como propriedade, tanto na forma em que é atualmente conhecida como na forma que poderia tomar, caso se manifestasse em ambientes alternativos.

A pergunta “O que é vida” tem ocupado a mente de filósofos e estudiosos durante os últimos séculos, e como resultado, inúmeras contribuições discretas e freqüentes foram feitas, tanto no sentido de fornecer respostas quanto no de levantar importantes questões (fundamentais para orientar as pesquisas), aumentando, assim, nosso conhecimento sobre “o que somos” e “de onde viemos”. Sendo esta uma questão tão antiga e duradoura, não há aqui qualquer pretensão de desvendar os segredos da vida, apenas de explorar os mecanismos que a natureza apresenta para solucionar problemas, dados os artifícios disponíveis no ambiente. O próximo tópico trata de alguns dos trabalhos desenvolvidos até o momento que contribuíram com o esclarecimento da questão. Muitos deles precederam as pesquisas atuais em genética e VA, sedimentando um terreno sólido nos quais novos estudos puderam se basear.

2.2. Predecessores Ilustres

Embora as perguntas sobre as origens da vida sejam milenares, só nos recentes duzentos anos foram feitas as primeiras tentativas de respondê-las cientificamente.

> *“Organic life beneath the shoreless waves*
> *Was born and nurs'd in ocean's pearly caves;*
> *First forms minute, unseen by spheric glass,*
> *Move on the mud, or pierce the watery mass;*
> *These, as successive generations bloom,*
> *New powers acquire and larger limbs assume;*
> *Whence countless groups of vegetation spring,*
> *And breathing realms of fin and feet and wing.”*
>
> *Erasmus Darwin. The Temple of Nature. 1802.*

Correndo o risco de deixar de fora alguns contribuintes de peso na área, aqui se encontram relacionados alguns dos mais renomados pesquisadores que influenciaram diretamente o desenvolvimento deste projeto, juntamente com suas contribuições para a elucidação da questão “O que é vida”.

“Erasmus Darwin” (1731-1802)

Físico respeitado e bem conhecido em seu tempo, poeta, biólogo, botânico e naturalista, com várias outras linhas de interesse, Erasmus, avô de Charles, nasceu em Nottinghamshire, Inglaterra, e se educou nas universidades de Cambridge e Edimburgo. Sua prática em medicina exercida em Lichfield ficou tão conhecida na época que chegou a ser convidado pelo rei George III para ser seu médico particular, posto que recusou com distinção, por motivos políticos. Seus interesses se estenderam muito além da medicina. Sua insatisfação em relação à teoria da imutabilidade das espécies de Lineu o levou a publicar suas próprias teorias em “*Zoonomia, or, The Laws of Organic Life*” (As leis da vida orgânica), em 1794, onde propõe a evolução gradual dos animais e plantas. Chegou a falar da seleção sexual, onde na competição entre os machos, o animal mais forte e ativo deve propagar a espécie visando seu aprimoramento. Este trabalho foi chamado de ‘a primeira hipótese consistente, universalmente abrangente da evolução’. Erasmus também apresentou suas idéias evolucionárias em verso, postumamente publicados em “*The Temple of Nature*”. As publicações de seus contemporâneos franceses, Cuvier e Lamarck, reforçaram sua posição e proveram base para estudos posteriores no campo, mais notavelmente o de seu neto Charles Darwin. Deve-se ressaltar, porém, que Charles rejeitou o mecanismo estrito e semi-experimental de seu avô, chegando a clamar que “*Zoonomia*” não teve qualquer influência no seu famoso “A Origem das Espécies”.

“Charles Darwin” (1809-1882)

Biólogo evolucionista, Charles Darwin é provavelmente um dos homens mais famosos que já viveram. Nasceu em Shrewsbury, Inglaterra, em 12 de fevereiro de 1809. Seguindo os passos do pai, começou a estudar medicina em Edimburgo, porém detestou o curso, embora este lhe tenha servido para desenvolver um interesse especial por história natural. Abandonando a medicina, dois anos depois foi estudar teologia em Cambridge. Sua pesquisa iniciou-se ao embarcar no veleiro Beagle em

dezembro de 1831 quando teve a oportunidade de viajar pelo mundo (inclusive pelo Brasil) registrando a diversidade de espécies ao redor do mundo. Embora não tenha se tornado nem médico, nem padre, sua teoria de 'seleção natural', publicada em "A Origem das Espécies" [DARWIN, 1859] mudou gradualmente a maneira como as pessoas pensavam sobre evolução. Sua teoria causou *frisson* na sociedade da época, por se confrontar diretamente com o ponto de vista religioso da criação, onde consta que as espécies criadas por Deus permaneceram imutáveis desde sua criação. Charles Darwin baseou-se, também, em trabalhos anteriores, como o de seu pai e de seu avô, além de em suas próprias pesquisas e amostras coletadas ao redor do mundo. Ainda hoje, "A Origem das Espécies" é lido por milhares de cientistas e pesquisadores das áreas de biologia e genética ao redor do mundo.

"Alan Turing" (1912-1954)

Nascido em Warrington Crescent, Inglaterra, Alan Turing foi um matemático e cientista da computação, antes da existência desta disciplina ou mesmo dos computadores. É considerado um dos pais da computação moderna. Já em 1926, Turing afirmava ser booleana a essência da inteligência. Inventou as máquinas de estado, a máquina universal de Turing, uma espécie de precursor dos computadores programáveis, e contribuiu para muitas áreas do conhecimento, como lógica, criptografia, filosofia, biologia, e conceitualmente, para criação de disciplinas como Ciências Cognitivas, Inteligência Artificial e Vida Artificial. Durante os anos finais de sua vida, Turing utilizava o computador Ferranti Mark I da Universidade de Manchester em experimentos com algoritmos que exploravam as semelhanças entre programas digitais e fenômenos como replicação, adaptação e cognição, tópicos que posteriormente viriam a se chamar Vida Artificial e Inteligência Artificial. Durante este período, ele atingiu a distinção de ser o primeiro a se engajar na exploração assistida por computador de sistemas de dinâmica não-linear, uma vez que sua teoria usava equações diferenciais não-lineares para expressar a química do crescimento. Ele morreu no meio de seu trabalho inédito, deixando alguns programas e uma pilha de notas manuscritas, material que até hoje permanece não completamente compreendido.

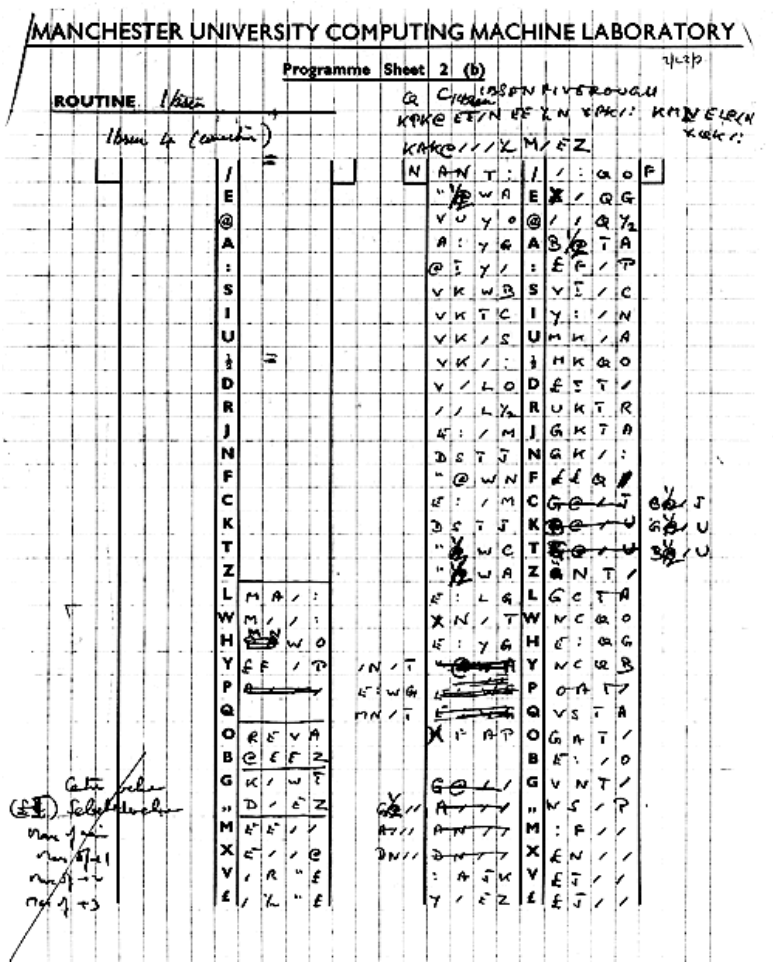

Figura 1. Programa manuscrito por Turing, parte de seu estudo sobre o desenvolvimento do cone de abeto, que já tentava estruturar fenômenos observados na natureza.

“John Von Newmann” (1901-1957)

Nascido na Hungria, John Von Newmann é considerado por muitos o pai da Vida Artificial. É também um dos inventores da Teoria dos Jogos, bem como de muitos conceitos estatísticos envolvidos com importantes aplicações em áreas como economia e engenharia. Newmann foi prodígio em matemática em Budapeste, obteve seu doutorado aos 22 anos, se convertendo aos 23 no professor mais jovem da Universidade de Berlin. Aos 30, junto com Albert Einstein, foi designado como um dos primeiros professores do Instituto de Estudos Avançados de Princeton, Nova Jersey. Ajudou a resolver problemas em mecânica quântica e aspectos ergódigos em matemática, contribuiu com a concepção da bomba atômica e participou de testes

nucleares em Los Alamos, mas, talvez, seja bem mais conhecido pela criação da arquitetura dos primeiros computadores, conhecida como arquitetura de Von Newmann. Na década de 40, Von Newmann integrou a equipe de trabalho da universidade da Pensilvânia, participando dos projetos EDVAC e ENIAC, os primeiros computadores construídos sob o paradigma arquitetônico que duraria mais de 40 anos. Von Newmann também construiu uma estrutura computacional conhecida como Autômato-Celular auto-reprodutivo, conceito que trouxe característica de seres vivos ao universo computacional, explorando as analogias existentes entre sistemas eletrônicos e sistemas vivos, contribuindo para a formação da futura disciplina Vida Artificial. Duas atividades ocuparam sua mente nos últimos anos de sua vida, quando declarou publicamente sofrer de câncer na próstata. A primeira tratava da corrida armamentista e "A Tecnologia da Morte". A segunda, mais abstrata, intitulada "A Tecnologia da Vida", tinha como meta criar uma teoria que compreendesse tanto a biologia natural quanto a artificial. Afrontado com o conceito de auto-reprodução, se perguntava: "Pode uma máquina artificial produzir uma cópia tal de si mesma, que poderia, por sua vez, ser capaz de criar mais cópias?". Em suas palestras na Universidade de Yale sob o tema "O computador e a Mente", afirmava serem computadores e os seres humanos diferentes classes de autômatos.

"Erwin Schrödinger" (1887-1961)

Doutor em física pela Universidade de Viena, Erwin Schrödinger é bem conhecido nos meios acadêmicos como um dos papas no estudo da mecânica quântica, havendo contribuído com equações e leis, das quais muitas levam seu nome. Porém, seus interesses não se limitavam à física nuclear. Em seu livro clássico "O que é vida?" [SCHRÖDINGER, 1943] explora as características físicas das células vivas, comenta sobre a evolução e origens da vida, e especula sobre o futuro evolutivo das espécies, incluindo a humana. Nesse texto se encontra a célebre dedução de que as informações hereditárias se encontram num "sólido aperiódico", uma molécula orgânica dotada de ordem, porém, sem repetição monótona identificada nos cromossomos, corpúsculos presentes no núcleo celular que podiam ser tingidos e

visualizados durante a divisão celular, frase que tem um caráter premonitório à descoberta de Oswald Avery, publicada no ano seguinte, sobre ácidos nucléicos. Schrödinger levantou questões e definiu problemas que se tornariam cruciais na definição das pesquisas genéticas realizadas nas décadas seguintes. Com sua obra de menos de 100 páginas, Schrödinger influenciou e inspirou uma legião de físicos, geneticistas, biólogos e filósofos que o sucederam, não por oferecer respostas às perguntas, mas por fazer as perguntas certas.

“Oswald Avery” (1877-1955)

Nascido em Halifax, Nova Escócia, educado no colégio de médicos e cirurgiões da Universidade de Columbia, o médico e bacteriologista Oswald Avery é melhor conhecido pelas suas descobertas em genética. Foi o primeiro a mostrar que o agente responsável pela transmissão da informação genética não era uma proteína, como acreditavam os bioquímicos, mas ácidos nucléicos, especificamente o DNA. Avery e seus colaboradores extraíram material de uma bactéria de superfície lisa e introduziram em uma bactéria de superfície rugosa. Quando a bactéria de superfície rugosa transformou-se em uma de superfície lisa, constatou-se que a substância extraída continha o gene responsável pela codificação da superfície lisa. Avery e seu grupo purificaram esta substância descobrindo que se tratava de um ácido desoxirribonucléico. Em 1944 Avery publicou seu trabalho, levando a comunidade científica a estudos intensivos sobre o DNA, o que posteriormente o revelou como sendo o agente responsável pela hereditariedade. Seu trabalho fundamentou a descoberta, na década seguinte, da estrutura fundamental da dupla hélice do DNA, e das bases nitrogenadas que o compõem.

“Francis Crick”

Juntamente com “Maurice Wilkins” (1916-), “James Watson” (1928-) e “Rosalind Franklin” (1920–1958), realizou, na década de 50, as pesquisas que levaram a descoberta da estrutura do DNA e das bases nitrogenadas que formam o código genético (Adenina, Timina, Citosina e Guanina). Crick e Wilkins, ambos físicos, em

conjunto com o biólogo James Watson e a química Rosalind Franklin, publicaram em 1953 o trabalho que lhes valeu o prêmio Nobel em 1962. Por ser o premio Nobel atribuído apenas aos vivos, Rosalind Franklin, que padeceu de câncer aos 37 anos não pode ser honrada na premiação. Fancis Crick é também autor de “O Oitavo Dia da Criação”, um tratado universal sobre a história da biologia e talvez o melhor já escrito até hoje.

“Manfred Eigen”

Físico, químico e doutor em ciências naturais pela George-August Universidade de Göttingen, Manfred Eigen trabalhou em experimentos em física, em áreas desde termodinâmica, teoria dos eletrólitos e condutividade térmica, até absorção sonora e reações iônicas rápidas. Por volta de 1960 se interessou por físico-química orgânica, elucidando uma série de passos individuais dos mecanismos de reação catalíticos dos ácidos-base em uma teoria geral, que pode ser verificada experimentalmente. Seu interesse envolveu também questões bioquímicas, que posteriormente passaram a ser seu maior interesse. Abordou questões como as pontes de hidrogênio dos ácidos nucléicos e a dinâmica da transferência de código às enzimas e membranas de lipídio, processos de regulação e controle biológicos e o problema do armazenamento de informação no sistema nervoso central. Na década de 80, Eigen trabalhou na demonstração da transição de substâncias químicas inorgânicas para compostos orgânicos. Seu trabalho é de certa maneira complementar ao trabalho de Leslie Orgel, uma vez que para demonstrar o processo, Eigen utilizou uma enzima de polimeraze, uma proteína catalítica extraída do organismo bacteriófago vivo que orienta estruturalmente a síntese e replicação dos ácidos nucléicos, o que descaracteriza o processo como realmente pré-biótico, embora seus experimentos tenham provido ferramentas fundamentais para o ataque da questão das origens.

“Leslie Orgel”

Nascido em Londres, Inglaterra, professor titular da Universidade de Cambridge, Leslie Orgel é diretor do Laboratório de Química da Evolução no Salk Institute e um

dos pioneiros no estudo das origens da vida na terra. Como Eigen, Orgel é um físico-químico experimental e trabalhou em experimentos de certa forma complementares aos de Eigen, fazendo com que RNA crescesse de nucleotídeos monômeros, sem qualquer molde de RNA para que os monômeros se copiassem, mas utilizando uma enzima de polimeraze para guiar os monômeros. Eigen descobriu que íons de zinco podem servir como catalisadores na síntese do RNA. A diferença entre os trabalhos de Eigen e Orgel é que Eigen construiu moléculas de RNA utilizando uma enzima sem um molde, e Orgel fez o contrário, utilizando um molde sem uma enzima. No RNA das células vivas, ambos devem estar presentes. Para comprovar o aparecimento da vida em condições pré-bióticas, é preciso criar moléculas orgânicas sem utilizar-se nem enzimas nem moldes de polimeraze. Nem Eigen nem Orgel chegaram perto de atingir este objetivo.

“Lynn Margullis”

Uma das mais brilhantes biólogas desta geração, Lynn Margullis leciona no departamento de biologia da Universidade de Massachusetts e coordena o programa de estágio em Biologia Planetária da NASA. Entre inúmeros artigos e mais de dez livros publicados, Margullis estuda a evolução das mitocôndrias, a dinâmica celular e os mecanismos adaptativos que levaram espécies originalmente diferentes a se combinarem para formarem os organismos complexos que observamos hoje. Juntamente com James Lowelock, Lynn Margullis introduziu a hipótese Gaia, uma espécie de teoria da relatividade biológica, que reconhece a terra como um organismo vivo que consiste de milhões de espécies, sutilmente interligadas em um único sistema simbiôntico interdependente, que parte da endo-simbiose microscópica até reconhecer a complexidade associativa do planeta como entidade macroscópica. As conexões efetuadas entre as espécies ao longo da evolução representam uma forma de ‘sinergia’, que Margullis descreve como o processo em que um organismo incorpora outro sem destruí-lo, permitindo que a criatura internalizada procrie como um autômato, adicionando funcionalidades a célula invadida, de forma que o produto final da fusão seja superior à soma das partes individuais. Em seu livro “O Que é Vida?” [MARGULLIS, 1998] Margullis apresenta muitos de seus conceitos e um

pouco de sua pesquisa numa linguagem informal e acessível. Seu novo livro "*Five Kingdoms*" (ainda não publicado no Brasil) explora os mecanismos que levaram a formação dos cinco reinos diferentes a partir de um ancestral comum.

"Christopher G. Langton"

Graduado com duplo mérito em antropologia e filosofia pela Universidade do Arizona, recebeu seu PhD em Ciência da Computação pela Universidade de Michigan, sobre o tema "Computação no limite do Caos". Trabalhou como cientista com o Grupo de Sistemas Complexos da divisão teórica no Laboratório Nacional de Los Alamos, e foi membro da Faculdade Externa do Instituto Santa Fé. É editor responsável pelo "*Artificial Life Journal*" publicado pela MIT Press. Langton organizou o primeiro simpósio de VA em Los Alamos em 1988 e tem sido o principal organizador de muitos dos simpósios posteriores em VA. Sua pesquisa envolve arquiteturas computacionais para estudos em VA, medidas formais de complexidade, sistemas dinâmicos distribuídos, autômatos celulares, morfologia da evolução e as origens da vida. Langton é autor de vários livros sobre o tema Vida Artificial, dentre eles "*Artificial Life: An Overview*" [LANGTON, 1995] que sintetiza a história da pesquisa realizada em vida artificial até recentemente.

Cientistas do campo em atuação

A Tabela 1 traz alguns dos principais cientistas (ordenados alfabeticamente por sobrenome) que trabalham atualmente na questão das origens da vida, bem como suas áreas específicas de atuação. Asteriscos (*) indicam os autores cuja obra ou atual pesquisa contribuíram com este projeto.

Tabela 1. Principais pesquisadores do tema "Origens da Vida"

Cientista	Pesquisa
Cristoph Adami *	Vida Artificial
Gustaf Arrhenius	Absorção mineral por superfícies ativas na formação dos ácidos nucléicos.
Jeffrey L. Bada	Estabilidade, fontes e composição do material orgânico primordial.
André Brack	Química pré-biótica, íons metálicos em peptídeos, origem molecular da vida.
A. Graham Cairns-Smith	Modelo de desdobramento pré-biótico em matrizes de argila.
Melvin Calvin	Assimilação do dióxido de carbono nas plantas.
Thomas R. Cech & Sidney Altman	Ribossomos e a natureza catalítica do RNA.
Francis Crick *	DNA e panspermia dirigida.
Richard Dawkins *	Emergência de complexidade e cognição
Russell Doolittle	Evolução protéica.
Christian de Duve	Organização estrutural e funcional da célula, muita pesquisa abordando as origens da vida.
Freeman Dyson *	Primeiro modelo de metabolismo em VA.
Manfred Eigen *	Polimerização de nucleotídeos *in vitro*, mutação e mecanismos de seleção.
Albert Eschenmoser *	Várias contribuições no estudo das origens da vida.
James P. Ferris	Evolução química pré-biótica.
Hyman Hartman	Evolução dos caminhos metabólicos fotossintéticos.
Sir Frederick Hoyle	Astro-biologia e panspermia.
Gerald Joyce	Bioquímica das enzimas de RNA.
Stuart Kauffman	Teoria da complexidade e modelos NK.
Bernd-Olaf Küppers	Teoria da informação e origem da vida.
Christopher Langton *	Vida Artificial, complexidade, dinâmica discreta.
Lynn Margulis *	Evolução mitocondriana, evolução por simbiose e assimilação.
Stanley Miller	Auto-construção bio-molecular.
Leslie Orgel *	Origens da vida, panspermia dirigida.
Guy Ourisson	Membrana celular e evolução. Presidente da Academia Francesa de Ciências
Juan Oró	Formação pré-biótica da adenina.
Ilya Prigogine	Estruturas dissipativas e auto-organização.
P. Schimmel	Código Genético.
Robert Shapiro *	DNA, análise objetiva de problemas das origens.
Jack Szostak	Pesquisa com RNA e oligonucleotídeos.
Charles H. Townes	Biofísica.
Craig Ventor	Seqüenciamento genético e síntese da vida.
Carl R. Woese	Classificação bacteriológica e evolução.
Hubert P. Yockey	Teoria da informação e biologia molecular.

Pode-se notar que não há uma formação preferencial para pesquisadores de VA, entre os nomes citados, encontram-se físicos, químicos, biólogos, filósofos e engenheiros, dado que algumas questões abrangem áreas de engenharia de sistemas e ciência da computação, enquanto outras se voltam à biologia e estudos das origens da vida, muitas ainda têm cunho predominantemente filosófico.

Figura 2. Hall de predecessores ilustres.

2.3. Vida Artificial e sua Abrangência

Na seção anterior, foi ressaltado que embora qualquer pessoa possa dizer facilmente a diferença entre uma coisa “viva” e uma “não viva” apenas olhando para ela, nenhum passo deve ser dado sem antes pesquisar minuciosamente os processos envolvidos no que caracteriza um comportamento como “vida”. Esse, talvez, seja o principal e mais forte apelo dos estudos em VA. É muito difícil observar um bom programa de VA em ação sem perceber que, de alguma forma, ele vive, ou ao menos apresenta muitas características de um ser vivo. Simulações em VA têm mudado a maneira como enxergamos a vida, e, conseqüentemente, despertado o interesse em círculos mais específicos de estudos, como a “astro-biologia”, por ser capaz de extrair propriedades da vida independentes das manifestações atualmente disponíveis para estudos.

Como ressaltou o Físico Marcelo Gleiser em sua coluna científica semanal no jornal Folha de São Paulo com o título “Repensando a nossa existência”, *“A vida é um experimento em complexidade: conhecemos os ingredientes, os vários compostos químicos que fazem parte dos seres vivos, mas não sabemos como combiná-los para formar sequer o mais simples deles”*. Embora muitas tentativas tenham sido feitas, algumas atingindo um certo grau de proximidade, até o momento, nenhum cientista foi capaz de recriar o processo que deu surgimento às primeiras formas de vida, e, dada a complexidade do processo, variedade e possíveis combinações dos átomos envolvidos, parece incrível que tal fenômeno tenha surgido espontaneamente, sem qualquer “ajuda” externa, na natureza, através de funções puramente estatísticas. Este enorme “salto” da química para a biologia intriga muitos pesquisadores. Alguns propõem uma origem “extraterrestre” para a vida, como Orgel e Dyson, que especulam que a vida pode ter surgido na terra proveniente de material presente em meteoros, como clama a teoria conhecida como “*panspermia*”. Mas caso a vida tenha realmente surgido na terra e se tal improbabilidade estatística ocorreu aqui e em tantas instâncias biologicamente diferentes, é natural esperar que tal fenômeno possa ser observado e reproduzido em ambientes alternativos, sejam eles reais, virtuais ou extraterrestres.

O aumento da complexidade, ou “auto-organização”, parece ser fenômeno físico natural. Como é observado na evolução da vida terrestre, tão bem quanto em experimentos de VA, existe na natureza uma tendência inexorável à formação de organismos cada vez mais complexos, culminando talvez em formas de vida inteligentes, como defende Richard Dawkins, em seu livro “O Gene Egoísta” [DAWKINS, 1976], onde afirma ser essa escalada de complexidade a rota usual de toda e qualquer forma de vida, em âmbito universal.

Nas palavras do cientista Christopher Langton, “*A coisa mais importante a se lembrar sobre VA é que a parte artificial não é a vida, mas os materiais. Coisas reais acontecem. Fenômenos reais são observados. É vida real em uma mídia artificial*”.

2.4. Problemas Abertos em Vida Artificial

Algumas questões são atualmente consideradas questões ‘chave’ no estudo de VA como disciplina. Visando direcionar os esforços na solução destas questões, o artigo de Bedau *et al*. [BEDAU, 2000] relaciona alguns dos problemas abertos em VA, comentando a relevância do estudo de cada tópico. Os tópicos relacionados são:

Como a vida surge da não-vida?

1. Gerar um proto-organismo molecular *in vitro;*
2. Atingir a transição para vida de um composto químico artificial *in silico;*
3. Determinar fundamentalmente se um organismo fictício pode existir;
4. Simular um organismo unicelular em todo o seu ciclo de vida;
5. Explicar como regras e símbolos são gerados da dinâmica física em sistemas vivos;

Quais são os potenciais e limites dos sistemas vivos?

6. Determinar o que é inevitável na evolução aberta da vida;
7. Determinar as condições mínimas para transições evolucionárias de sistemas de resposta específicos para genéricos;
8. Criar um modelo formal para sintetizar hierarquias dinâmicas em todas as escalas;
9. Determinar a previsibilidade das conseqüências evolutivas da manipulação de organismos e ecossistemas;
10. Desenvolver uma teoria de processamento, fluxo e geração de informação para sistemas em desenvolvimento;

Como a vida se relaciona com a mente, as máquinas e a cultura?

11. Demonstrar a emergência de inteligência e mente em um sistema de vida artificial;

12. Avaliar a influência de máquinas na próxima transição evolutiva da vida;
13. Prover um modelo quantitativo da conexão entre evolução biológica e cultural;
14. Estabelecer princípios éticos para experimentos em vida artificial.

Pesquisas envolvendo o “estado da arte” em VA geralmente focam a solução de um ou mais destes problemas. Porém, os frutos colhidos com a pesquisa em VA muitas vezes envolvem a aplicação dos resultados obtidos para solução de problemas em outras áreas da ciência e engenharia. A exemplo dos métodos de busca evolutiva com algoritmos genéticos, aplicados amplamente na solução de problemas tecnológicos, vêm ganhando muito com a pesquisa em genética, utilizando-se de VA para reproduzir seus conceitos. O número de artigos encaminhados a congressos e simpósios sobre o tema aumentou muito desde a primeira conferência realizada em Los Alamos em 1987, demandando uma quantidade cada vez maior de eventos para cobrir o trabalho desenvolvido. Isso demonstra que muito tem sido feito em matéria de pesquisa e muitos pesquisadores de diversas áreas se fascinaram pelo assunto. Os tópicos cobertos atualmente compreendem uma vasta lista que se estende desde astro-biologia até aplicações em microeletrônica, robótica e estratégia de jogos.

2.5. Linhas de Pesquisa em Vida Artificial

A tentativa de recriação do fenômeno da vida pelo homem, disciplina que recebeu o nome de VA, pode tomar rumos radicalmente diferentes em sua pesquisa, que entendem desde a emulação e simulação de características de seres vivos em sistemas computacionais até a sua forma mais pretensiosa, recriação de formas de vida em silício ou carbono. Tendo em vista que a ciência falhou em recriar a vida de condições químicas e físicas consideradas como pré-bióticas, a alternativa foi conceber formas elementares de mecanismos e processos que podem ser classificados como potencialmente vivos, apresentando muitas das características e fenômenos presentes nos organismos vivos. Para tanto, os cientistas e pesquisadores de VA utilizaram os aparatos tecnológicos que tinham em mão, ou os que dominavam melhor. O resultado foi uma ampla diversidade de aproximações ao

tema, que utilizam os mais variados ramos da ciência para tal propósito. Algumas das linhas de pesquisa são:

- Origem da vida, auto-organização e auto-replicação;
- Desenvolvimento e replicação;
- Dinâmica evolucionária e adaptativa;
- Robôs e agentes autônomos;
- Comunicação, cooperação e comportamento coletivo;
- Simulação em VA, ferramentas de síntese e metodologias.

Entre as linhas supracitadas, conhecimentos em matemática, ciência da computação, engenharia, física, química e biologia são aplicados, enfocando abordagens científicas diferentes em cada caso. Porém, todas as pesquisas têm em comum o uso de simulações em computadores para obter seus resultados.

Das primeiras e mais comuns entre as linhas de pesquisas em VA, os experimentos com máquinas de estado conhecidas como autômatos celulares foram dos mais explorados. Ainda hoje, muitas das pesquisas em desenvolvimento empregam variações dos conceitos fundamentais da teoria de autômatos. O capítulo 3 traz uma introdução aos conceitos de máquinas de estado e autômatos celulares.

O uso de redes de mapas acoplados ou CML (*Coupled Map Lattices*) foi proposto como alternativa à aplicação da teoria de autômatos em muitas áreas. Redes de mapas acoplados utilizam em cada célula, um gerador caótico, e, através de um acoplamento com sua vizinhança próxima, produz complexidade além da observada em autômatos celulares, com a vantagem de poder ter sua “agitação” ou “temperatura” ajustada através dos parâmetros das equações dos geradores caóticos. Para exemplos de implementação de CML e geradores caóticos, o sítio do projeto apresenta exemplos e Applets para experimentação [ALIVE SITE].

Programas de VA que abordam a adaptação e a evolução, explorando a dinâmica evolucionária e adaptativa tornaram-se muito comuns e populares. Hoje, existem

centenas de programas deste tipo, que podem ser encontrados através de uma simples busca pela Internet.

A próxima seção traz alguns dos trabalhos pioneiros e mais conhecidos em algumas das áreas de pesquisa em VA.

2.6. Trabalhos Notórios em Vida Artificial

Aqui são relacionados alguns trabalhos notórios no estudo de VA, e possuem um grande número de citações em publicações da área. Seguem algumas pesquisas conhecidas na área de vida artificial:

“Tierra”

Thomas Ray, ecólogo norte-americano, em 1990 criou o primeiro software capaz de simular os mecanismos evolutivos seguindo os princípios darwinianos. O programa “Tierra”, disponível em seu sítio na Internet [RAY SITE], simula diversos tipos de programas simples disputando espaço na memória do computador, replicando-se e sofrendo mutações aleatórias que podem ou não favorecê-los na disputa pela propagação de seu genoma.

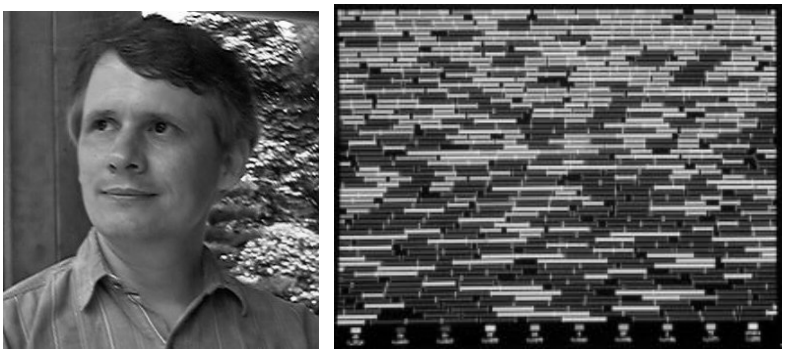

Figura 3. Thomas Ray e a tela do programa Tierra.

“The Blind Watchmaker”

Richard Dawkins, zoologista da universidade de Oxford, criou um simples e popular programa de VA para computadores pessoais chamado “O Relojoeiro Cego” (“The Blind Watchmaker”), que demonstra o poder inerente da evolução darwiniana para criar elaborados padrões e estruturas. O programa cria padrões de traços branco-e-preto que se ramificam de acordo com regras muito simples. O programa acompanha o livro de mesmo nome ou pode ser obtido gratuitamente na Internet.

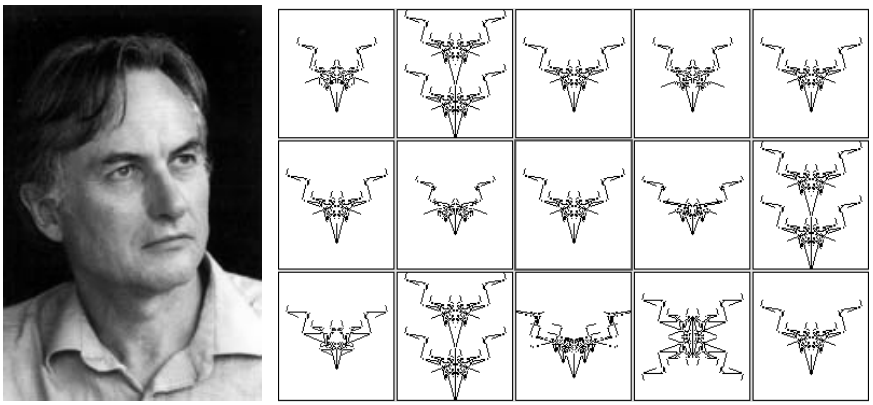

Figura 4. Dawkins e seu programa “The Blind Watchmaker”.

“Robôs-Insetos”

Rodney Brooks, diretor do Laboratório de Inteligência Artificial do MIT, construiu com sua equipe robôs autônomos chamados de “robôs-insetos” utilizando-se dos princípios fundamentais de VA. Seguindo a filosofia: “Rápidos”, “Baratos”, e “Fora de Controle”, ele constrói robôs que se assemelham a formigas, e desempenham uma série de funções elementares. Os robôs são construídos com uma rede neural iniciada aleatoriamente, que se adapta posteriormente com o aprendizado. O objetivo de Rodney é futuramente construir robôs extremamente simples e baratos que desempenhem funções úteis coletivamente, se espalhando por todos os cantos, como em uma cena de filme de ficção científica, ágeis e pragmáticos, tornando-se aliados indispensáveis nas tarefas cotidianas. Suas últimas pesquisas podem ser visitadas no sítio do laboratório na Internet [BROOKS SITE].

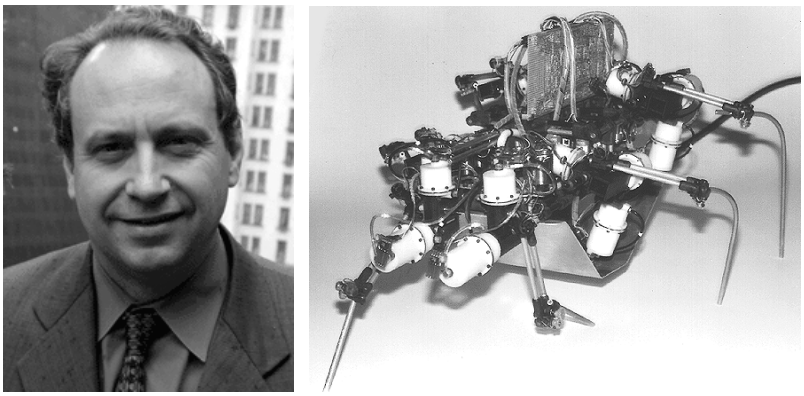

Figura 5. Os robôs-inseto de Rodney Brooks.

“Avida”

Cristoph Adami, do Laboratório de Vida Artificial do MIT, criou este popular programa de vida artificial, que explora as capacidades evolutivas de uma população de genótipos diferentes em uma grade bi-dimensional. O programa de Adami lembra muito o pioneiro Tierra, do ecólogo Tomas Ray, embora apresente algumas diferenças fundamentais. Avida aplica muito da teoria de autômatos celulares, onde as regras de reprodução e os “programas” que controlam os autômatos evoluem à medida que o experimento se desenvolve. O sítio do projeto na Internet apresenta as últimas versões do programa e alguns experimentos desenvolvidos por Adami [ADAMI SITE].

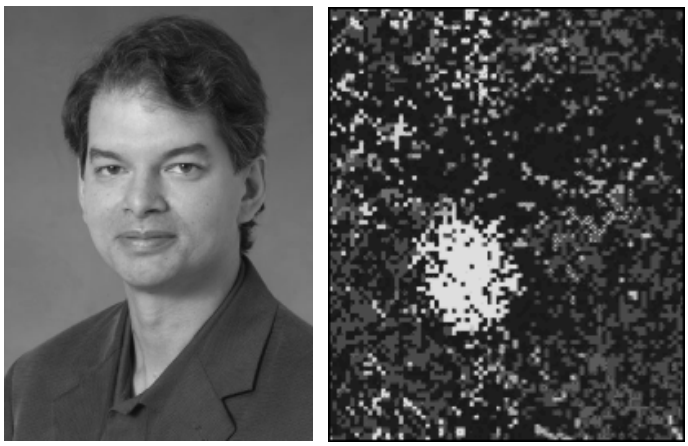

Figura 6. Tela do programa Avida de Adami.

“Evolução Morfológica”

Karl Sims, especialista em computação gráfica, criou uma série de experimentos em que visa explorar não somente o fenômeno emergente, mas as conseqüências de tais fenômenos dadas às ferramentas evolutivas disponíveis e as limitações impostas pelo ambiente. Para tanto, Sims reuniu um conjunto de conceitos simples e fundamentalmente importantes: algoritmos simples, redes neurais evolutivas, e seleção do melhor adaptado. Os experimentos de Sims são por muitas razões impressionantes, dada a riqueza visual e a variedade de comportamentos e morfologias observadas em suas criaturas construídas com blocos móveis, que muitas vezes lembram espécies existentes na natureza. Sims direciona seu trabalho à computação gráfica, obtendo incríveis “imagens genéticas” utilizando técnicas de VA. Seus experimentos podem ser acessados no seu sítio pela Internet [SIMS SITE].

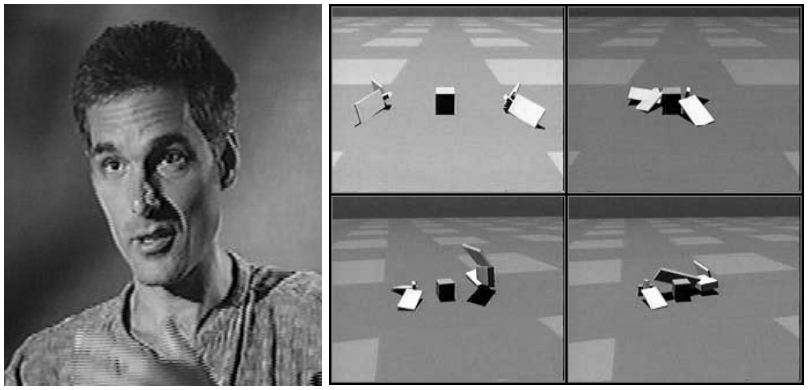

Figura 7. Experimento de Karl Sims com morfologia evolutiva.

3. SIMULAÇÕES EM VIDA ARTIFICIAL

Simular computacionalmente um ser vivo trata de expressar os comportamentos deste ser através de regras procedimentais. Em suma, escrever um programa que de respostas semelhantes aos estímulos, ou permita que estas sejam atingidas espontaneamente após um tempo de simulação. Duas abordagens podem ser dadas a esse aspecto: "*Bottom-up*" e "*Top-down*".

"*Bottom-up*", ou de baixo para cima, é o método observado na natureza, onde mecanismos são fornecidos para que os seres se adaptem às regras impostas pelo ambiente, deixando que a adaptação cuide de desenvolver a complexidade necessária a sua sobrevivência. A maioria da pesquisa científica realizada em VA sobre emergência de complexidade e origens da vida se enquadra nesta classe.

"*Top-down*", ou de cima pra baixo, é a forma adotada na engenharia, onde o problema é estudado sistematicamente e as soluções apresentadas de maneira direta. A solução visa, unicamente, resolver o problema proposto e nada garante que esta solução seja a melhor para o dado problema. Abordagens como essa não levam em conta fatores como mudança repentina das regras, o que levaria a necessidade de uma nova especificação.

Ainda assim, para muitos casos, a abordagem *Top-down* pode ser aplicada na simulação de seres vivos sem grande comprometimento do realismo da simulação. Um caso específico é o de criação de personagens cinematográficos em computação gráfica, como apresentado no filme "O Rei Leão" dos estúdios Disney, onde um estouro de uma manada e representada com extremo realismo. Para implementar tal cena, esperar que os búfalos artificiais atinjam naturalmente o padrão de movimentação de búfalos verdadeiros pode ser demais arrojado para o projeto. Uma implementação "comportamental" é mais indicada.

Exemplificando o caso acima, em uma arquitetura *Top-down*, desejamos simular em computação gráfica o comportamento de uma vaca. Relacionamos então as ações

possíveis de uma vaca, como mugir, pastar, andar, virar, beber água, etc. seguidas das probabilidades de cada ação em um contexto cronológico, excluindo seqüências de ações inconsistentes. Programando tais movimentos em um personagem de computação gráfica, obtém-se o comportamento visual necessário para se simular uma vaca, ao menos pelo surgimento de possíveis inconsistências, como um movimento errático, ou ações repetidas constantemente. Em um contexto populacional, onde cada vaca segue tal comportamento de maneira probabilística, dificilmente seria notado o surgimento de padrões “anormais” de comportamento.

Como exemplo de uma implementação *Bottom-up*, o mesmo personagem, a vaca, receberia uma gama de comportamentos e probabilidades aleatoriamente associadas, aos quais seriam atribuídos pontos para manifestações coerentes com o comportamento real observado. De tempos em tempos, a população seria avaliada, mantendo-se os caracteres com maior pontuação, recombinando e adaptando constantemente a população até se atingir os resultados esperados.

Em ambos os casos, a arquitetura de estímulo e resposta dos personagens é semelhante, podendo ser representado através de um diagrama de blocos. As diferenças se encontram na implementação dos blocos de estímulo, processamento e resposta. A Figura 8 mostra um possível diagrama de entrada e saída para um ser virtual.

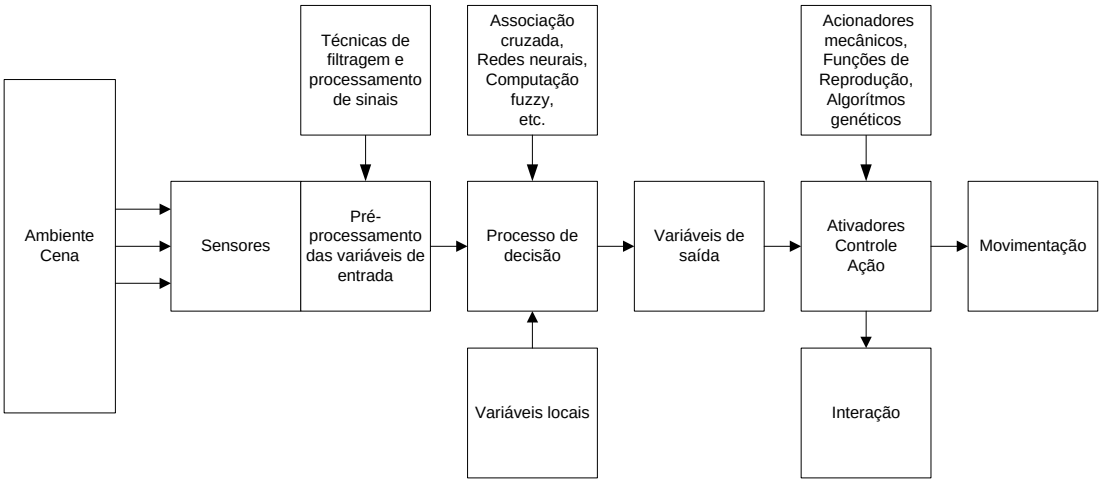

Figura 8. Mecanismo de processamento e resposta aos estímulos ambientais.

A associação estímulo-resposta é implementada através de associações presentes nos cromossomos. Por exemplo, uma rede neural também pode ser iniciada aleatoriamente, passando por processos genéticos, que oferecem a possibilidade de posterior adaptação e aprendizado. Lógica Nebulosa (*fuzzy*) é comumente empregada na tomada de decisão em sistemas com linguagem natural (humanística).

As etapas consideradas na criação de um experimento em vida artificial são:

- Criação do universo físico;
- Criação de mecanismos de sensoriamento;
- Criação de mecanismos de processamento de estímulos;
- Criação de mecanismos de resposta mecânica e ação;
- Criação de mecanismos genéticos e de reprodução.

O universo físico trata da interação entre os objetos e agentes presentes no ambiente simulado. Cálculo vetorial e conceitos de mecânica clássica são aplicados para determinar o movimento dos corpos físicos em um espaço, que a princípio pode ter qualquer dimensão.

Os mecanismos de sensoriamento determinam como os organismos “sentem” o ambiente que habitam. Em termos técnicos, tratam das variáveis de entrada, que contém a informação que recebem do ambiente. O sensoriamento determina como um organismo se relaciona com os outros organismos, objetos e substâncias presentes, permitindo a noção de presença, percepção de temperatura ou de concentrações de substâncias existentes em sua volta.

Os mecanismos de processamento de estímulos podem empregar uma série de técnicas para obter associações que levem das variáveis de entrada às ações produzidas em resposta. Alguns exemplos são: processamento de sinais por redes neurais, lógica nebulosa, autômatos e máquinas de estado.

Uma vez obtidas, as variáveis de saída correspondem a ações no universo físico. Os dispositivos de resposta mecânica controlam as ações desempenhadas no ambiente virtual pelo organismo, em resposta a comandos de controle fornecidos pelos estágios anteriores. Por exemplo, acionadores ou propulsores desempenham a função de movimentar os agentes presentes no experimento, porém, restringindo sua liberdade de movimentação às regras físicas estipuladas.

Operadores genéticos devem estar associados aos agentes na reprodução, visando tirar proveito dos mecanismos adaptativos e evolutivos inerentes a este conceito. Várias técnicas podem ser aplicadas aqui, sendo as mais utilizadas a mutação, muito parecida com o mecanismo de reprodução assexuada e o cruzamento (*cross-over*), associado à reprodução sexuada. Ambos serão explicados em mais detalhes na seção sobre algoritmos genéticos.

Um fator importante a ser considerado nas simulações é a introdução de quantidades específicas de ruído. Em casos como reprodução, o ruído introduz uma desordem estatística no sistema, permitindo que este se re-organize de forma mais eficiente. Valores baixos de ruído são recomendados (da ordem de 1% no caso do operador de mutação), valores extremos de ruído podem desordenar completamente o sistema, levando-o a um estado aleatório.

3.1. Simulando Vida

Quando desejamos sintetizar um conjunto de normas ou reações que caracterizam um organismo como vivo, devemos considerar tanto fatores que dependem da natureza do organismo simulado, quanto fatores universais. Para se obter resultados válidos, as simulações em VA devem seguir alguns princípios fundamentais em seus experimentos, visando evitar uma “condução intencional” que leve ao resultado desejado. Entre tais princípios, podemos citar:

- **Considerar regras locais em vez de regras globais**: Salvo os princípios físicos que regem o sistema, cada instância deve considerar apenas estímulos próximos,

ou seja, ocorrências observadas próximas da instância do organismo em questão influenciam mais que eventos distantes. Parte do princípio que dificilmente algo que aconteça a grande distância venha a afetar o organismo avaliado. Uma análise global sobre o ambiente para cada instância de organismo virtual presente na simulação também encarece computacionalmente o experimento, exigindo processamento além do necessário.

- **Considerar regras simples em vez de complexas**: pode levar a resultados mais satisfatórios. A complexidade surge da reprodução e combinação de uma grande quantidade de regras simples, conforme observado na natureza. Empregar regras complexas pré-definidas como forma de obter a complexidade comportamental esperada acaba por definir um comportamento "viciado", com pouca chance de adaptação ou mudança ao longo das gerações.
- **Considerar comportamentos emergentes em vez de comportamentos pré-especificados**: é certamente o mais importante princípio, e provavelmente, o mais cientificamente controverso. Regras pré-especificadas causam sistematização. Formas de vida dificilmente agem de maneira pré-estipulada aos estímulos que recebem, reagindo de maneira idêntica a uma série desses estímulos. Especificar um caractere programaticamente pode torná-lo aparentemente animado, porém tão vivo quanto um personagem de computação gráfica dos desenhos de Disney.

Como exemplo, podemos recorrer a uma simulação comum em Ciências da Computação: Algoritmo de Formigas. Formigas formam sociedades extremamente desenvolvidas, com funções bem definidas e distribuídas. Porém, uma formiga individualmente é uma criatura extremamente estúpida. Entomologistas afirmam que cada formiga possui entre 50 e 500 neurônios (dependendo da espécie considerada e função no formigueiro), e podem desempenhar na média 40 funções específicas. Mas essas funções, quando desempenhadas no tempo correto e disparadas em resposta ao estímulo certo, em grupo, operam maravilhas, demonstrando como um conjunto de regras simples, podem resolver problemas complexos, como, por exemplo, encontrar o menor caminho entre um conjunto de pontos, propósito para o qual tal algoritmo é comumente empregado.

A maneira como a complexidade do formigueiro aumenta com o passar do tempo, baseado unicamente em regras simples, pode surpreender qualquer arquiteto. A estrutura que atinge uma incrível “ordem a partir do caos” gerando algo do nada parece ser um dos segredos físicos da vida, onde estudo de VA pretende lançar luz.

Existem formigueiros por todo o mundo, embora nenhum seja exatamente igual ao outro. Isso porque formigueiros não são construídos por programas, sem nenhum planejamento ou inteligência. Uma formiga sente instintivamente a necessidade de cavar, bloquear o sol, criar caminhos; outras formigas a vêem trabalhar e se juntam a ela, imitando-a. Tal processo é conhecido como “Alelomimese”, quando instintivamente o mesmo padrão de comportamento é transmitido através do grupo.

Princípios como o citado, onde padrões surgem espontaneamente, são largamente observados em simulações de VA. Processos como interações entre presa e predador, efeitos parasitários e interdependência na dinâmica populacional, aparentemente se manifestam em várias instâncias e suas regras se aplicam a vários fenômenos internos e externos dos seres vivos, como crescimento de plantas, formação de corais e regeneração de tecidos. A lista é infindável. As pesquisas sobre a dinâmica envolvidas nos processos podem levar futuramente a obtenção de regras genéricas que regem esses sistemas de modo geral.

Dos experimentos mais intrigantes em complexidade, os com “Autômatos Celulares” talvez sejam os mais implementados por sua simplicidade matemática e funcional. Autômatos são comumente aplicados nos geradores randômicos dos computadores e equipamentos eletrônicos. Stephen Wolfram, físico e criador do Software ‘best-seller’ “Mathematica” é um dos mais fascinados pela idéia de que algumas regras simples podem gerar padrões de complexidade inacreditável. Em seu livro “A New kind of Science” [WOLFRAM, 2002] de mais de mil páginas e pesadamente ilustrado, ele afirma ser possível reproduzir uma enorme quantidade de processos físicos e biológicos utilizando-se apenas variações das regras, estruturas e condições iniciais em experimentos empregando somente autômatos celulares de dinâmica

discreta. Wolfram tem sofrido pesadas críticas da comunidade científica, principalmente por não haver mostrado experimentos que sustentassem suas afirmações.

3.2. Representação da Dinâmica dos Seres Artificiais

Aqui são descritos, sucintamente, algumas técnicas ferramentais e modelos matemáticos geralmente empregados na simulação de agentes em VA, bem como fatores que devem ser considerados em relação à dinâmica dos seres vivos em simulações de tempo discreto. As ferramentas, empregadas nas simulações não devem se limitar às apresentadas aqui, e podem, de maneira geral, utilizar quaisquer técnicas, metodologias e operadores numéricos disponíveis, dependendo apenas da criatividade do programador em aplicá-las nos modelos simulados. Aqui se encontram apenas algumas referências introdutórias às mesmas.

3.2.1. Dinâmica de Tempo Discreto

Para se efetuar as operações funcionais de forma computacionalmente barata, deve-se aplicar uma dinâmica de tempo discreto às operações realizadas no ambiente virtual. Uma dinâmica de tempo discreto sugere que o instante atual é calculado a partir do estado de uma ou mais variáveis no instante imediatamente anterior. De forma que, para uma variável *X* em tempo discreto, temos:

$$X_{T+1} = F\left(X_T\right)$$

A definição envolve a noção de recursão. Na função, o tempo deixa de ser uma variável do sistema, passando a ser meramente um rótulo simbólico, contado o número (inteiro) de interações realizadas desde a condição inicial (X_0). No contexto apresentado, a função é dita recursiva, i.e., re-aplicada sucessivamente sobre os valores obtidos pela própria operação.

As vantagens de utilizar-se dinâmica discreta em relação a equações diferenciais ordinárias ou funções em tempo contínuo são facilmente percebidas, tanto do ponto

de vista computacional, onde operações recursivas elementares com variáveis numéricas substituem o uso de literais e regras matemáticas intrincadas, quanto do ponto de vista arquitetural, onde normalmente basta considerar o estado imediatamente anterior para se estabelecer uma regra.

Técnicas para transformar funções e equações diferenciais para sistemas dinâmicos de tempo discreto podem ser encontradas em livros de cálculo numérico, matemática aplicada à engenharia [KREYSZIG, 1998] ou de algoritmos para computação. Alguns dos métodos mais usados são: Runge-Kunta, método de Euler e esquema do trapézio, entre outros.

3.2.2. Máquinas de Estado

As máquinas de estado, como o nome sugere, têm seu estado atual determinado a partir de um ou mais dos estados anteriores e condições de transição entre estados. Uma máquina de estado pode ser representada por um grafo de estados, onde cada nó indica o estado em que se encontram variáveis discretas representativas de características funcionais, testes ou decisões, levando a um novo estado no instante seguinte.

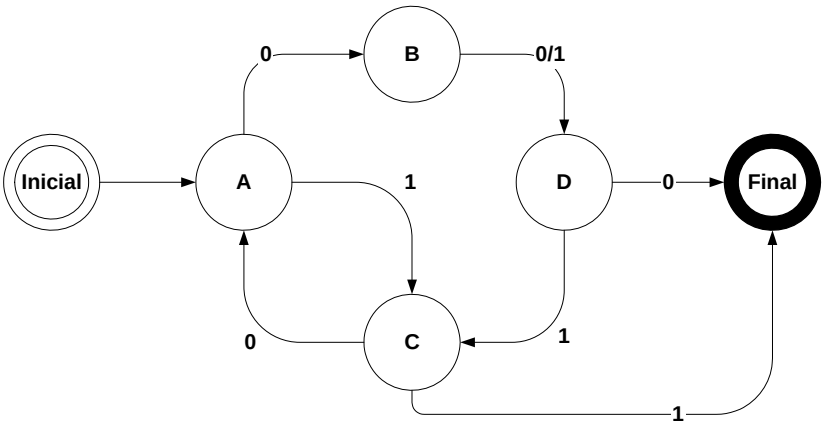

Figura 9. Exemplo de um grafo descrevendo uma máquina de estados.

As variáveis discretas ou resultados de testes efetuados determinam o número de estados possíveis do sistema, portanto, para 3 bits, teremos 8 estados possíveis. Os nós efetuam operações e modificam os estados das variáveis, determinando o estado seguinte que o sistema vai tomar. Uma máquina como a apresentada no grafo da

Figura 9 pode ser expressa em uma tabela e implementada em um programa através de relações do tipo IF-THEN-ELSE. A Figura 10 mostra como o comportamento de um agente pode ser definido por uma máquina de estados.

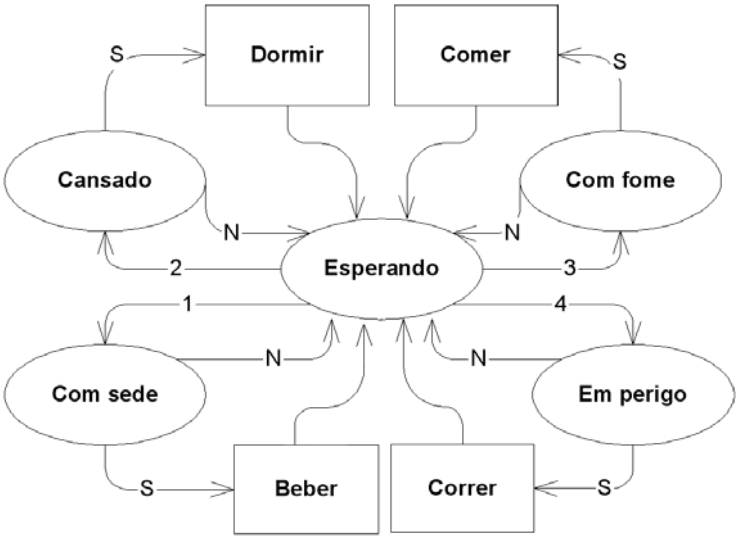

Figura 10. Possível máquina de estados para controle de um agente.

Entre os sistemas de dinâmica discreta, os Autômatos Celulares estão entre os mais implementados. Numa grade de dimensões arbitrárias de números binários, o estado seguinte de cada bit é determinado pelo seu estado atual e pelo estado da vizinhança próxima. O autômato celular é uma máquina de estados onde o número de estados é definido pelo número de bits avaliados no processo de decisão de cada célula. Apesar de sua simplicidade, as regras apresentam uma enorme variedade de comportamentos distintos. No seu exemplo mais simples, com uma dimensão, o autômato celular pode assumir 256 variações, para um conjunto de três bits de entrada e um de saída. A Tabela 2 descreve uma regra possível para o autômato celular de uma dimensão (regra 90).

Tabela 2. Máquina de estados do Autômato Celular 1D, regra 90.

Vizinho à esquerda	Estado atual	Vizinho à direita	Novo estado
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

A tabela traz os estados possíveis (8 estados) para o autômato e seus dois vizinhos. Os novos estados, de cima para baixo, definem o número da regra, no caso 01011010 = 90 em decimal. A Figura 11 traz algumas representações visuais para três regras aplicadas. Na representação gráfica branco representa 0 e preto 1, uma linha horizontal indica o estado da seqüência de bits a cada instante. Novos instantes calculados adicionam linhas abaixo na figura.

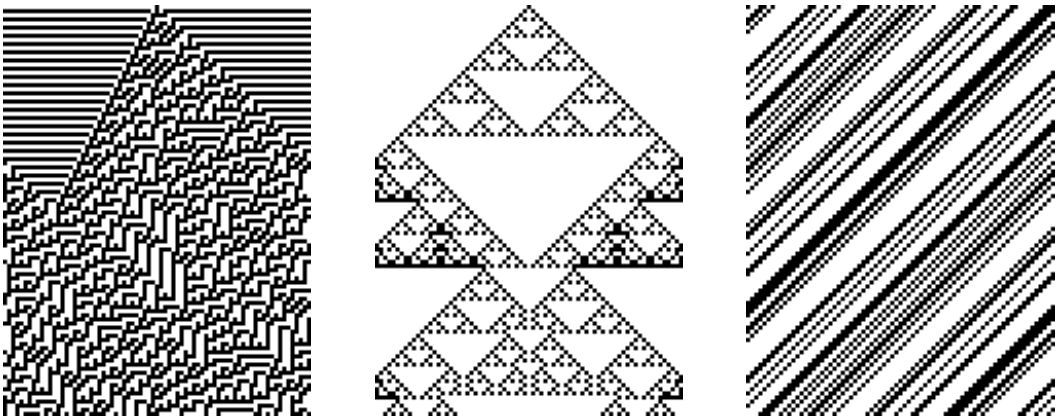

Figura 11. Grade de autômatos celulares de uma dimensão para as regras 45, 90 e 170 respectivamente.

Os dois primeiros exemplos utilizaram como estado inicial uma seqüência de zeros, com um único bit = 1 no meio da seqüência; o último exemplo foi iniciado aleatoriamente. A primeira regra apresenta uma repetição monótona, a menos que um dos bits seja diferente dos demais, o que leva o sistema a uma total desestabilidade. A segunda regra, batizada de Sierpinsky apresenta uma geometria fractal, onde triângulos se completam até a unidade mais elementar, no caso, o pixel. A regra 170

é conhecida em eletrônica como operador “deslocamento à esquerda” (*left-shift*); o “deslocamento à direita” (*right-shift*) é obtido com o uso da regra 240.

Os autômatos celulares podem ter uma ou mais dimensões. Para o caso do autômato de 2 dimensões com vizinhança de 4, teremos $2^4 = 16$ estados possíveis, totalizando $2^{16} = 65536$ regras possíveis. Para uma vizinhança de oito casas, as regras possíveis são da ordem de 134×10^{152}. Um caso interessante de autômato celular de duas dimensões é o popular “Jogo da Vida” criado pelo matemático John Holton Conway, onde para cada célula, com vizinhança de oito casas, considera-se apenas o número de vizinhos “vivos”, ou com valor binário 1. As regras do “Jogo da Vida” são:

- Uma célula viva com 2 ou 3 vizinhos vivos permanece viva;
- Uma célula viva com menos de 2 vizinhos morre de solidão;
- Uma célula viva com mais de 3 vizinhos morre sufocada;
- Uma célula morta com 3 vizinhos vivos passa a viver.

Apesar das regras elementares, o resultado visual do sistema é extremamente complexo, e os fenômenos observados imprevisíveis, mesmo para uma grade de pequenas dimensões, dada a complexidade das interações. Dependendo do tamanho da grade e das condições iniciais, o sistema pode demorar um grande número de ciclos para se estabilizar. Um único bit mudado nas condições iniciais e o estado final pode ser completamente diferente, o que caracteriza alta sensibilidade às condições iniciais, requisito para classificar o sistema como caótico.

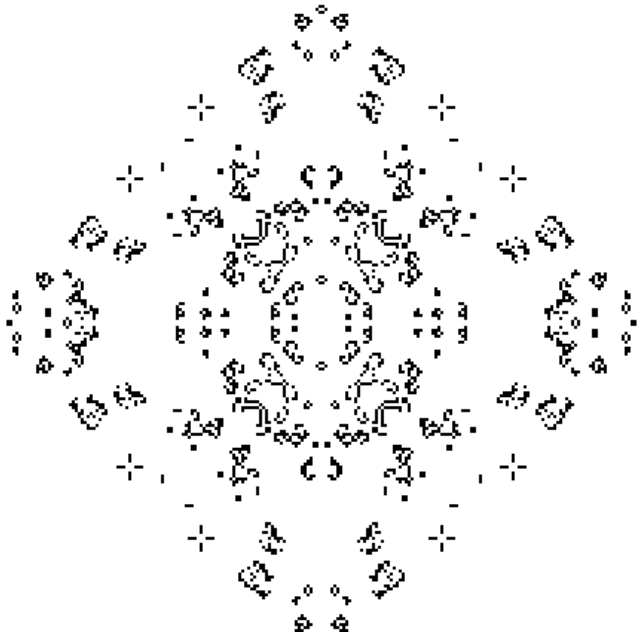

Figura 12. O "Jogo da Vida" de John Conway.

Outro exemplo de máquina de estado é a máquina universal de Turing, considerada uma das primeiras abstrações matemáticas que levaram à construção da moderna arquitetura dos computadores. Na máquina de Turing, uma fita contém as informações em uma seqüência binária, e as regras descrevem quais operações devem ser realizadas considerando o estado atual. Os estados possíveis são lendo (0 ou 1), gravando (0 ou 1), movendo para a direita, movendo para a esquerda, parado, e estado final.

3.2.3. Sistemas Não-Lineares com Dinâmica Caótica

Alguns sistemas dinâmicos de tempo discreto apresentam um comportamento peculiar para alguns valores de parâmetros, definido como comportamento caótico ou simplesmente caos. Comportamentos caóticos podem ser observados também na natureza ou em alguns sistemas mecânicos. Para que um sistema seja descrito como caótico deve apresentar as seguintes características:

- Aperiódico. O sistema não apresenta qualquer tipo de repetição, não re-visitando seqüências de valores já assumidos, o que levaria a uma periodicidade.

- Limitado. Os valores assumidos sempre se encontram em um determinado limite máximo e mínimo, não explodindo para o infinito ou menos infinito em nenhum momento.
- Determinístico. Há uma regra definida governando o sistema, sem aleatoriedade, logo, para uma dada condição inicial dos parâmetros, os valores visitados são sempre os mesmos, bem como sua seqüência.
- Alta sensibilidade às condições iniciais. Uma pequena variação em algum parâmetro gera uma seqüência nova que, em longo prazo, apresenta valores que diferem em muito da seqüência inicialmente considerada.

Algumas das equações dinâmicas mais conhecidas e simples que apresentam comportamento caótico são o “mapa logístico”, o “mapa de tenda” (Tent Map) e o “mapa seno-circular” (Sine-Circle Map). A equação do mapa logístico é dada por:

$$X_{T+1} = \mu . X_T (1 - X_T)$$

O μ representa o parâmetro e o X_0 é a condição inicial (X em T=0) que determina a seqüência definida pela função recursiva. A figura abaixo apresenta os valores visitados pela série temporal para valores de μ entre 0 e 4, intervalo onde a equação é estável, apresentando as características necessárias para defini-la como caótica. Esta figura é conhecida como diagrama de bifurcação, e determina, estatisticamente, os valores visitados pela série temporal conforme a variação dos parâmetros. No caso do mapa logístico, com apenas um parâmetro, o diagrama de bifurcação tem apenas duas dimensões. Mapas com mais de um parâmetro apresentam graus de caos em duas ou mais dimensões, tornando difícil sua representação.

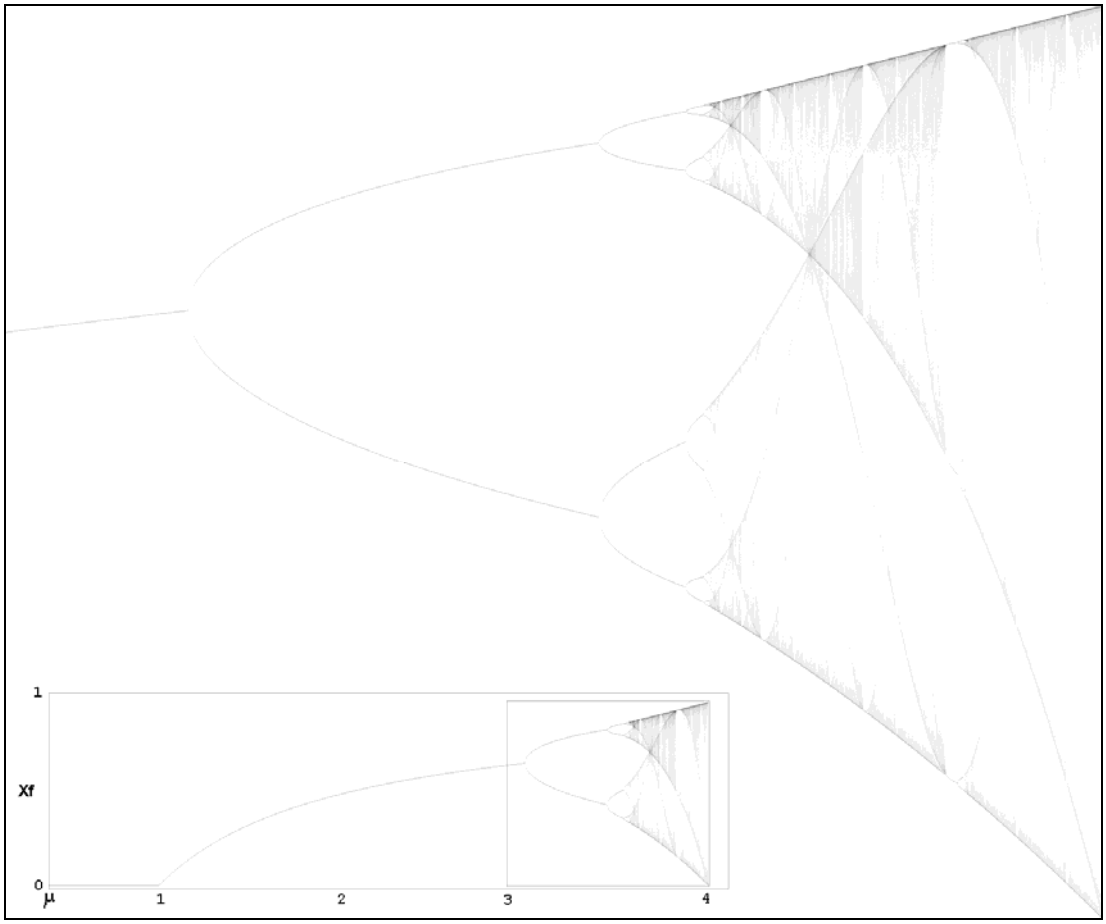

Figura 13. Diagrama de bifurcação para o mapa logístico, mostrando ampliada a região onde o sistema apresenta comportamento caótico.

Outro caso interessante de sistema com dinâmica caótica é o “mapa de tenda”. Ao contrario do “mapa logístico”, que descreve uma parábola, o “mapa de tenda” apresenta uma combinação de duas funções lineares. Seguem as equações do “mapa de tenda”, seu grau máximo de caos é atingido em $\mu = 2$.

$$X_{T+1} = \begin{cases} \mu.X_T & , se\left(0 < X < \frac{1}{2}\right) \\ \mu.\left(1 - X_T\right), & se\left(\frac{1}{2} < X < 1\right) \end{cases}$$

Qualquer equação pode, a princípio, apresentar comportamento complexo para algum intervalo de parâmetro e estado inicial pré-estabelecido. Porém, um estudo minucioso é necessário para comprovar que o sistema apresenta as características necessárias para defini-lo como caótico. A seguir, a equação do “mapa seno-circular”.

$$X_{T+1} = X_T + a + b.\sin(2.\pi.X_T)$$

A equação descreve um comportamento caótico dependente de dois parâmetros *a* e *b*, e uma condição inicial X_0; O parâmetro *a* pode assumir valores no intervalo {0, 1} e *b* valores maiores que zero, X_0 oscila entre os limites 0 e 1.

Tal comportamento pode ser aplicado para descrever comportamentos complexos, ou em casos onde se deseje introduzir um controle de "temperatura", ou seja, deseja-se que o sistema apresente um certo grau de aleatoriedade, porém com alguma previsibilidade. Equações não-lineares permitem ajustar o grau de agitação do sistema através de parâmetros, como μ no caso da equação do mapa logístico.

Uma boa introdução aos princípios da dinâmica não linear é "Understanding Nonlinear Dynamics" [KAPLAN, 1995], que trás uma apresentação clara e concisa sobre os conceitos matemáticos envolvidos e suas aplicações em áreas como simulação de fenômenos biológicos e VA. Para uma introdução matematicamente mais detalhada à dinâmica caótica, pode-se recorrer à "Chaotic Dynamics of Nonlinear Systems" [RASBAND, 1990].

3.2.4. Lógica Nebulosa

A lógica nebulosa foi introduzida por Lotfi Zadeh em 1973, como alternativa para expressar matematicamente a incerteza associada ao pensamento humano através do emprego de variáveis lingüísticas e funções de pertinência em lugar de ou em conjunto com variáveis numéricas. Em sua publicação [ZADEH, 1973] são apresentados métodos para descrever o comportamento de sistemas que são muito complexos ou mal definidos para permitir uma análise matemática comum.

As relações entre variáveis deste tipo se dão através de expressões condicionais nebulosas e algoritmos nebulosos. No uso de computadores, o processo de decisão visa dar um enfoque similar ao de sistemas humanísticos, atingindo, assim, um

desempenho comparado ao de um operador humano no processo. O autor defende que o enfoque convencional aplicando técnicas quantitativas de análise é inadequado para lidar com sistemas humanísticos. O “princípio da incompatibilidade” tem como essência a afirmação de que, conforme a complexidade do sistema aumenta, nossa habilidade para fazer alegações precisas e ainda significantes sobre o seu comportamento diminui, até que um limite seja atingido, a partir do qual precisão e significância (ou relevância) se tornam características quase mutuamente exclusivas. O enfoque descrito baseia-se na premissa de que o pensamento humano não se baseia em número, porém em rótulos de conjuntos nebulosos, onde a transição de pertinência ou não pertinência é mais gradual que abrupta. A Figura 14 apresenta um exemplo da aplicação de uma variável nebulosa e conjuntos nebulosos para determinação da distância.

Dentre as habilidades mais notáveis do pensamento humano, destaca-se a habilidade de sumarizar informação, mencionando que grande parte dos trabalhos realizada por humanos não requer um grande grau de precisão em sua execução. Essa capacidade de sumarização distingue a inteligência humana da inteligência computacional. Métodos tradicionais de análise são inadequados para lidar com sistemas humanísticos, pois falham em lidar com a imprecisão, ou “nebulosidade” do pensamento e comportamento humano.

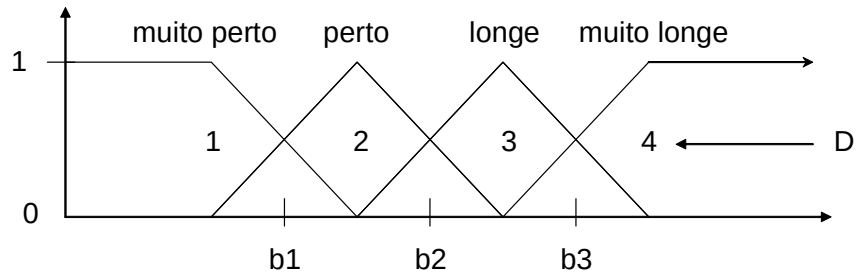

Figura 14. No exemplo, a Variável Nebulosa D (Distância) é formada por quatro Conjuntos Nebulosos, cujos rótulos definem a distância. Os pontos b1, b2 e b3 são conhecidos como Pontos de Intersecção (*Crossover points*) que ocorrem onde o grau de pertinência entre dois conjuntos consecutivos é igual (0,5/n, 0,5/n+1).

O conceito de controladores de lógica nebulosa foi posteriormente introduzido, motivado por trabalhos preliminares, como o de Zadeh. Controladores com Lógica

Nebulosa (*Fuzzy Logic Controllers* ou FLC) são conjuntos de regras de controle lingüístico, relacionadas pelos conceitos de implicação nebulosa e regras de inferência, provendo um algoritmo capaz de descrever uma estratégia de controle lingüístico baseado em conhecimento especializado, de forma similar ao processo de decisão humano. Métodos de aquisição de conhecimento baseado em processos descritos por um operador experiente, em estratégias de controle lingüístico, sem que seja necessário ter conhecimento sobre a dinâmica envolvida no processo, são apresentados em [LEE, 1990a], [LEE, 1990b], onde o autor descreve o uso de controladores baseados em lógica nebulosa para ajudar a resolver problemas comuns relacionados a processos mal definidos, ou demasiadamente complexos, que envolvem falta de dados quantitativos em relações de entrada e saída. O enfoque dado segue o espírito do texto fundamental de Zadeh, dando ênfase ao uso da linguagem natural do pensamento humano, que leva em consideração a natureza inexata do mundo real.

Um exemplo da aplicação de lógica nebulosa para simulação de agentes em experimentos de VA é apresentado no artigo "*Evolutionary Search for Optimization of Fuzzy Logic Controllers*" [NEVES, 2002] disponível on-line [ALIVE SITE].

3.2.5. Redes Neurais Artificiais

Com os avanços obtidos no estudo do cérebro durante o século passado, pôde-se compreender melhor como a informação é armazenada e utilizada nos sistemas nervosos biológicos, como o humano. A tecnologia de Redes Neurais artificiais aplica aproximações do modelo biológico a conjuntos de neurônios artificiais, simulados em computador, garantindo a sistemas computacionais algumas das características cognitivas observadas em organismos vivos dotados de sistema nervoso. Talvez o avanço mais significativo em eletro-neurofisiologia se deu graças a Hudgkin e Huxley, que modelaram em 1960 os processos biofísicos envolvidos na geração do potencial de ação do neurônio, trabalho que lhes valeu o prêmio Nobel de Fisiologia em 1963. A história do estudo do cérebro, que deu origem à disciplina conhecida como conexionismo, é apresentada em [KOVACS, 1997b]. Um gráfico

cronológico das pesquisas que levaram a criação da teoria do “cérebro digital” é representada na Figura 15.

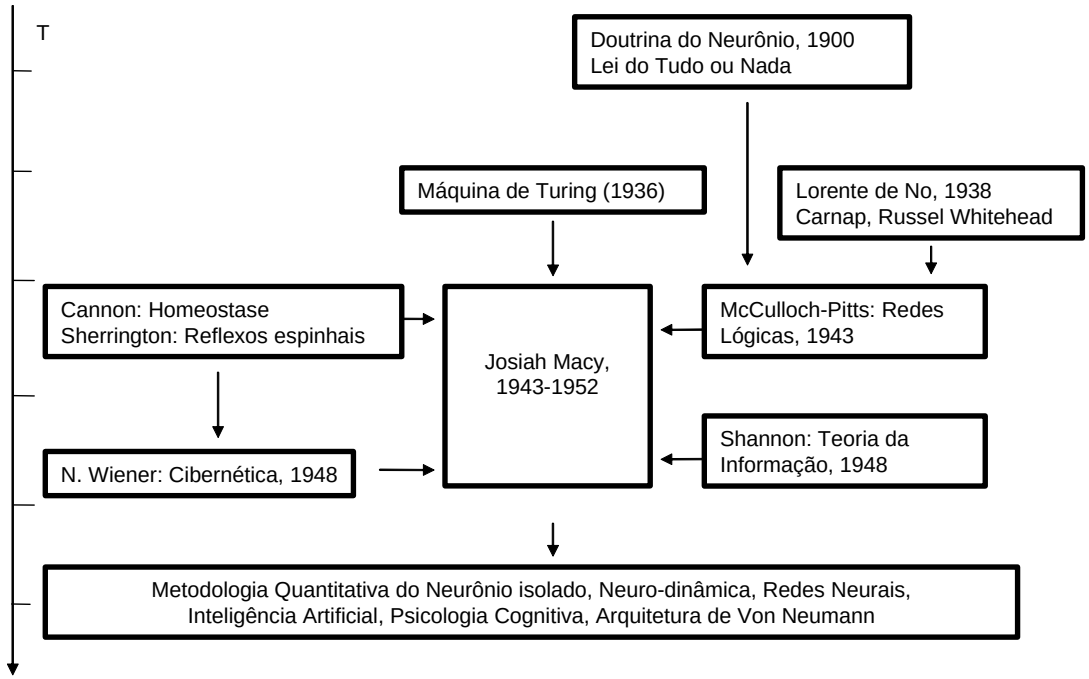

Figura 15. Raízes do cérebro digital.

Um neurônio artificial pode ser descrito como um circuito integrador com várias entradas, ponderadas por pesos individuais (*Wn*) e uma única saída, que é acionada mediante uma função de ativação pré-estabelecida. A função mais comumente empregada para tal fim é a sigmóide, que tem grande similaridade com o modelo biológico.

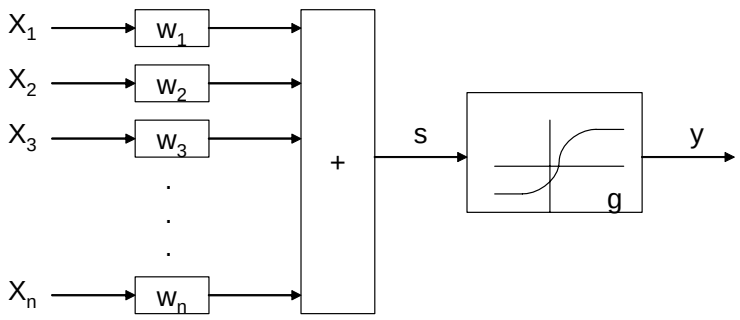

Figura 16. Modelo de blocos do neurônio artificial com função sigmóide.

De acordo com o esquema apresentado na Figura 16, um neurônio artificial tem uma fórmula matemática do tipo:

$$f(t) = g\left(\sum_i w_i(t) X_i(t) \right)$$

A saída é resultado da função de ativação *g(s)* aplicada sobre a somatória das entradas individuais multiplicadas pelo peso de cada conexão associado a ela. Toda a informação contida nos neurônios artificiais da rede é armazenada pelos valores das conexões *Wn*.

Para que a Rede Neural Artificial implementada desempenhe uma função específica, algoritmos de treinamento devem ser empregados para ajustar os valores de *Wn*. Existem modelos de algoritmo de treinamento dos mais variados que podem ser empregados para tal fim, entre eles modelos matemáticos teóricos, que podem ser supervisionados ou não supervisionados, modelos com ajuste por diferenças discretas utilizando cálculo numérico e um novo algoritmo baseado no modelo biológico, que se encontra em atual pesquisa.

A teoria de redes neurais, origens, tipos, aplicações e algoritmos de treinamento são apresentados de forma compreensiva em [HAYKIN, 1998]. Uma breve introdução à história e conceitos fundamentais da disciplina de Redes Neurais é apresentada de maneira sintática e didática em [KOVACS, 1997a], livro em português.

3.2.6. Algoritmos Genéticos e Sistemas Adaptativos

A versatilidade observada na adaptação de formas de vida ao seu ambiente, que na maioria das vezes leva estas formas de vida a atingir um resultado ótimo na adaptação ao meio, inspirou o uso dos métodos “naturais” para solução de problemas. Algoritmos genéticos aplicam conceitos de biologia e evolução na solução de problemas nas ciências e engenharia, aplicando conceitos de hereditariedade e técnicas como cruzamento e mutação.

Combinando critérios adaptativos, conceitos de inteligência computacional, multi-estratégias, entre outras técnicas, o potencial da busca evolucionária pode ser expandido, levando a algoritmos mais eficientes, que consomem menos recursos e levam a melhores soluções. Algoritmos de busca evolutiva estocástica visando capacidades de otimização requerem busca extensiva. Tais algoritmos apresentam alguns atributos desejáveis:

- Pouca, ou nenhuma informação adicional sobre o ambiente procurado, deve ser necessária *a priori* (ex. informação de gradiente).
- A eficiência da busca melhora de acordo com a população que amostra o espaço de soluções possíveis.
- Habilidade de evitar mínimos locais, re-amostrando o espaço aleatoriamente, evitando a convergência para um mínimo sub-ótimo.
- Habilidade de lidar com múltiplas dimensões, em problemas que envolvam grande número de variáveis.
- Generalidade sobre várias classes de problemas, onde a técnica apresente vantagens sobre outros algoritmos de otimização mais determinísticos.
- Provisão de múltiplas boas-soluções, visando identificar no espaço de soluções possíveis, as soluções de alta-performance para o dado problema.
- Habilidade de localizar a região de solução ótima.

Tecnologias nebulosas (*Fuzzy Logic, Fuzzy Controllers*), quando integradas à busca evolucionária, podem lidar melhor com as incertezas inerentes a projetos de alto-nível. O uso de agentes inteligentes como base populacional pode aumentar consideravelmente a inteligência computacional, a um custo computacional razoável, como será discutido posteriormente. "*Evolutionary Search for Optimization of Fuzzy Logic Controllers*" [NEVES, 2002] traz um exemplo de aplicação de Controladores com Lógica Nebulosa em um contexto multi-agentes, disponível on-line no sítio do projeto [ALIVE SITE].

Para uma introdução ao uso de algoritmos genéticos "*An Introduction to Genetic Algorithms*" [MITCHELL, 1997] traz os fundamentos e algumas aplicações básicas.

Técnicas avançadas e algoritmos para busca evolutiva são apresentados em "*Evolutionary Design by Computers*" [BENTLEY, 1999]. Para estudos mais aprofundados sobre algoritmos genéticos, "*Genetic Algorithms + Data Structures = Evolution Programs*" [MICHALEWIC, 1996] traz uma abordagem detalhada sobre estruturação das entradas e saídas e alguns algoritmos funcionais.

3.3. Codificação dos Organismos e do Ambiente Virtual

Aqui são apresentados alguns conceitos sobre engenharia de software, entre eles tópicos sobre como estruturar um programa em uma linguagem orientada a objetos, criando uma analogia natural com o problema abordado, como organizar e sincronizar processos que serão executados paralelamente utilizando-se da teoria de Sistemas Multi-Agentes e como efetuar cálculos vetoriais que envolvem intersecção e colisões de formas geométricas em espaços virtuais.

3.3.1. Paradigma de Programação Orientada a Objetos

Programação orientada a objetos (POO) é uma maneira de abordar a decomposição de um problema para desenvolver uma solução algorítmica. Os cinco conceitos principais que definem uma linguagem orientada a objetos são: **Objetos**, **Classes**, **Encapsulamento**, **Herança** e **Polimorfismo**.

Objetos são pacotes de programas que podem conter tanto atributos (dados e variáveis), quanto comportamento (métodos). São utilizados, geralmente, para representar objetos do mundo real, mas podem representar, a principio, qualquer estrutura de dados ou comportamento funcional relacionado a estas estruturas, como, por exemplo, matrizes ou pacotes IP.

Classes são protótipos que definem variáveis e métodos comuns a um conjunto de objetos. Objetos não existem sozinhos, mas são instâncias de uma particular classe. Objetos com uma mesma estrutura de dados (atributos) e mesmos comportamentos (métodos ou operações) são agrupados em uma determinada classe. Quando

definimos uma classe, definimos as propriedades de um conjunto de objetos que compartilham os mesmos atributos e métodos.

Encapsulamento é o princípio da caixa-preta, e diz respeito a esconder do usuário, ou cliente, os detalhes da implementação, o acesso aos dados e comportamentos é feito sem que seja necessário qualquer conhecimento do funcionamento interno da interface. Utilizando o exemplo do gravador, um programa cliente que utilize a classe conhece seus atributos ou dados (gravação) e seus comportamentos (*play*, *rec*, *rewind*), mas sem o conhecimento do mecanismo por trás de seu funcionamento. Toda classe tem duas partes: Interface e Implementação. A interface é a parte que fica visível para acesso e utilização; a implementação é o resto do código, que realiza as operações necessárias para prover o retorno baseado nos parâmetros passados pelo cliente. Desta maneira, criam-se níveis de complexidade, onde nenhuma tarefa de um sistema complexo depende de detalhes internos de outra parte.

Encapsulamento é também uma técnica para minimizar a interdependência entre módulos, definindo uma simples interface externa e permitindo que o código interno seja modificado sem afetá-la. Desde que novas implementações mantenham a interface externa compatível, o código de um objeto pode ser modificado sem afetar a aplicação que o utiliza, melhorando a performance, corrigindo falhas, consolidando o código ou aumentando sua portabilidade.

Herança determina que objetos pertencentes a um determinado grupo compartilhem as propriedades deste grupo. Uma classe pode estender (ou especializar) uma classe superior, herdando os atributos e comportamentos da superclasse, implementando adições ou restrições ao seu funcionamento. Herança é um método eficiente e robusto de reutilização de código, além de prover um mecanismo natural e poderoso de estruturar e organizar programas em um aplicativo.

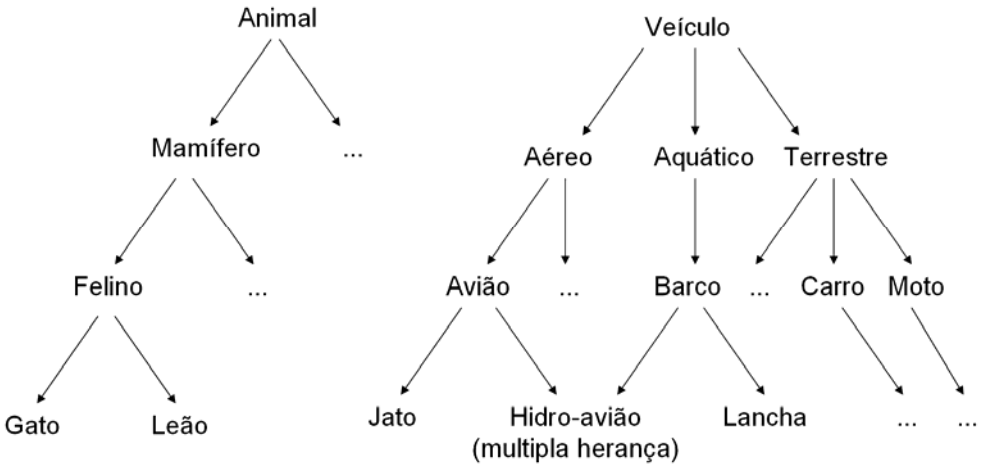

Figura 17. Exemplos de herança em classes diferentes.

O último dos conceitos de POO, denominado **polimorfismo**, é um pouco ambíguo e também diz respeito à ambigüidade. Existem definições diferentes em diferentes literaturas, porém, o que parece ser um consenso é que o polimorfismo garante a um usuário efetuar uma mesma chamada em diferentes objetos de diversas classes e ainda assim ter seu desejo realizado. Polimorfismo prevê a habilidade de enviar uma mesma mensagem para uma larga gama de tipos de objetos, ou seja, requisitar uma mesma operação a objetos ou classes diferentes. Polimorfismo também diz respeito ao tipo de parâmetro passado nas mensagens, associando, a um mesmo método, várias combinações possíveis de parâmetros.

POO vê um programa como uma coleção de objetos (ou agentes) conectados, em que cada objeto é responsável por uma tarefa específica. Um objeto é um encapsulamento de estados, dados e comportamentos ou operações. O comportamento de um objeto é ditado pela classe a que pertence. Todo objeto é uma instância de alguma classe. Todas as instâncias de uma mesma classe vão se comportar de uma maneira similar, em resposta a um estímulo (ou chamada) similar.

Um objeto exibirá seu comportamento evocando um método em resposta a uma mensagem recebida. A interpretação da mensagem é decidida pelo objeto e pode ser diferente entre classes distintas.

Classes podem ser organizadas em árvores de hierarquias hereditárias. Os dados e comportamentos associados com classes de maior hierarquia na árvore podem ser acessados por classes hierarquicamente inferiores.

Exemplificando com um experimento de VA, determinamos uma superclasse “ambiente” como sendo responsável pelas operações e atributos comuns a qualquer tipo de ambiente construído. A superclasse “agente”, similarmente, contém atributos e operações comuns à manipulação de agentes no domínio do ambiente virtual. Para realizar um experimento em um meio aquoso, criamos uma classe “filha” que estende a superclasse ambiente, e adiciona o comportamento específico dos objetos em meio à dinâmica ambiental representada. Da mesma forma, os agentes que habitam o ambiente específico são objetos de classes “filhas” da superclasse agente, que representam as peculiaridades dos agentes, definidas por seus códigos.

Num contexto de POO, definimos os seguintes conceitos:

Agentes: Um objeto que provê meios para completar uma tarefa é visto como um agente apropriado para esta tarefa. Se alguém deseja viajar, procura um agente de viagens. O agente de viagens é um agente apropriado para a tarefa de agendar e organizar viagens. Se o desejo é comprar um carro, provavelmente o agente de viagens será pouco apropriado. O agente apropriado para se comprar um carro será um vendedor de automóveis.

Mensagens: Uma ação é iniciada em POO pela transmissão de uma mensagem para um agente ou objeto responsável pela ação. A mensagem codifica a requisição para dada ação e é acompanhada por toda informação (argumentos) necessária para realizar a requisição.

Métodos: Métodos são os meios pelos quais os agentes executam as requisições contidas na mensagem, e envolvem a responsabilidade implicada na aceitação de tal mensagem. Métodos são evocados pelo agente, em resposta às mensagens recebidas.

O funcionamento particular de cada método não está disponível a nenhum outro objeto, senão àquele que contém o método evocado.

Responsabilidades: Um conceito fundamental em POO é descrever o comportamento em termos de responsabilidades. Um requisito por uma ação indica apenas o resultado esperado. O agente é livre para empregar qualquer técnica (ou método) que o leve a obter o objetivo designado e não interfira com outros processos.

Se desejarmos escrever uma classe em POO para cadastrar pessoas em uma academia, por exemplo, podemos ter uma classe como segue:

```
class pessoa {
  String Nome;          // e.g. "Rogério Neves"
  String RG;            // e.g. "12.234.432-8"
  double altura;        // em metros
  double peso;          // em Kg
}
```

As variáveis (nome, RG, altura e peso) são chamadas variáveis membro, de instância ou de campo (*field variables*), e podem ser diferentes em cada instância ou objeto pertencente a esta classe. Os campos dizem o que é uma classe, e descrevem suas propriedades. Se uma variável é declarada como estática, seu valor é o mesmo para todas as instâncias da classe, o que geralmente é associado a uma memória compartilhada por todos os objetos deste tipo.

Os comportamentos ou métodos para a classe especificada poderiam ser:

```
class pessoa {
  public void setNome(String nome) {
    ...  }
  public void setRG(String RG){
    ...  }
  public void setAltura(double altura){
    ...  }
  public void setPeso(double peso){
    ...  }}
```

As declarações dos métodos são precedidas pelo tipo de acesso e tipo de retorno devolvido. Os tipos de acesso possíveis são:

Público (*public*): O método pode ser evocado por código em qualquer instância, pertencente a qualquer classe, em qualquer pacote.

Privado (*private*): O método pode ser evocado apenas pelo próprio objeto da classe que contém o método.

Protegido (*protected*): O método pode ser apenas evocado por instâncias da mesma classe, ou que a estendam, por herança, desde que pertençam ao mesmo pacote.

Um método pode ser modificado em uma classe que o herda, técnica conhecida como substituição (*overriding*). A seqüência natural de execução de uma mensagem recebida é: procurar um método local; caso não o encontre, enviar a mensagem para a superclasse hierarquicamente superior, sucessivamente, até que seja executada ou falhe em todas as instâncias, retornando uma exceção.

Métodos definidos como abstratos (*abstract*) dentro de uma classe, determinam que tal classe é abstrata, significando que não existe código para o método na implementação, mas as classes que a estendem deverão prover o código para o método abstrato. Uma classe abstrata não pode ser diretamente instanciada; apenas as classes “filhas” que implementarem os métodos definidos como abstratos poderão ser instanciadas.

Métodos definidos como finais (*final*) não podem ser substituídos em classes hierarquicamente inferiores.

Os métodos definidos como estáticos (*static*) não serão instanciados, sendo compartilhados (escopo global) por todos os objetos da classe. Apenas variáveis estáticas, ou parâmetros, poderão ser operados dentro do escopo destes métodos.

3.3.2. Sistemas Multi-Agentes

Numa definição geral, um agente seria qualquer entidade existente que, em um ambiente propício, age sobre ele de alguma maneira determinada. Quando falamos de agentes infecciosos, nos referimos a entidades biológicas que quando presentes no corpo (ambiente propício) se proliferam causando danos à saúde. Quando falamos de agentes poluentes, nos referimos a substâncias tóxicas que causam danos ao meio ambiente, na atmosfera, nos rios ou nos oceanos.

Do ponto de vista profissional, o agente é aquele que representa os desejos e vontades do cliente, e, com base nos objetivos estipulados, tenta realizar a tarefa que lhe foi atribuída da melhor maneira possível, com base no seu conhecimento. Um agente de viagens tenta realizar o desejo do cliente no planejamento de viagens, determinando melhores rotas, preços e horários, visando a satisfação dos objetivos impostos com base no seu conhecimento sobre empresas, serviços e tarifas relacionados à sua especialidade. Se o objetivo é arrumar um emprego, no entanto, o agente de viagens é pouco indicado para o trabalho. Portanto, um agente possui um "ambiente" ou "nicho" de atuação, fora do qual não é útil ou funcional.

Da mesma maneira, um agente eletrônico é uma entidade computacional (de software), que tem por objetivo desempenhar alguma tarefa para a qual foi designado. Para tanto, pode lançar mão de uma sorte de técnicas e métodos de forma flexível e autônoma, visando unicamente a satisfação dos objetivos impostos, da melhor maneira possível, segundo os critérios estabelecidos e sua habilidade própria.

Definir o "agente" não é tão trivial quanto possa parecer, porque a palavra pode ter alguns significados diferentes: em Inteligência Artificial, pode ser usado para referir um processo ou tarefa computacional. Agente é uma entidade computacional ativa a quem um humano delega uma ou várias funções. O agente tem a percepção do ambiente em que está imerso e também a capacidade de fazer parte de uma comunidade de agentes com diferentes capacidades e objetivos. Seguem algumas definições formais do termo agente, apresentadas em literaturas envolvendo o estudo de SMA [FRANKLIN, 1996].

O agente AIMA [RUSSEL, 1995]: "*Um agente é qualquer coisa que pode ser visto como percebendo seu ambiente através de sensores e agindo sobre este ambiente através de atuadores*". Definindo o ambiente como algo que provê uma entrada e recebe uma saída, considerando receber como "sentir" e produzir saída como "agir", todo programa pode ser considerado um agente.

O agente MAES [MAES, 1995]: "*Agentes autônomos são sistemas computacionais que habitam algum ambiente de dinâmica complexa, sentem e agem autonomamente neste ambiente e, fazendo isso, realizam uma série de objetivos ou tarefas para os quais foram designados*". Nesta definição, além das noções "agir" e "sentir", o autor vincula a noção de "autonomia" às propriedades de um agente.

O agente de Hayes-Roth [HAYES, 1995]: "*Agentes inteligentes continuamente realizam as funções: percepção das condições dinâmicas do ambiente; ação que afeta tais condições no ambiente; e raciocínio para interpretar percepções, resolver problemas, lançar inferências, e determinar as ações*". A definição inclui, além das noções de "agir" e "sentir", a noção de "raciocínio".

O agente Wooldridge-Jennings [WOOLDRIDGE, 1995]: "*… um sistema baseado em hardware ou (mais comumente) software que apresenta as seguintes propriedades*":

- Autonomia: "*agentes operam sem intervenção direta de humanos ou outros, e tem algum controle sobre suas ações e estados internos*";
- Habilidade social: "*agentes interagem com outros agentes (e possivelmente humanos) através de algum tipo de linguagem de comunicação de agentes*";
- Reatividade: "*agentes percebem seu ambiente, (que pode ser um mundo físico, um usuário através de uma interface gráfica, uma coleção de outros agentes, Internet, ou uma combinação destes) e respondem de modo temporal a mudanças que ocorram nele*";

- Pro-atividade: "*agentes não simplesmente agem em resposta ao seu ambiente, são capazes de exibir comportamento dirigido ao objetivo, tomando iniciativa*".

As definições levam a dois usos distintos da palavra agente:

1. Aquele que age ou pode agir;
2. Aquele que age em lugar de outro com permissão.

Finalmente, a definição de agentes de VA [FRANKLIN, 1997] que habitam ambientes virtuais como a memória ou tela do computador: "*Um agente autônomo é um sistema situado e parte de um ambiente que sente este ambiente e age nele, no tempo, em busca de sua própria agenda (ou objetivos), o que afeta, no futuro, o que ele sente*".

As várias definições apresentadas acima do que é ser um agente ressaltam de forma menos restritiva algumas das propriedades de um agente eletrônico. A Tabela 3 apresenta algumas propriedades, não mutuamente necessárias, porém suficientes, que podem ser úteis na classificação de agentes por tipo.

Tabela 3. Propriedades desejáveis de um agente.

Propriedade	**Nome alternativo**	**Significado**
Reatividade	Sentimento e ação	Responde temporalmente a mudanças no ambiente
Pró-atividade	Propósito, objetividade	Não "responde" simplesmente aos estímulos ambientais, mas persegue um propósito
Sociabilidade	Comunicabilidade	Cooperação com outros agentes, incluindo o usuário
Mobilidade		Habilidade de mover-se no ambiente ou entre ambientes (ou máquinas) distintos
Continuidade		Executa ininterruptamente
Autonomia		Exerce controle sobre suas próprias ações
Cognição	Aprendizado, adaptação	Muda sua atitude, baseado na experiência anterior
Flexibilidade		Ações não prescritas
Crença		Personalidade, estado emocional

Sistemas Multi-Agentes, ou simplesmente SMA, são sistemas que implementam um conjunto de agentes eletrônicos para desempenhar as tarefas estabelecidas pelo usuário. Em um Sistema Multi-Agente, todos os agentes estão interessados em solucionar o problema em consideração, podendo cooperar na tentativa de obter a melhor solução para o determinado problema.

Os Sistemas Multi-Agentes permitem, também, modelar o comportamento de um conjunto de entidades (inteligentes ou programáticas) organizadas de acordo com leis sociais. Estas entidades dispõem de uma certa autonomia e estão imersas em um ambiente com o qual necessitam interagir e do qual devem possuir uma representação parcial, assim como meios de percepção e comunicação.

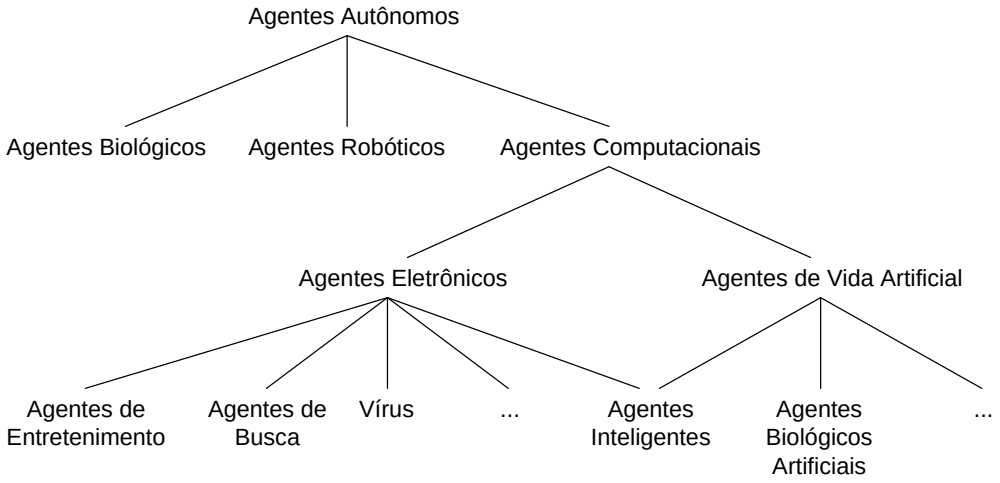

Figura 18. Árvore de ramificações dos tipos de agentes.

Os vários tipos de agentes eletrônicos são geralmente implementados em classes, embora não seja obrigatório o uso de linguagens orientadas a objetos para implementar SMA [SMITH, 1994]. O paradigma de orientação a objetos, no entanto, torna mais simples e natural a implementação de SMA. A Figura 18 mostra uma possível hierarquia na classificação de agentes. Agentes são construídos e operam em ambientes. Esses ambientes impõem restrições sobre o comportamento dos agentes e providenciam serviços e facilidades que podem ser usados pelos agentes. Um número significativo de arquiteturas de agentes tem sido proposto. Estas são tipicamente estruturadas em camadas, tendo no topo a camada de raciocínio. Um programa orientado a objetos, que utiliza objetos do tipo agente para desempenhar

funções em um ambiente comum, é denominado Programa Orientado a Agentes [SHLOAM, 1998].

Um experimento de VA Orientado a Agentes, por exemplo, utiliza agentes para simular os organismos existentes no ambiente virtual que interagem entre si e com o ambiente, assim como para representar características de entidades, outros agentes ou objetos, através de atributos observáveis (atores), como será apresentado no próximo capítulo.

Para detalhes sobre os paradigmas da implementação de experimentos orientados a agentes e a teoria envolvida no estudo de agentes eletrônicos e SMA, o leitor pode recorrer aos tutoriais [FONER SITE] disponíveis gratuitamente na Internet.

3.3.3. A Visão do Agente do Universo Virtual

Em um ambiente virtual, geralmente é preciso fazer testes para determinar se um objeto, ou agente, está no campo de visão de um determinado agente. Para tanto, algoritmos de cálculo vetorial são empregados para avaliar se os objetos são interceptados pela área ou volume representados pela forma geométrica, que representa o campo de visão do agente avaliado. Em um espaço em três dimensões, a forma geométrica em questão pode ser uma esfera, no caso de uma visão radial ou um cone, no caso de uma visão direcional, ou ainda qualquer forma geométrica simples ou composta.

Para exemplificar o conceito, a interceptação de um objeto pela área de visão de um agente em duas dimensões é apresentado na Figura 19, onde a visão radial é representada pelo círculo e a visão direcional é representada pelo setor angular.

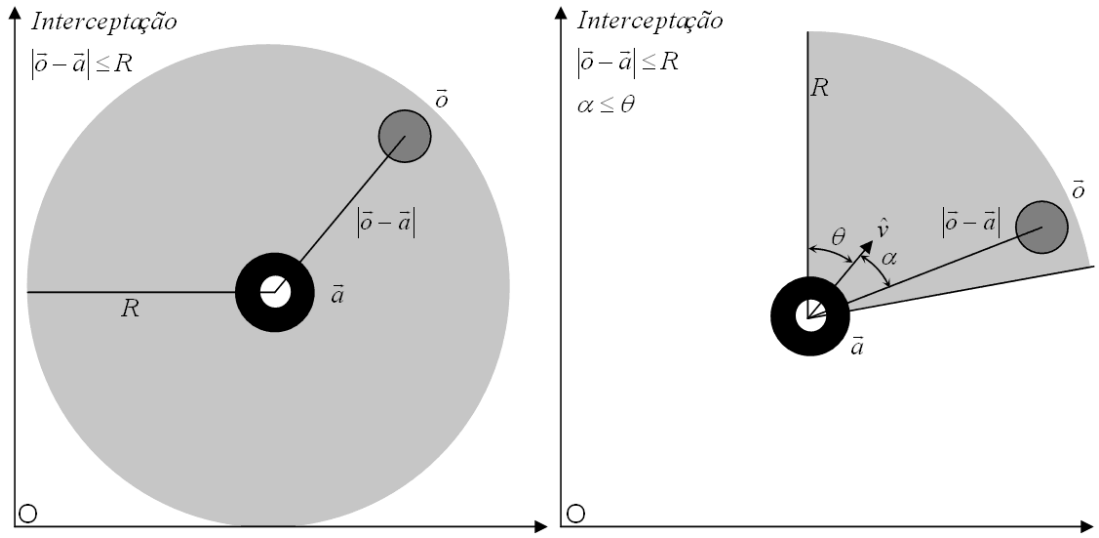

Figura 19. Exemplos de áreas de interseção em duas dimensões.

No caso esférico, para determinar se o objeto esta dentro do campo de visão do agente, basta avaliar se o comprimento do vetor calculado pela diferença entre os vetores representando a posição do agente $\vec{a} = (x_1, y_1, z_1)$, e do objeto $\vec{o} = (x_2, y_2, z_2)$ é igual ou inferior ao raio máximo do circulo ou esfera.

$$\left\|\vec{a} - \vec{o}\right\| \le R$$

Se o ângulo de sensibilidade é limitado por um cone, além da distância máxima, deve ser avaliado, também, se o ângulo entre o vetor que representa a orientação da visão do agente e o ângulo do vetor da diferença entre o agente e o objeto não excede o ângulo que define o cone.

$$angulo\left(\vec{d}, \vec{a} - \vec{o}\right) \le \theta$$

O calculo do ângulo acima pode ser feito utilizando-se propriedades do produto escalar e vetorial, embora a maioria dos pacotes de programação para cálculo vetorial já apresente funções ou métodos para se obter o ângulo entre dois vetores.

$$sen(\theta) = \frac{|u \times v|}{|u|.|v|} = \left|\frac{u}{|u|} \times \frac{v}{|v|}\right|$$

$$\theta = sen^{-1}\left(\left|\hat{u} \times \hat{v}\right|\right)$$

Onde os vetores unitários $\hat{u}$ e $\hat{v}$ são os vetores normalizados de $\vec{u}$ e $\vec{v}$.

Os testes interseção entre objetos com formatos mais complexos podem ser decompostos em testes envolvendo formas geométricas mais simples, ou utilizar algoritmos de interceptação utilizados em áreas de computação gráfica, que consistem de algoritmos que deslocam objetos de teste pela cena varrendo a área de teste.

3.3.4. Detectando Colisões

Os cálculos dinâmicos e testes responsáveis pela detecção de colisões podem ser implementados no código do ambiente, ou no código de cada agente. Este código deve verificar se as fronteiras definidas para os objetos presentes na cena foram violadas, aplicando as regras dinâmicas vigentes no caso de ter havido colisões.

No caso de tal algoritmo ser implementado junto ao código do agente, este deve varrer a lista de objetos contidos na cena, verificando se suas posições e formas interceptam sua própria forma. Para tanto, se estabelecem fronteiras (*boundings*) de contato, que dispara o alerta de contato quando outros objetos penetram seu perímetro.

Para um algoritmo executando no ambiente, uma tabela pode ser mantida contendo as posições e fronteiras de cada agente, informando as instâncias quando uma colisão ocorrer.

O próprio Java3D implementa algoritmos para detecção de contatos por fronteiras, porém o uso do código nativo impõe algumas limitações, como a de detectar apenas uma colisão, com um único objeto por vez. No caso de vários objetos penetrando as fronteiras de um agente, apenas a primeira colisão seria detectada, e até que esta terminasse, nenhuma outra seria reportada.

Alternativamente, o algoritmo para detectar colisões pode ser implementado de forma semelhante ao descrito no tópico anterior, determinando objetos geométricos correspondentes aos formatos dos agentes presentes na cena e avaliando se suas fronteiras se interceptam. No caso de objetos esféricos, ou que podem ter suas fronteiras representadas por objetos esféricos, a colisão ocorre quando:

$$\left|\vec{p}_2 - \vec{p}_1\right| \leq R_2 + R_1$$

onde R_1 e $\vec{p}_1$ são o raio e a posição do objeto 1 e R_2 e $\vec{p}_2$ são o raio e posição do objeto 2.

O teste acima é de simples implementação e computacionalmente barato. Também é possível se representar uma fronteira de um objeto mais complexo por um conjunto de testes simples, como o apresentado. Para tanto, define-se um conjunto de pontos e raios que produzam esferas que cubram todo (ou quase) o volume de detecção. A Figura 20 mostra um exemplo de detecção de colisão, representando a fronteira de uma forma complexa por simples volumes esféricos.

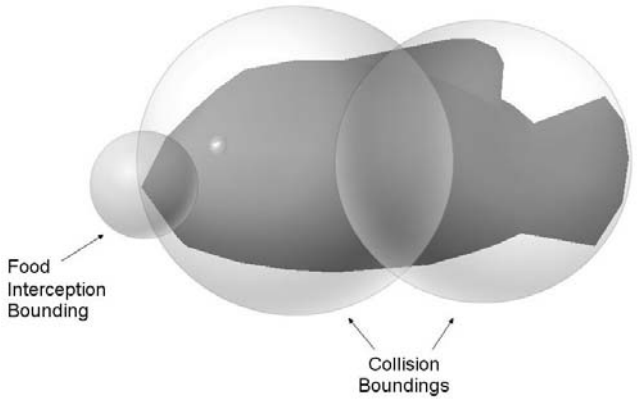

Figura 20. Exemplo de teste de interceptação e colisão de fronteiras de um objeto complexo por um conjunto de testes utilizando volumes esféricos.

A performance é um ponto crucial, considerando que cada agente na simulação deverá realizar todos os testes de interceptação em cada ciclo de execução; a eficiência do algoritmo de teste determina de forma crítica o tempo gasto na execução de cada ciclo.

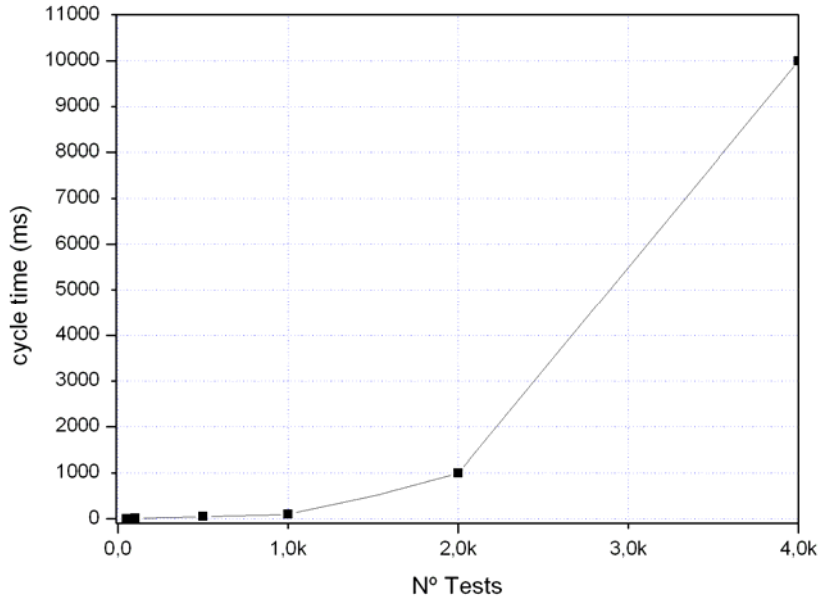

Figura 21. Gráfico do tempo gasto em cada ciclo versus o número de testes efetuados.

A ordem de grandeza do tempo gasto em cada ciclo pode ser extrapolada pela formula:

$$T_{total} = N_{agentes} \times \left(T_{testes} + T_{calculos}\right) + T_{ambiente}$$

O tempo em que cada ciclo executa, então, dependerá do número de agentes em execução, da quantidade de operações, cálculos e testes realizados por cada agente e da quantidade de operações, cálculos e testes desempenhadas pelo código do ambiente. A performance resultante pode variar significativamente, dependendo de onde o algoritmo de intersecção é codificado, no ambiente ou em cada agente.

3.4. Visualização do Ambiente Virtual

Definidos os parâmetros da simulação, resta determinar a técnica ou tecnologia que será empregada na visualização dos dados do experimento. Experimentos em vida artificial, geralmente empregam visualizações rudimentares, ou saídas em texto para representar o ambiente virtual. Alguns experimentos, porém, visam ressaltar a

semelhança entre os eventos ocorridos na simulação e os observados no mundo real. Para tanto, o uso de tecnologias de ponta é necessário para se obter o resultado visual desejado. Algumas características de cada tipo de representação são apresentadas abaixo.

3.4.1. Representação em Duas Dimensões

A exibição bi-dimensional é computacionalmente barata, se mostrando altamente eficiente para representação de fenômenos dinâmicos que acontecem sobre um plano, ou com dinâmica que envolve apenas dois graus de liberdade, ou menos.

Um caso específico que ilustra a aplicação de uma visualização gráfica 2D, esta na visualização do programa do projeto FUZZY-BOT [NEVES, 2002], cujo objetivo é treinar evolutivamente robôs virtuais controlados por lógica nebulosa para uma tarefa específica: coletar recursos. O ambiente virtual se resume a uma arena de dimensões limitadas onde os robôs se locomovem em uma superfície plana, na qual os robôs e recursos são representados por círculos cheios que representam suas fronteiras. No contexto da simulação, a visualização consome muito pouco dos recursos computacionais exigidos.

O mapeamento dos atributos de agentes e do ambiente podem ser feitos visualmente através da coloração dos objetos, etiquetas que flutuam sobre eles, ou através da interação, utilizando-se dispositivos como o Mouse para obter informações sobre os agentes em execução (clicando sobre os atores). O uso de tabelas auxiliares é largamente empregado, embora seja pouco prático para sistemas com dinâmica rápida ou grande número de agentes.

Embora seja barata e de fácil implementação, a representação em 2D é visualmente pobre e pode tornar-se desinteressante em experimentos onde se deseja ressaltar a analogia entre o ambiente virtual e o mundo real.

3.4.2. Visualização em Três Dimensões

A visualização em três dimensões apresenta um custo computacional mais elevado, podendo exigir, dependendo da complexidade do universo virtual, a aplicação em computadores modernos e o uso de caros dispositivos aceleradores gráficos para atingir os resultados desejados. Porém, apesar do custo elevado, a visualização em 3D apresenta uma qualidade visual superior e representa uma maior riqueza de informação, permitindo mapear atributos não só através da coloração, mas, também de formas, de texturas, ou símbolos dispostos sobre estas texturas. A transparência, também, pode representar informação, ou permitir que sejam observados eventos que ocorrem no interior das células ou agentes.

A visualização 3D cria ainda uma analogia direta entre eventos ocorridos em um ambiente virtual e fenômenos observados no mundo real.

3.4.3. Visualização do Ambiente em Realidade Virtual

Os Sistemas de Realidade Virtual são hoje a forma mais avançada de interface entre usuário e computador existente. A realidade virtual (ou simplesmente RV) é uma forma das pessoas visualizarem, manipularem e interagirem com computadores e dados extremamente complexos utilizando, para tanto, seus canais multi-sensoriais, i.e., seus sentidos, particularmente visão, audição e os movimentos naturais tridimensionais do corpo. A realidade virtual também pode ser considerada como a junção de três idéias básicas: imersão, interação e envolvimento [MORIE, 1994]. Isoladamente, essas idéias não são exclusivas de realidade virtual, mas, aqui, elas coexistem.

A visualização em RV, no entanto, requer computadores de alto desempenho e excelente capacidade gráfica, além de dispositivos não convencionais para exibição e interatividade, como capacete de visualização e controle, luva, entre outros que permitem a exploração do ambiente e a manipulação natural dos objetos com o uso das mãos, por exemplo, para apontar e pegar, entre outras ações.

Para a elaboração de sistemas de visualização em realidade virtual, também é necessário ter algum domínio sobre tópicos como: dispositivos não convencionais de entrada e saída, sistemas paralelos e distribuídos, modelagem geométrica tridimensional, simulação e sistemas em tempo real, navegação, detecção de colisão, avaliação, impacto social, projeto de interfaces, orientação a objetos, aplicações simples, distribuídas e multi-tarefas em diversas áreas além de tópicos de redes, modelagem geométrica, modelagem física, etc.

Um sistema é considerado imersivo se projeta o usuário dentro da cena virtual. O efeito imersivo pode ser obtido com o uso de capacete de visualização (HMD), ou uma CAVERNA (CAVE), sistema baseado em uma sala com projeções nas paredes e no piso [CAVERNA SITE]. Além do fator visual, os dispositivos relacionados a outros sentidos também são importantes para a sensação de imersão, como o som posicional estéreo. A realidade virtual não imersiva baseia-se no uso de monitores com óculos 3D (*Shutter Glasses*).

De acordo com Cris Shaw [SHAW, 1993], existem cinco requisitos e propriedades que um sistema de realidade virtual deve satisfazer para ser utilizável e satisfatório:

1. Um sistema de RV deve gerar imagens estereoscópicas, animadas e suaves, visando manter a característica de imersão. Isto significa que a taxa de quadros por segundo não deve ser menor que 10;
2. Um sistema de RV deve reagir rapidamente às ações do usuário. A resposta da imagem não deve exceder 100ms;
3. Um sistema de RV deve fornecer suporte para distribuir uma aplicação em diversos processadores. Para aplicações distribuídas e complexas, a distribuição permite múltiplos usuários e a computação cooperativa;
4. Num sistema distribuído de RV, é necessário um mecanismo eficiente de sincronização e comunicação de dados. Dados compartilhados ou remotos só são viabilizados com uma comunicação eficiente, que assegure o tempo real do sistema;

5. É necessário algum mecanismo de avaliação de desempenho. Um sistema de desenvolvimento de RV deve ter métodos de monitoração do desempenho geral da aplicação para garantir o sucesso do conjunto e a execução em tempo real.

Do ponto de vista da engenharia de software, deve-se observar os quatro requisitos que seguem [SHAW, 1993]:

1. Garantir a portabilidade das aplicações. Normalmente, as aplicações de realidade virtual são fortemente ligadas com o ambiente de desenvolvimento. As aplicações deverão possibilitar a execução em diversas plataformas, exigindo no máximo uma re-compilação do código;
2. Suporte para uma larga gama de dispositivos de entrada e saída. Como a tecnologia de realidade virtual ainda está em evolução, o sistema deverá ter capacidade de empregar novos dispositivos vindouros;
3. Independência das aplicações com relação à localização física do usuário, bem como de seus dispositivos de entrada e saída. O sistema deverá ajustar-se a diferentes configurações de localização física do usuário (geometria da sala e situação dos rastreadores);
4. Flexibilidade do ambiente de desenvolvimento de aplicações. Muitas vezes a aplicação em RV é desenvolvida num ambiente e executada em outro. O sistema deve ter flexibilidade para permitir a utilização de ambientes de desenvolvimento diferentes, bem como a execução de testes com outros dispositivos, com o mínimo de alteração do código.

Alguns passos no desenvolvimento de aplicações em RV podem ser relacionados sucintamente:

- Definição da aplicação;
- Caracterização do sistema de imersão;
- Avaliação dos dispositivos de visualização;
- Estabelecimento das capacidades de rastreamento;
- Avaliação de outros dispositivos de entrada e saída;

- Avaliação do conjunto de recursos e suas capacidades;
- Seleção do sistema de desenvolvimento:
    - Criação e edição da geometria;
    - Criação e edição de texturas;
    - Requisitos de programação;
    - Caracterização da estereoscópia;
    - Modelagem do comportamento físico;
    - Suporte aos periféricos;
    - Requisitos do sistema;
    - Portabilidade;
    - Suporte de rede;
    - Suporte a distribuição.
- Seleção do hardware:
    - Quantidade e características de comunicação;
    - Características do acelerador gráfico;
    - Capacete de visualização (HMD);
    - CAVERNA (CAVE);
    - Monitor externo;
    - Óculos 3D (*Shutter Glasses*);
    - Navegadores 3D, rastreadores e apontadores;
    - Luvas e dispositivos de força;
    - Outros dispositivos especiais.

A realidade virtual vem revolucionando a forma de interação das pessoas com sistemas complexos tratados por computadores, propiciando maior desempenho e economizando custos [RV SITE]. Novas aplicações surgem a cada dia, como sistemas de modelagem e visualização de dados científicos, ou o laboratório virtual de Vida Artificial descrito no próximo capítulo. A implementação da plataforma descrita no próximo capítulo visa, entre outros objetivos, diminuir o número de requisitos necessários e conhecimentos envolvidos na implementação de um experimento que utiliza RV. Um tutorial que descreve os principais conceitos e um pouco da história da RV pode ser obtido on-line em [RV SITE].

4. IMPLEMENTAÇÃO DA PLATAFORMA

A especificação inicial do projeto visava empregar técnicas de visualização científica a experimentos de VA, fazendo uso das tecnologias de computação gráfica e realidade virtual disponíveis. A princípio, o projeto visava utilizar os recursos do Núcleo Realidade Virtual do Laboratório de Sistemas Integráveis da Escola Politécnica da USP, especificamente os equipamentos que compõem a CAVERNA digital, uma sala cúbica com cinco telas retro-projetadas cuja função é dar uma sensação de imersão total no ambiente virtual exibido. Sua operação é feita por seis computadores IBM-PC interconectados em um aglomerado (*cluster*), ou por uma estação Silicon Graphics.

A plataforma experimental proposta tinha então como objetivo disponibilizar uma interface de aplicação que simplificasse o processo de implementação de experimentos em um ambiente de desenvolvimento de fácil utilização, portável, com suporte a dispositivos gráficos variados em um contexto multi-agentes. Também se previa a utilização de sistemas multi-processados, como o do projeto SPADE, ou de arquiteturas distribuídas, como o cluster integrado à CAVERNA digital, para se obter ganhos de desempenho. À medida que era desenvolvida, funcionalidades foram adicionadas à plataforma.

Desde a primeira versão funcional do programa, foram adicionadas funcionalidades como uma interface de usuário, aprimoramento gráfico da visualização e a possibilidade de executar os experimentos em modo *Applet*, permitindo a visualização de experimentos pela Internet, em qualquer *browser* com suporte a Java e Java3D. A filosofia de código aberto permite que usuários ao redor do mundo contribuam para o aprimoramento da plataforma, continuando seu desenvolvimento de forma colaborativa. A seção 4.1 trata da especificação da plataforma implementada e das características propostas para tal ambiente de desenvolvimento.

Tratando-se de uma simulação de agentes em um ambiente físico, que envolve comunicação entre entidades e alguma independência entre as diversas instâncias, a

adoção do paradigma de POO permitiu a definição de uma arquitetura de software em uma linguagem natural, com direta analogia à estrutura observada no mundo real. Pelos motivos apresentados até agora, a escolha da linguagem Java™ da Sun Microsystems, disponibiliza a versatilidade e robustez necessárias ao projeto, além de apresentar vantagens adicionais, como disponibilidade de material e suporte de grupos de utilizadores através da Internet. Os motivos que levaram a escolha da linguagem Java, e do API Java3D para desenvolver o projeto são apresentados na seção 4.2, bem como uma breve introdução a sua utilização. Também são apresentados alguns conceitos fundamentais para utilização do API Java3D, como operações com vetores para manipulação de objetos.

A seção 4.3 descreve a arquitetura da plataforma, apresentando os diagramas de classe definidos para a especificação e explicando a utilização dos métodos e variáveis das classes base.

A seção 4.4 traz algumas considerações importantes sobre a implementação de experimentos com objetos que executam concorrentemente, como sincronização, criação, destruição e uso de semáforos.

A seção 4.5 explica como foi testada a plataforma, apresentando alguns experimentos simples que foram implementados com este fim.

4.1. Proposta do Projeto da Plataforma

Muitas vezes, a maior dificuldade encontrada na realização de experimentos em VA não está na definição teórica do problema, na especificação do experimento ou na modelagem comportamental dos seres virtuais, mas, sim, na implementação da simulação. Na fase de programação, é preciso lidar com uma série de peculiaridades da linguagem de programação escolhida e contornar uma série de limitações nela existentes.

Mesmo após a fase inicial de implementação, sequer os programadores mais experientes escapam da tarefa de busca e correção de erros de programação, que podem envolver desde simples erros de sintaxe até complexos erros de consistência, acesso ilegal e de violação de memória entre uma infinidade de modalidades. Alguns erros só se revelam muito tempo após o programa estar pronto e distribuído.

Neste sentido, é esperado que o pesquisador, disposto a desempenhar tal tarefa, possua um domínio razoável da linguagem de programação, pois caso contrário, a tarefa pode se tornar extremamente complexa e desgastante. A complexidade aumenta indefinidamente quando se deseja utilizar objetos distribuídos, ou que executam concorrentemente. Falhas na comunicação e sincronização de instâncias concorrentes, bem como violações de acesso a variáveis, dados e arquivos, são causa comum de problemas envolvendo programação concorrente.

O projeto propõe a criação de uma plataforma de experimentação em VA que facilite ao máximo a implementação de experimentos, fornecendo um mecanismo eficiente de visualização e ferramentas de comunicação e sincronização de instâncias de agentes, aplicando cálculo vetorial para a movimentação e orientação dos agentes.

Utilizando as superclasses, o usuário, interessado em implementar um experimento utilizando a plataforma, deve: escrever o código referente às regras específicas presentes no ambiente simulado; e o código referente aos agentes nele presentes e seus particulares comportamentos. Métodos de comunicação e sincronização de agentes são fornecidos, bem como ferramentas utilitárias com o propósito de tornar o processo de implementação o mais simples e rápido possível, exigindo assim o mínimo de conhecimento do usuário para o processo de implementação de experimentos.

O projeto visa aplicação em sistemas de visualização de realidade virtual e computação gráfica, podendo utilizar para exibição dos experimentos desde placas aceleradoras gráficas e monitores convencionais, até sistemas de realidade virtual, como *Shutter Glasses*, HMD (*Head Mounted Displays*) e de imersão total, como a

CAVERNA digital. Placas aceleradoras gráficas são recomendadas para diminuir a carga computacional sobre o processador referente ao processo de síntese das imagens (*rendering*).

Algumas das características propostas para o projeto são:

- Paradigma de programação orientada a objetos;
- Compatibilidade com diversas plataformas computacionais;
- Ambiente simulado tri-dimensional com dinâmica vetorial;
- Visualização em 3D (estéreo) com suporte a diversos dispositivos gráficos;
- Possibilidade de visualização em dispositivos de realidade virtual e em ambientes imersivos (CAVERNA);
- Possibilidade de execução concorrente em sistemas multi-proccessados, num contexto multi-agentes;
- Possibilidade de utilização de sistemas distribuídos em cluster, para otimização da performance na computação e apresentação gráfica;
- Possibilidade de execução em modo *Applet*, para visualização na Internet.

Para implementar muitas das características citadas, o projeto conta com o poder da linguagem Java™ e do API Java3D™, desenvolvidos pela Sun Microsystems. Os motivos que levaram à escolha da linguagem e do acessório de visualização são descritos na próxima seção.

4.2. Sobre a Linguagem Sun Java™ e o API Java3D™

Programação envolve lidar com complexidade, e problemas complexos exigem uma linguagem de programação versátil e robusta. A linguagem Java começou como um esforço da Sun Microsystems para criar uma linguagem de programação extensível, que criasse programas capazes de executar em vários tipos diferentes de dispositivos e arquiteturas. Graças a sua versatilidade, Java se tornou a linguagem principal para distribuição de material executável pela Internet, dando aos usuários retorno

automático em páginas inteligentes, que executam no lado de cliente (*client-based application*) ou do servidor (*servlets*).

A linguagem Java estendeu e facilitou a maioria das tarefas complexas compreendidas no universo da computação, entre as quais: programação em rede; programação distribuída e multitarefa; programação multi-plataforma; mudanças dinâmicas de código; gerenciamento de segurança [ECKEL, 2002]. Também obteve grande repercussão por se tratar de uma ferramenta livre, seguindo a filosofia do código aberto, deixando pouco mistério a respeito de seu funcionamento interno e de suas capacidades.

Algumas das características que levaram a escolha da linguagem Java são:

- Linguagem orientada a objetos;
- Suporte à interação por console ou interface visual;
- Arquitetura concorrente com o uso de "*threads*";
- Compatibilidade com diversas plataformas sem re-compilação de código;
- Execução em modo aplicativo ou *Applet* para execução na Internet;
- Distribuição gratuita;
- Material de suporte disponível on-line.

A disponibilidade de material de qualidade e suporte on-line em um fórum de usuários estão entre as principais características da linguagem. Para uma introdução detalhada à linguagem e seus paradigmas, o livro "*Thinking in Java*" pode ser obtido on-line ou nas livrarias [ECKEL, 2002]. O livro "Java, como programar" [DEITEL, 2002] é utilizado na maioria dos cursos de graduação de introdução à linguagem Java. Como referência para usuários iniciados, a documentação da linguagem Java disponível no sítio da Sun na Internet é a mais completa existente [SUN SITE].

O API Java3D é uma extensão da linguagem Java cuja finalidade é prover acesso a dispositivos gráficos dependentes de plataforma, sem que a execução dos programas comprometa sua portabilidade. O API (*Application Programming Interface*)

incorpora métodos de acesso de alto-nível para a modelagem da cena e lida com os acessos às arquiteturas de baixo nível como OpenGL e DirectX.

O API Java3D utiliza um modelo de visualização flexível e orientado a objetos para construção da cena, onde os objetos são conectados hierarquicamente em um grafo descritivo da cena representada, podendo gerar saídas para uma série de dispositivos, provendo acesso às mais modernas tecnologias de hardware e capacidades gráficas disponíveis, tirando proveito de aceleradores gráficos para melhorar a performance. Para uma experiência imersiva, o mecanismo de entrada aceita uma série de dispositivos, desde simples *mouses* até apontadores (*wands*) e luvas (*gloves*).

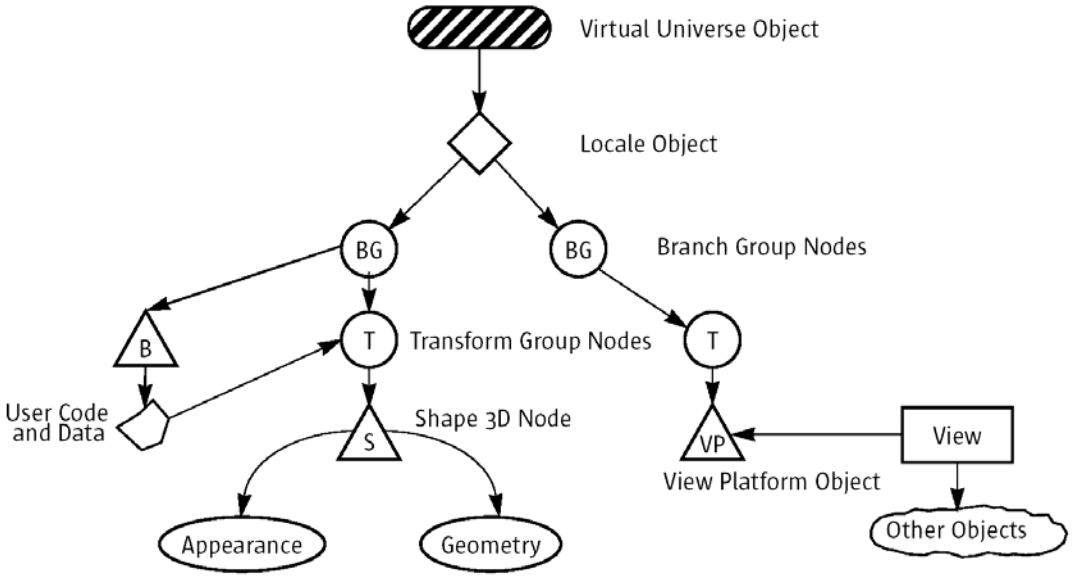

Figura 22. Exemplo de grafo de cena no API Java3D.

Não é requerido ao usuário da plataforma possuir conhecimento detalhado sobre a operação do API Java3D a não ser que deseje construir personagens com morfologia dinâmica ou que utilizem algoritmos para modificar a forma do ator representado em tempo de execução. Uma boa referência para o uso de Java3D é "*Getting Started with the Java3D™ API*", disponível gratuitamente no sítio da Sun [SUN SITE], onde também estão disponíveis a especificação do API e os manuais oficiais.

Para utilizar a plataforma, o usuário deverá escrever algum código, e, portanto, deverá possuir algum conhecimento da linguagem Java. No Apêndice C são apresentadas características essenciais da linguagem Java, necessárias e suficientes para o entendimento dos segmentos de código que são apresentados nas próximas seções. Apresenta, também, uma breve explicação sobre a utilização do API Java3D para manipulação dos objetos no universo virtual.

4.3. Arquitetura do Simulador

A plataforma disponibilizada apresenta sete classes principais e uma classe auxiliar, sendo três delas chamadas de Superclasses ou classes base para o desenvolvimento de experimentos. Os experimentos desenvolvidos deverão estender as classes base e utilizar as variáveis e métodos públicos disponibilizados para construir o comportamento dos agentes e a física ambiental do experimento implementado. A Figura 23 apresenta as classes fundamentais e suas dependências.

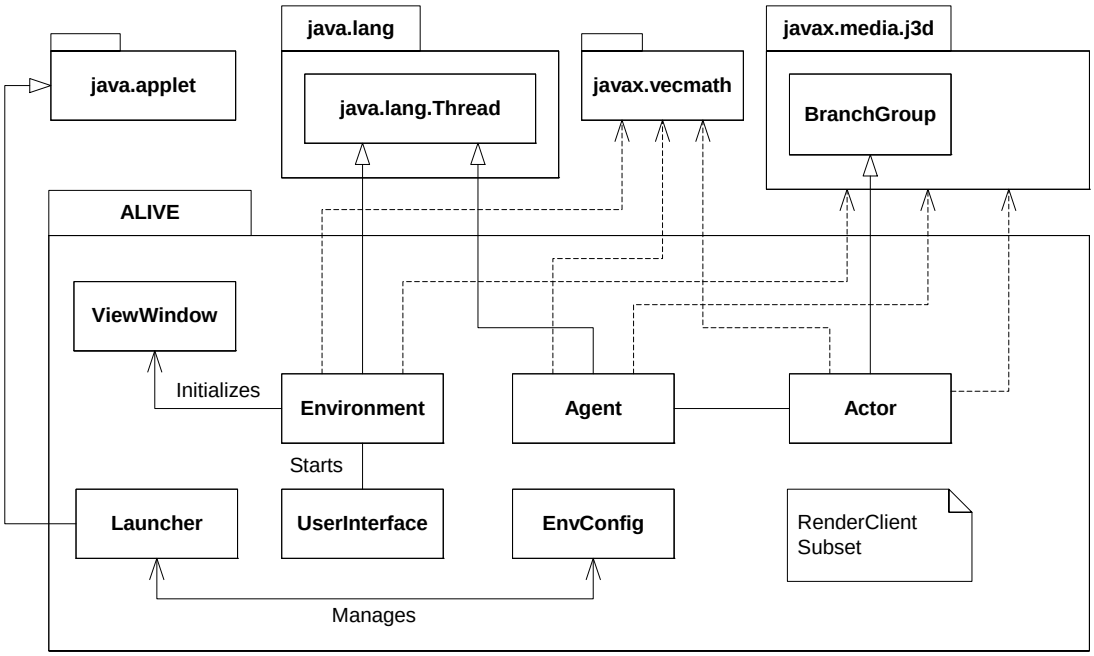

Figura 23. Diagrama de classes do projeto ALIVE e suas dependências.

A classe auxiliar (**RenderClient subset**) representa o subconjunto de cliente, que serve ao propósito da síntese de imagens em sistemas distribuídos, e será explicada posteriormente.

4.3.1. Componentes do Aplicativo Principal

A classe **Launcher** é um aplicativo de configuração para objetos do tipo **EnvConfig**, ou configuração de ambiente. Ele fornece uma interface visual que permite ajustar as opções e selecionar o experimento a ser executado, podendo ser executado como Aplicativo ou *Applet* em uma página ativa na Internet.

Com o lançador configurado, o botão "*Start*" cria um novo objeto referente ao experimento selecionado, passando como parâmetro o objeto de configuração criado.

A superclasse Ambiente (*Environment*) acondiciona toda a funcionalidade relativa a criação e manipulação do universo virtual, inicia a interface de usuário (**UserInterface**) e os dispositivos de visualização selecionados pela classe **ViewWindow**. A classe Ambiente também disponibiliza as variáveis de sistema e os métodos de acesso e controle do ambiente e dos agentes na simulação. Um experimento deve conter uma única sub-classe descendente da superclasse Ambiente, expressando a dinâmica e as regras do ambiente para o experimento implementado.

A superclasse Agente (*Agent*) é uma classe abstrata que contém todos os métodos de acesso e controle comuns para todos os agentes criados no contexto da plataforma. Um ou mais agentes desenvolvidos estenderão a superclasse agente substituindo alguns de seus métodos padrões (*overriding methods*), para que representem a dinâmica e comportamento do agente simulado. Um agente criado no contexto da superclasse deve ter um ou mais métodos construtores, responsáveis pela iniciação da instância em vários casos distintos, como quando criado inicialmente pelo sistema, com valores padrões ou aleatórios para os cromossomos, ou quando criado por outro agente, herdando alguns de seus parâmetros, aplicando mutações e cruzamentos. No caso, o construtor é selecionado pelo tipo de parâmetro passado, que pode ser um objeto do tipo ambiente, um, dois ou mais agentes. A subclasse

também deve substituir alguns métodos padrão, para que estes expressem qual ator representará o agente e quais parâmetros estão envolvidos na sua visualização e movimentação, além da dinâmica do particular agente representado. Classes auxiliares (*helper-classes*) ainda podem ser utilizadas para orientar o comportamento, como, por exemplo, classes para manipulação de redes neurais, conjuntos nebulosos ou processamento de sinais.

A superclasse Ator (*Actor*) é a última superclasse da plataforma e diz respeito à visualização em computação gráfica do experimento. A superclasse contém os métodos para manipulação do ator que representa um agente na cena. Em cada ciclo, um agente atualiza a posição, orientação e atributos visuais do ator que o representa. Alguns modelos de atores já acompanham o pacote, mas novos modelos podem ser construídos ou importados se utilizado um dos carregadores (*loaders*) fornecidos, que interpretam os formatos mais comuns de objetos 3D, como 3DStudio e LightWave, entre outros. Para construir seu próprio ator representativo dentro do contexto da plataforma, o usuário deve ter algum conhecimento de Java3D, de forma a criar uma subclasse (*Actor_Shape*), estendendo a superclasse ator e substituindo os métodos responsáveis pela construção da forma (*shape*).

Exemplificando o uso da plataforma para implementação de um experimento:

1. Uma subclasse do Ambiente (*Env_name*) deve ser criada, estendendo a superclasse Ambiente, para que expresse as leis físicas e regras que regem este ambiente específico.
2. Uma ou mais subclasses de agente podem ser criadas estendendo a superclasse Agente, expressando as regras comportamentais e mecanismos reprodutivos e de resposta dos agentes simulados.
3. Atores devem ser selecionados entre os modelos existentes para representar os agentes no universo virtual, ou novos atores podem ser criados estendendo a superclasse ator para representá-los.

O diagrama de classes do experimento exemplificado está representado na Figura 24.

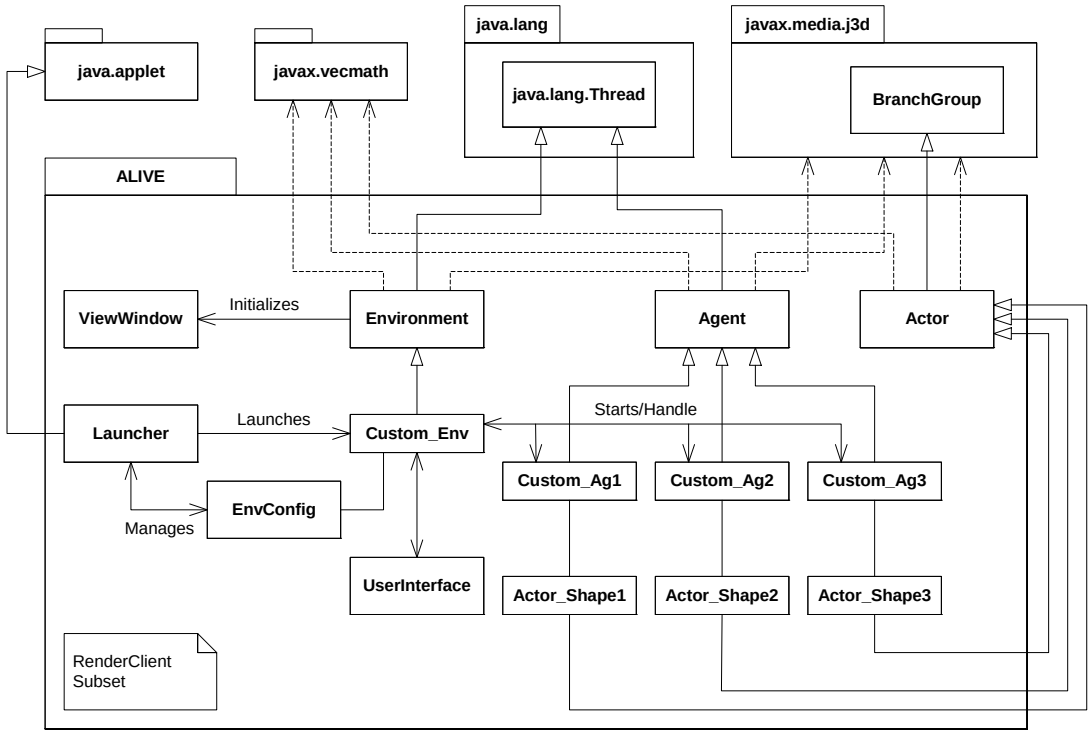

Figura 24. Diagrama de classes de um experimento desenvolvido no contexto da plataforma.

Os próximos tópicos trazem uma explicação geral sobre as classes distribuídas no pacote do projeto e uma breve descrição sobre sua utilização. O Apêndice B traz detalhes funcionais sobre a arquitetura das classes implementadas, diagramas estruturais e a especificação dos métodos disponíveis, juntamente com uma explicação detalhada de como utilizá-los.

4.3.2. Subconjunto do Cliente de Síntese

O subconjunto do cliente de síntese, formado pela classe **RenderClient** e suas dependências, é um segundo aplicativo contido no pacote que não se conecta diretamente a nenhuma classe do experimento em execução. Este aplicativo é iniciado em uma máquina remota cuja função é servir como estação de visualização do experimento, sendo executado em um servidor ligado na rede.

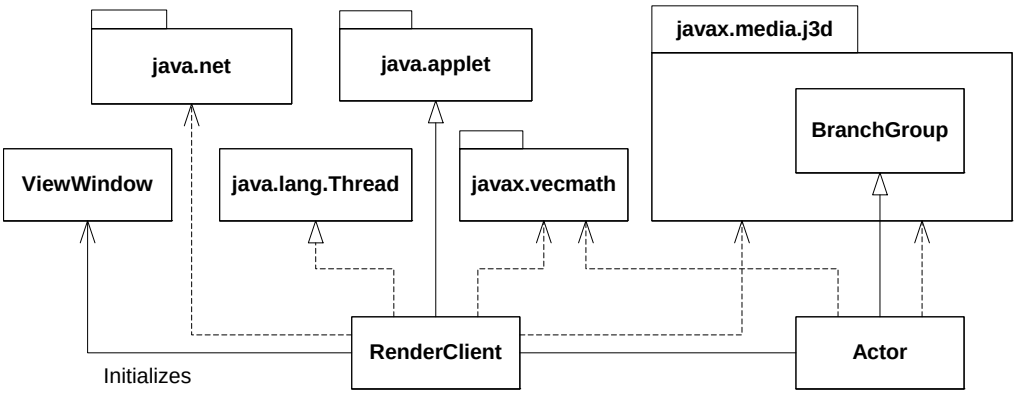

Figura 25. Esquema de comunicação do subconjunto do cliente de síntese.

O cliente de síntese de imagens pode servir tanto para reduzir a carga computacional no computador que realiza o experimento (servidor), realizando o processo de síntese em outro computador conectado na mesma rede, quanto para exibição em dispositivos com múltiplas saídas visuais, onde as saídas podem ser controladas por computadores independentes conectados em rede, como no caso da CAVERNA digital, onde um aglomerado (cluster) com cinco computadores interconectados geram as saídas para os cinco projetores, que projetam imagens em cinco paredes dispostas na forma de um cubo (quatro lados e o chão).

A classe executável **RenderClient** contém a funcionalidade necessária para iniciar uma cena e sua respectiva janela de visualização, e adicionar e operar atores nesta cena. O programa é executado remotamente e observa a rede no endereço IP e porta de difusão (*Multicast*) especificada procurando por pacotes contendo comandos do servidor. Uma vez recebidos os pacotes com tais comandos, as operações especificadas são executadas, atualizando o estado da cena, luzes e atores que representam os agentes em execução no servidor.

4.4. Utilização das Classes do Pacote

Aqui é apresentada uma breve descrição do funcionamento das classes fornecidas no pacote do projeto. Uma descrição detalhada sobre o funcionamento das superclasses e seus métodos é apresentada de maneira compreensiva no Apêndice B.

4.3.1. Superclasse Ambiente

A superclasse ambiente (**Environment.class**) contém os métodos de visualização e controle do ambiente. Para utilizar seus métodos e facilidades, um experimento desenvolvido no contexto da plataforma deve estender esta classe na construção de um ambiente particular.

```
// Exemplo de código de Ambiente construído

public class XP1_Env extends Environment {
  // Código do ambiente do Experimento 1
}
```

Alguns métodos padrões podem ser substituídos para que representem a dinâmica e propriedades do particular ambiente simulado. O ambiente também pode possuir variáveis locais, bem como utilizar as variáveis ou métodos padrões da superclasse.

```
public class XP1_Env extends Environment {
  // Variáveis locais
  int tempo_de_vida = 10000;
  double valor = 10;
  double teste = rnd(-1,1); // gerador aleatório
  ...
```

A classe filha deve possuir um único método construtor, onde são efetuados os seguintes processos:

1. Configuração do ambiente;
2. Dimensionamento do número de parâmetros do experimento;
3. Iniciação dos parâmetros e variáveis do ambiente;
4. Iniciação do experimento.

Dois passos são obrigatórios em qualquer dos construtores: o primeiro é a configuração, através do método `configureEnvironment(config)` que passa o objeto de configuração recebido a superclasse para que sejam efetuadas as mudanças necessárias; o segundo é o último passo, a chamada ao método `start()`, que inicia a execução do laço principal no código da superclasse (*thread*).

```
  // Construtor
  public XP1_Env(EnvConfig cfg) {
    configureEnvironment(cfg); // *Configura ambiente
    setMaxParameters(3);       // maximo de parâmetros
    set(0,"paramNome1",val);   // inicia parâmetros
    set(1,"paramNome2",0);
    set(2,"paramNome3",.3);
    message("Experiment 1 started...");
    super.start();             // *Inicia simulação
  }
// * Obrigatórios
```

Além do método construtor, o experimento construído deve adicionar componentes à simulação, como agentes e obstáculos. Para tanto, o método `createContent()` deve ser substituído realizando a adição de componentes na cena. O método padrão `addAgent(agent)` adiciona os agentes criados na cena.

```
  // Cria conteúdo
  public void createContent() {
    message("\nAdding agents...");
    double agTipo1 = 80, agTipo2 = 20;
    for (int i = 0; i < agTipo1; i++)
      addAgent(new xp1_Agent1(this));
    for (int i = 0; i < hunters; i++)
      addAgent(new xp1_Agent2(this));
  }
```

Outros métodos padrão podem ser substituídos no código do usuário, dentre os quais o mais importante é o método `update()`, onde são executadas as operações em tempo de execução sobre o ambiente e suas variáveis. Na forma seqüencial, os métodos padrão que podem ser substituídos ou ocultos pelo código da classe filha, são executados na seguinte seqüência:

1. `createLights()`: adiciona iluminação à cena (no método padrão duas luzes direcionais são criadas nas extremidades do retângulo volumétrico);
2. `createCustomUI()`: adiciona controles a interface de usuário (no método padrão, nada é criado);
3. `createContent()`: adiciona agentes ao ambiente (no método padrão, nenhum agente é criado);

4. `startUp()`: método é executado antes do início do laço principal;
5. <Início do laço principal `while(alive)`>
6. <atraso em milésimos de segundos (*delay*) `rest(config.sleep)`>
7. `update()`: Método deve conter a dinâmica do ambiente em execução;
8. <Executa código dos agentes*>
9. <Difusão da cena (*multicast*)>
10. <Fim do laço principal>
11. `cleanUp()`: método é executado depois do fim do laço principal;
12. <Fim>

* Apenas em modo seqüencial, sem concorrência (*multi-thread*)

O laço principal executa enquanto a variável do ambiente `alive` for verdadeira. O experimento termina quando um valor falso é atribuído à variável. Exemplo: não há mais agentes em execução.

O acesso aos agentes é feito através de uma variável que contém a lista de referências para todos os agentes em execução. A lista (ou vetor) `agents` armazena as referências para todos os agentes “vivos” ou em execução. Para operar os agentes basta percorrer esta lista efetuando os ajustes necessários com cada referência.

Os métodos `set()`, `adjust()` e `get()` são métodos sincronizados, construídos para a comunicação de parâmetros entre agentes e ambiente. Para um agente requisitar um determinado parâmetro do ambiente, deve utilizar sua variável de referência ao ambientem seguido do método, passando o nome do parâmetro requisitado entre parênteses. Exemplo: `env.get("nomedoparametro")`. O método retorna o valor `double` correspondente ao parâmetro requisitado, ou zero, caso o parâmetro especificado não exista. Da mesma maneira, para o ambiente requisitar uma informação sobre determinado agente, a expressão `agent1.get("energy")` retorna o valor correspondente à energia do agente. Acessos diretos a variáveis internas devem ser evitados por motivos explicados na seção 4.5.2.

4.3.2. Superclasse Agente

A superclasse agente é uma classe abstrata que contém os métodos de acesso e controle dos agentes desenvolvidos no contexto da plataforma. Para utilizá-la, a classe filha deve estender a superclasse na construção de um particular agente.

```
// Exemplo de código de agente construído
public class XP1_Agent1 extends Agent {
  // Código do agente 1 do Experimento 1
}
```

A classe filha deve possuir ao menos um construtor, porém, ao contrário da classe ambiente, ela pode possuir mais de um método construtor, sendo os construtores ativados alternativamente dependendo do tipo de parâmetro passado ao construtor (polimorfismo).

```
  // chamada do construtor pelo método do ambiente
  public XP_Template_Agent(Environment env) {
    super.configureToEnvironment(env);
    setMaxParameters(3);      // maximo de parâmetros
    set(0,"paramNome1",val);  // inicia parâmetros
    set(1,"paramNome2",0);
    set(2,"paramNome3",.3);
    // message("T"); // caractere do agente
    super.makeAlive();
  }

  // Chamada do construtor por outro agente (reprodução)
  public XP_Template_Agent(XP_Template_Agent parent) {
    super.configureToEnvironment(parent.env);
    cloneParameters(parent); // metodo do usuário
    mutateParameters(1%);    // metodo do usuário
    super.makeAlive();
  }
```

Como no código do ambiente, os métodos `configureToEnvironment(env)` e `makeAlive()` devem ser evocados para configurar o agente para o ambiente e iniciar sua execução consecutivamente.

Por tratar-se de uma classe abstrata, implica que três métodos declarados como abstratos devem ser substituídos no código do agente. São eles:

- `behavior()` : Método executa as operações do agente em tempo de execução;
- `createActor()` : Método deve retornar um ator válido para representar o agente na cena. Não é executado se a síntese estiver desabilitada;
- `updateActor()` : Método realiza alterações visuais no ator em tempo de execução, para representar as características desejadas. Não é executado se a síntese estiver desabilitada;

De forma similar ao ambiente, o agente também pode substituir alguns métodos padrões, que são executados na seguinte ordem:

1. `constructor()`: É executado assim que a classe é instanciada;
2. `startUp()`: É executado antes do início do laço principal;
3. <Início do laço principal `while(agent.alive)`>
4. <Atraso `rest(env.sleep)`>
5. `behavior()`: Operações com o agente em tempo de execução;
6. `updateActor()`: Operações sobre o ator em tempo de execução ;
7. <Fim do laço principal `agent.alive==false`>
8. `CleanUp()`: É executado após o término do laço principal;
9. `die()`: Efetua limpeza das variáveis criadas pela instância, remove o ator da cena e o agente do ambiente;
10. <Fim>

O laço principal executa enquanto a variável do agente `alive` for verdadeira. O experimento termina quando um valor falso é atribuído à variável por método do agente ou do ambiente.

Além dos métodos e variáveis da superclasse agente, o agente criado também pode acessar métodos e variáveis da superclasse ambiente que o iniciou, através da variável de referência `env`. Assim o agente tem acesso aos parâmetros da simulação

e a lista de agentes para realizar comunicações e operações envolvidas na interação entre agentes.

A operação da dinâmica do agente é realizada através de quatro vetores de três dimensões (variáveis do tipo `Vector3d`). São eles: posição (`position`), direção (`direction`), velocidade (`speed`) e aceleração (`acceleration`). Para operá-los, deve-se fazer uso dos métodos de acesso e das operações vetoriais disponibilizados pela classe `Vector3d` apresentados anteriormente.

4.3.3. Atores

A última superclasse do projeto foi escrita para facilitar a implementação de atores, padronizando o acesso e operação dos atores criados no contexto da plataforma, tornado-os compatíveis com qualquer agente que deseje utilizá-lo. A classe **Actor.class** é utilizada de forma semelhante a superclasse agente, sendo os métodos de acesso e operação apresentados em detalhes no Apêndice B. Segue um exemplo elementar do uso da superclasse para criar um objeto.

```
// Exemplo de código de ator

import javax.media.j3d.*;

public class Actor_Sphere extends Actor {
  Sphere sph = null;
  int resolution = 26;

  // Construtor
  public Actor_Sphere() {
    model = "Actor_Sphere";
    emissiveColor.set(0.0f, 0.0f, 0.5f);
    build();
  }
  public void createGeometry() {
    sph = new Sphere(1f, resolution, ap);
    tg.addChild(sph);
  }
}
```

No exemplo, o ator é a esfera criada no método `createGeometry()`, sendo suas características visuais definidas pela variável de aparência `ap`. Outras variáveis

afetam a aparência do ator criado, uma referência completa destas variáveis é apresentada no Apêndice B.

Além de um construtor, que deve chamar o método `build` assim que as variáveis de aparência tenham sido definidas. O método `createGeometry()`, deve ser substituído para criar a geometria desejada, adicionando-a, posteriormente, ao grupo de transformação `tg`.

4.3.4. Configuração do Ambiente

A classe **EnvConfig** carrega variáveis com parâmetros para construção, operação e visualização do ambiente. Quando se inicia um experimento, um objeto dessa classe é instanciado e modificado de acordo com as especificações desejadas para o ambiente construído.

Tabela 4. Parâmetros de configuração do ambiente.

Tipo	**Nome**	**Função**
String	codeBase	Caminho de acesso aos arquivos do projeto
boolean	STAND_ALONE	Falso se executado em modo Applet, verdadeiro se em modo Application.
boolean	RENDER	Síntese de imagens habilitada, se falso, não há saída visual
boolean	STEREO	Separação em estéreo habilitada
boolean	FULL_SCREEN	Exibição em tela cheia habilitada
double	EYE_SEPARATION	Separação entre as imagens em estéreo
double	CAM_FIELD	Ângulo de abertura da câmera
boolean	KEY_NAVIGATION	Navegação por teclado habilitada
boolean	MOUSE_NAVIGATION	Navegação por mouse habilitada
boolean	PICKING	Habilita seleção de objetos com o mouse ou dispositivo apontador conectado
boolean	view[n]	Habilita janela n = de 0 a 5
boolean	TEXT_OUTPUT	Habilita saída de mensagens para o console
String	logFile	Nome do arquivo para log das mensagens
boolean	AUTOSIZE	Dimensiona automaticamente as janelas para melhor encaixe na tela
int	xSize, ySize	Para tamanho fixo das janelas
double	xPos, yPos, zPos	Posição inicial da câmera no volume
double	xMin, yMin, zMin xMax, yMax, zMax	Dimensões do volume do espaço virtual em que se dará o experimento
boolean	wrapx, wrapy, wrapz	Habilita referência circular (*wrap around*), ao tocar a borda, objeto aparece do outro lado do volume na coordenada especificada
double	k	Constante para choque elástico com a borda (*wrap around* desabilitado)
double	POP_LIMIT	Limite populacional de agentes
boolean	SEQUENTIAL	Desabilita concorrência (*multi-threading*)
boolean	NET_CAST	Habilita difusão (*multicast*) pela rede
String	IP	Endereço IP:PORTA de difusão
int	sleep	Tempo de descanso ou atraso (*delay/cycle*)
int	frameCount	Conta os quadros sem exibição, zero mostra todos os quadros

A Tabela 4 mostra os parâmetros de configuração do ambiente, que podem ser modificados antes da execução da simulação. As variáveis, cujo nome contém todas as letras maiúsculas, não podem ser modificadas após o início do experimento (constantes).

4.3.5. Lançando o Experimento

A classe lançadora de experimentos (**launcher.class**) serve para configurar parâmetros comuns do ambiente de execução antes do início do experimento. Nesta interface é possível alterar as configurações do objeto da classe **EnvConfig.class** que será passado para o ambiente construído.

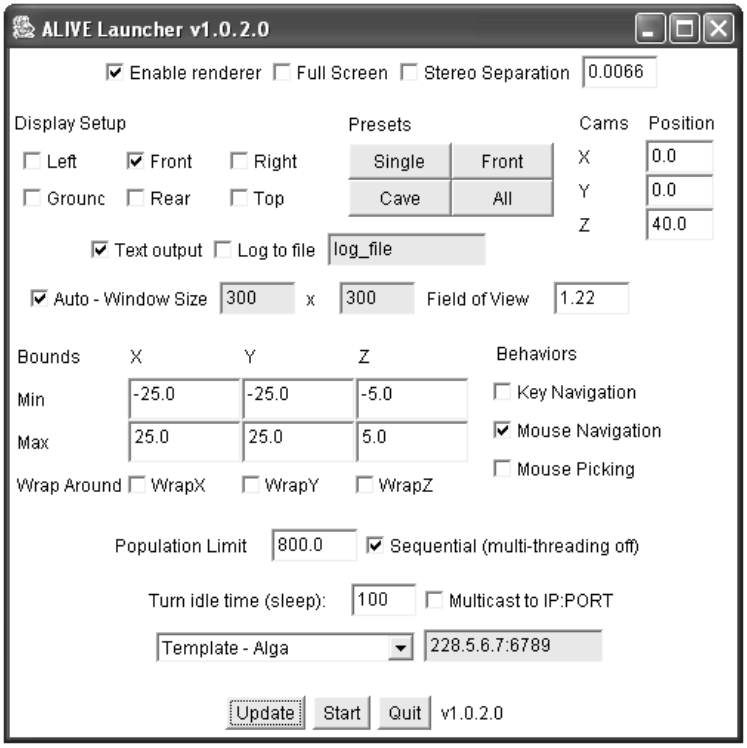

Figura 26. Tela de configuração do lançador de experimentos.

As configurações são exibidas com os valores padrão, permitindo que se altere os valores desejados, gravar ou carregar configurações do disco. O botão "*Update*" atualiza as referências do objeto de configuração, e o botão "*Start*" atualiza e passa o objeto para uma nova instância do experimento selecionado. O botão "*Quit*" nesta interface termina todos os experimentos por ela iniciados.

4.3.6. A Interface de Usuário

A interface do usuário é dividida em seis partes:

1. Controle de exibição: por onde é possível habilitar e desabilitar a síntese de imagens e a saída para as diversas janelas de visualização;
2. Interface de controle do experimento: parte reservada para que o código de um particular experimento adicione botões, campos barras de rolagem para controle das variáveis próprias da simulação;
3. Campo de texto de informação: este campo serve para exibir ou modificar os parâmetros da simulação e de agentes em execução, interativamente;
4. Seleção de agente: permite selecionar os agentes por número de identificação;
5. Controle de tempo: permite ajustar o tempo de exibição e o atraso (tempo de descanso)
6. Botões de controle.

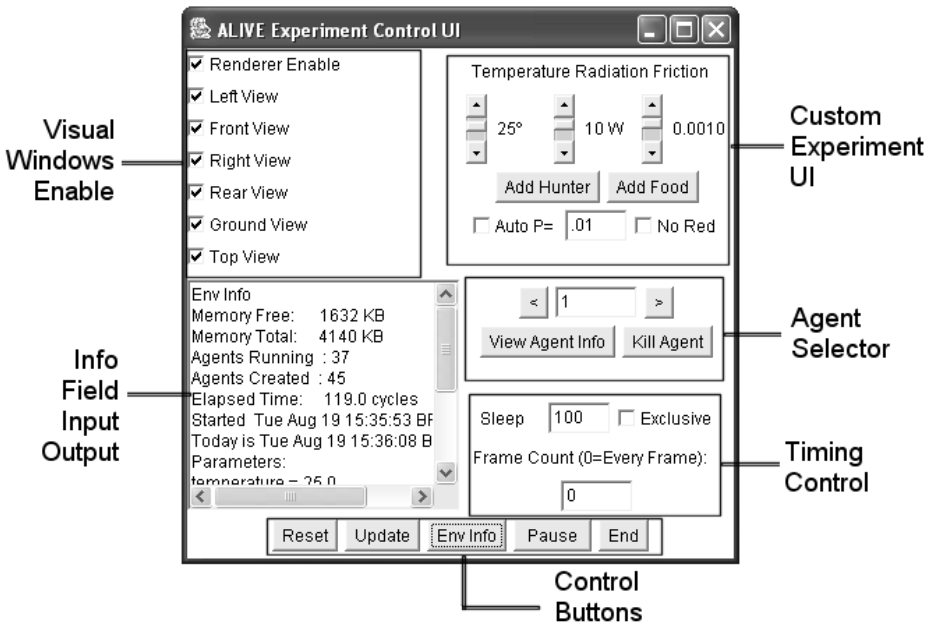

Figura 27. Interface do usuário para controle do experimento.

O controle de exibição permite salvaguardar poder computacional para os cálculos voltados ao desenvolvimento do experimento, desabilitando temporariamente a exibição em algumas janelas ou todo o processo de síntese. Este recurso é especialmente útil quando se deseja observar os resultados em longo prazo, ou não há observadores presentes, adiantando consideravelmente a evolução do experimento.

A interface de controle do experimento muda para cada experimento, que deve adicionar os controles específicos para variáveis e parâmetros, visando uma fácil manipulação dos valores pelo usuário.

O campo de texto tem duas funções: exibir os detalhes sobre o ambiente em simulação; e exibir parâmetros dos agentes em execução. Os campos que contém o símbolo de igual, geralmente, permitem modificações. Para atualizar os campos modificados, basta utilizar o botão "*Update*".

O seletor de agentes permite circular entre os agentes em execução ou selecionar diretamente por seu número único de identificação, que lhe é atribuído seqüencialmente conforme a ordem de criação.

O controle de tempo serve para ajustar o atraso (*sleep*), caso a simulação execute muito rápido ou lentamente e o número de quadros descartados entre cada exibição (zero exibe todos os quadros).

Os botões de controle apresentam as seguintes funções, respectivamente:

- *Reset*: localiza a câmera no local de origem;
- *Update*: atualiza os valores entrados no campo de texto de informações e os valores nos demais campos, como "*Sleep*" e "*Frame Count*";
- *Env Info*: exibe informações sobre o ambiente no campo de texto de informações;
- *Pause*: interrompe ou retorna a execução da simulação;
- *Quit*: termina o experimento.

A interface de controle pode, também, ter seus componentes padrões modificados pelos experimentos, embora possa resultar em um mau funcionamento das funções primárias da interface.

4.3.7. Janelas de Visualização

A classe **ViewWindow.class** é utilizada pela superclasse ambiente e cliente de síntese na criação de janelas e telas de visualização. Não é necessário ao usuário possuir qualquer conhecimento sobre seu funcionamento interno, ao menos que seja necessário adaptar seu funcionamento a algum dispositivo de visualização não convencional e não suportado pelo API Java3D com a configuração padrão.

4.3.8. Operação do Cliente de Síntese

A operação do cliente de síntese se dá através de uma interface de configuração onde são fornecidos dados de comunicação e composição visual da janela de exibição. Por esta interface, especifica-se o endereço de difusão e porta por onde o servidor distribui os pacotes, bem como os parâmetros da janela que exibirá os dados.

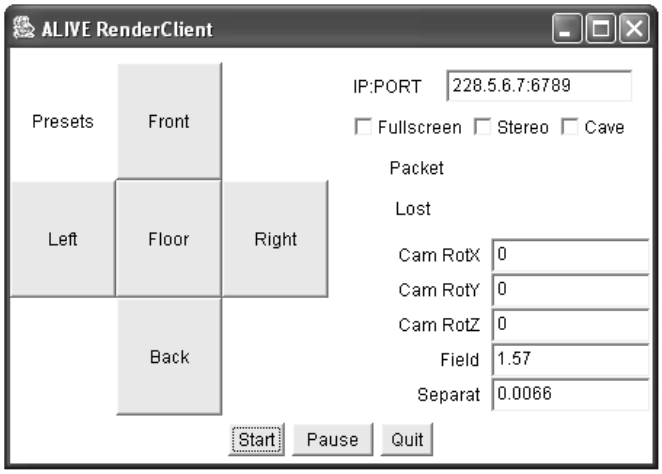

Figura 28. Interface de operação do cliente de síntese.

A interface permite utilizar pré-configurações para orientar as câmeras (*presets*), facilitando sua configuração em aglomerados (*clusters*) empregados no acionamento de dispositivos imersivos, como a CAVERNA. Para o uso em CAVERNA, os sinalizadores (*checkbox*) para estéreo e "*Cave*" devem ser selecionados para, respectivamente, habilitar a separação em estéreo e o ajuste ao tamanho físico da tela. O valor da separação entre os olhos "*Separation*" deve ser fornecido para regular o efeito estéreo quando a opção "*Stereo*" for selecionada. A saída visual pode ser feita em janela ou ocupar toda a tela (*fullscreen*).

Configurados os valores para exibição, o botão "*Start*" inicia a visualização dos pacotes, que pode ser interrompida em qualquer momento, através do botão "*Pause*". O botão "*Quit*" termina o programa.

4.5. Criando Experimentos no Contexto da Plataforma

Apresentado o uso elementar das classes contidas no pacote, resta fornecer exemplos da aplicação das classes na construção de experimentos. O pacote distribuído na página do projeto [ALIVE SITE] contém alguns experimentos desenvolvidos e uma classe modelo que pode ser utilizada para a implementação de novos experimentos. Algumas considerações, porém, devem ser feitas, dizendo respeito à execução de tarefas no modo concorrente, onde vários processos são executados simultaneamente.

4.5.1. Classes Modelo

Modelos de classes (*Templates*) são fornecidos juntamente com o pacote para demonstrar a utilização dos métodos padrões nas classes base para comunicar, orientar e mover os agentes e atores no ambiente virtual. As classes modelo implementam tipos simples de agentes e de ambiente que desempenham funções elementares.

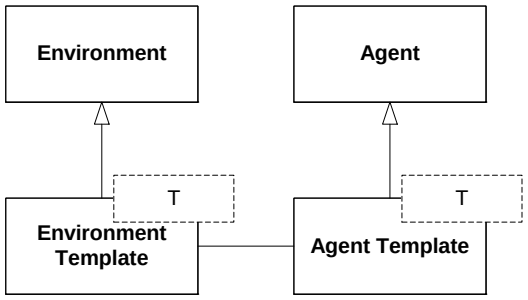

Figura 29. Classes modelo para utilização da plataforma.

As classes modelo podem ser utilizadas para iniciar novos experimentos, uma vez que possuem as referências necessárias para elaboração do código pelo usuário, como os métodos abstratos e comentários sobre seu funcionamento. Para tanto, o usuário deve copiar todo o conteúdo de texto da classe modelo para classe recém

criada (*user_Environment.class*, *user_Agent.class*), modificando no código o nome da classe e do método construtor para coincidir com o nome do arquivo. Posteriormente, deve-se introduzir nos métodos padrão, o comportamento (*behavior*) desejado para os agentes simulados e a dinâmica específica para o ambiente.

4.5.2. Conectando e Sincronizando Instâncias em Processos Paralelos

Algumas considerações devem ser feitas sobre o desenvolvimento em arquiteturas paralelas (*multi-threaded*), i.e., quando o experimento é configurado para executar concorrentemente (variável SEQUENTIAL desabilitada).

Quando o aplicativo (ou Applet) é configurado para rodar de forma concorrente, o ambiente, bem como os agentes, constituem, cada um, uma linha de execução (*thread*) paralela, significando que todos serão executados ao mesmo tempo em sistemas multi-processados, ou darão esta impressão em sistemas com um único processador, onde os processos são chaveados diversas vezes por segundo, não deixando qualquer possibilidade de determinar onde cada programa será interrompido. O fato é que em ambos os casos, os processos executarão de maneira assíncrona, sendo impossível predizer em que linha o programa estará quando receber uma chamada de acesso ou controle a um de seus métodos.

Isto pode causar diversos tipos de conflito, desde violações de acesso, quando mais de um processo tenta modificar, ao mesmo tempo, a mesma variável em uma dada instância, até laços circulares (*deadlocks*), onde dois processos travam aguardando um a resposta do outro. Além disso, valores podem mudar dinamicamente, durante a manipulação, causando erros de precisão quando um valor antigo é modificado por outra instância e colocado de volta sobre um valor que já sofreu modificações por outras instâncias.

Para lidar com tais problemas, não se recomenda o uso de referências diretas às variáveis nas instâncias operadas, e sim o emprego de métodos que operem tais variáveis. Os métodos devem ser precedidos pela palavra reservada `Synchronized`, que indica que apenas um processo por vez tem acesso a este método.

Para variáveis que sofrem alterações graduais e não abruptas, o acesso pode ser direto, porém, quanto mais rápidas forem realizadas as operações sobre a variável remota (menor número de operações), menor a probabilidade de que o valor sofra grandes alterações durante o manejo.

Para variáveis que podem sofrer alterações abruptas no tempo, o uso de uma expressão única, que realize todas as operações de teste e modificação da variável de vez, pode evitar uma falha de acesso à variável remota (quando dois objetos distintos modificam-na simultaneamente). Por exemplo, a expressão abaixo:

```
If (instância.força>4||instância.flag)
    instância.força-=1;
else instância.força+=1
If (instância.força==0)
    instância.força=1;
```

Poderia ser substituída por uma única expressão:

```
instância.força+=test(instância.força>4||instância.flag)*-1 +
                 test(instância.força==0)*1;
```

Ou ainda pelo uso de um método sincronizado que efetuaria a operação.

```
Instância.setValue(test(instância.força>4||instância.flag)*-1
                  +test(instância.força==0)*1;)
```

Esta última pode ter sua chance de falha ainda mais reduzida com o uso do seguinte código na instância operada:

```
  public synchronized double setValue(double value) {
    try {
      força = value;
    } catch (Exception e) { }
  }
```

No exemplo, o método "*test*" foi implementado para retornar 1d (*double precision*) se verdadeiro e 0d se falso.

Se necessário, pode-se empregar a utilização de semáforos para indicar que a variável está em uso. Quando uma instância assume o controle da variável, nenhuma outra instância, incluindo a proprietária da variável, poderá modificá-la, até que o controle seja liberado.

```
if (!instância.forçaHandler) {
  holdForça(this);
  instância.força+=
    test(instância.força>4||instância.flag)*-1 +
    test(instância.força==0)*1;
  releaseForça(this);
}
```

O uso de semáforos deve ser empregado em cada instância com o objetivo de liberar o acesso à variável para a instância requerente. O uso de métodos sincronizados para modificação ou captação de parâmetros é implementado no contexto da plataforma para as variáveis `param[]` e `paramName[]`, *arrays* do tipo `double` e `String` respectivamente.

O usuário que pretende empregar arquiteturas concorrentes na execução das simulações e deseja saber mais sobre programação concorrente, pode recorrer a "*Concurrent Programming in Java: Design Principles and Patterns*" [LEA, 1999].

5. EXPERIMENTOS REALIZADOS

Aqui são apresentados alguns “estudos de caso”, que ilustram a aplicação da plataforma para realização de experimentos em VA, demonstram conceitos e sugerem estudos a serem realizados com o uso da tecnologia desenvolvida. Os experimentos envolvem tanto pesquisas desenvolvidas no âmbito do próprio projeto ALIVE, cujos resultados foram submetidos para congressos e simpósios, quanto pesquisas correlatas de membros do grupo de vida artificial para demonstrar o uso da plataforma para implementação da prova de conceito em suas pesquisas.

5.1. Algas

O Experimento foi baseado na evolução de um conjunto de algas fotossintéticas. A idéia compreende parte do trabalho desenvolvido no projeto ALGA (*Artificial Life with Genetic Algorithms*) que visava estudos sobre adaptação e evolução de organismos utilizando somente mecanismos genéticos.

Simulações envolvendo populações de algas envolvem geralmente análises de variáveis discretas e de suas flutuações estatísticas. Isto se baseia no fato de que algas recebem uma quantidade finita de radiação por ciclo metabólico; seu tempo de vida é bem definido e baseado nesse ciclo; sua forma de reprodução é regular e bem definida para muitas espécies; seu movimento é involuntário, sendo causado por forças externas como ventos e correntes, ou aleatório, no caso de algas que habitam fluídos em repouso, devido unicamente à agitação térmica (movimento browniano).

Uma demonstração prática da dinâmica das algas pode ser implementada utilizando-se o contexto multi-agentes, onde as flutuações são ocasionadas por fenômenos físicos definidos no ambiente, podendo-se, assim, medir o sucesso entre os agentes, individualmente, baseado em suas características desenvolvidas, avaliando a importância de cada mutação e demonstrando quais características simuladas se demonstram mais importantes em meio à especiação.

No experimento, algas aquáticas fotossintéticas que se movem aleatoriamente em um fluido qualquer são simuladas. A produção de toxinas pode ser representada estatisticamente (sendo o resultado metabólico o oxigênio, eliminado no meio e inerte ao desenvolvimento do sistema, e o ATP, que permanece no citoplasma como reserva de energia). O movimento pode ser considerado aleatório, uma vez que apenas o movimento médio no interior da colônia é relevante. Caso esta colônia venha sendo arrastada em grupo pelas correntezas e marés, o seu referencial se move com a colônia. O nível de energia é tomado como variável, podendo assumir ciclos de emissão, simulando o ciclo solar.

O espectro da radiação ambiente pode ser controlado no experimento, permitindo observar a adaptação das algas frente a mudanças nas condições ambientais. Para simplificar o experimento, ao invés de um espectro continuo, a radiação é representada por componentes vermelho, verde e azul. A taxa de absorção das três componentes é definida cromossomicamente e sofre mutações aleatórias entre as gerações, assim como as demais informações genéticas.

Para um primeiro experimento, relaciona-se como características relevantes:

- Taxa metabólica: velocidade com a qual a alga pode converter a energia recebida em glicose;
- Taxa de consumo: velocidade com a qual o açúcar é consumido;
- Limite metabólico: valor máximo de reserva de glicose;
- Limite reprodutivo: condições nas quais a alga se reproduz;
- Limite de colapso: condições críticas, que levam à morte da alga.
- Sensibilidade espectral: à radiação, com três bandas principais (RGB).

Eventos sinalizados pelo sistema podem ser enumerados como:

- Mutação (pode haver variações bruscas no genoma);
- Variação (diferenças sutis, ou pequena variação entre os descendentes);
- Reprodução (envolve mutação e variação);

- Morte (por falta de energia ou velhice).

Cromossomos:

1. Energia
2. Tempo de vida
3. Temperatura específica
4. Limite de reprodução
5. Taxa de absorção para vermelho
6. Taxa de absorção para verde
7. Taxa de absorção para azul

Em cada ciclo, as seguintes operações são realizadas:

```
  public Behavior() {
    Lifetime--;                          // (-/+)perda ou ganho
    gain = ( -.001                           //- por turno
             - test(get("lifetime")<0)       //- metabolismo
             * Math.abs(get("lifetime"))/1000   //- idade
             - Math.abs(get("temperature") //- temperatura
             - env.get("temperature"))/40
             + env.get("r")*get("r")         //+ absorbância R
             + env.get("g")*get("g")         //+ absorbância G
             + env.get("b")*get("b")         //+ absorbância B
           ) /100;
    energy+=gain;
    if ((gain<0||energy>reproduction)&&probability(.01))
      reproduce();
    if (energy<.1)  die();
  }
```

Em cada turno, o tempo de vida é decrescido; a taxa de consumo é representada por esta perda por turno; a taxa metabólica é determinada pelas absorbâncias, que representam a sensibilidade espectral e pela diferença de temperaturas: a temperatura específica da alga; a temperatura do ambiente. A perda por velhice se dá quando o tempo de vida chega a zero, aumentando conforme o valor de `lifetime` fica mais negativo. O limite metabólico é determinado pela função de reprodução, procedimento que consome metade das reservas de energia e o limite de colapso é representado pela função `die()`, levando a morte.

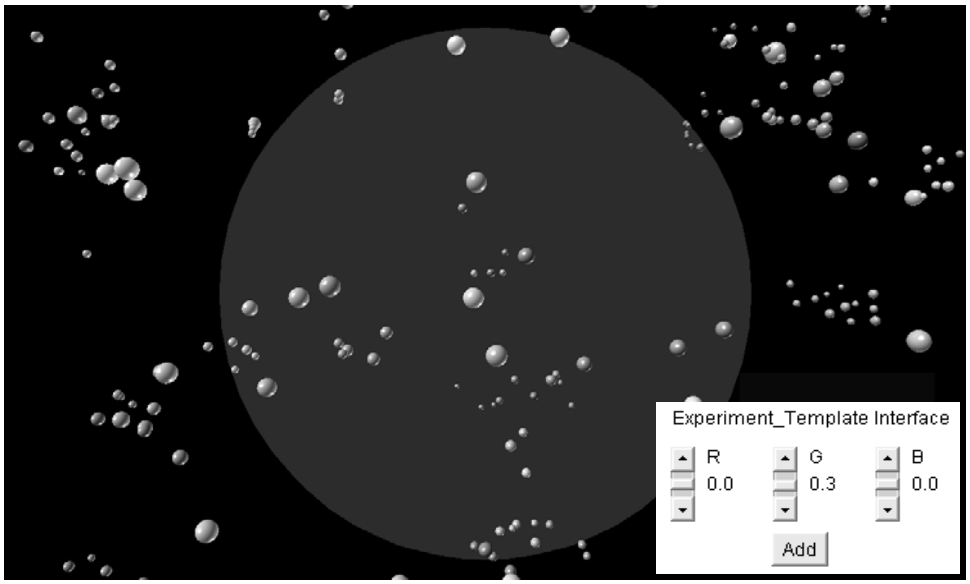

Figura 30. Tela do experimento. No detalhe a interface de controle.

Observações:

Através do experimento é possível observar a sobrevivência do mais bem adaptado, especialmente quando se alteram as condições ambientais (através do controle do experimento) em tempo de execução. A adaptação por mecanismos evolutivos (mutação e variação) leva, rapidamente, a uma nova população perfeitamente adaptada às novas condições.

5.2. Equilíbrio em um Sistema do Tipo Presa-predador

O equilíbrio em um sistema do tipo presa-predador pode ser modelado matematicamente, aplicando-se as equações de Lotka-Voltera, propostas por Lotka (1925) e Voltera (1926-1931) [KAPLAN, 1995]. O modelo permite observar as flutuações, explosões populacionais ou extinções, alterando-se parâmetros das equações ou as variáveis probabilísticas envolvidas. Aqui, no entanto, é apresentada uma alternativa ao modelo matemático para o estudo de sistemas do tipo presa-predador, utilizando-se uma população de agentes, onde é possível observar, entre outros aspectos, quais características evolutivas envolvidas nos eventos descritos são

mais relevantes, além de identificar quais delas são determinantes no processo evolutivo a longo prazo.

No experimento proposto, duas classes de agentes foram implementadas, podendo apresentar diferenças entre os vários indivíduos da mesma classe (especiação) através da mudança de parâmetros no seu código genético.

Agentes do tipo "presa" (*Food*):

O agente do tipo "presa" (alimento ou toxina) foi implementado para representar um tipo simples de célula fotossintética, que utiliza a radiação e componentes químicos presentes para sintetizar nutrientes e se reproduzir. A "presa" possui os seguintes parâmetros:

1. Energia: quantidade de nutrientes acumulados;
2. Tempo de vida: número de ciclos antes do início da perda metabólica gradual;
3. Temperatura específica: temperatura preferida ou característica;
4. Tolerância: tolerância a variações de temperatura;
5. Taxa metabólica: taxa de geração de nutrientes;
6. Taxa de crescimento: afeta a taxa metabólica;
7. Limite de reprodução: quantidade de nutrientes necessários para se reproduzir;
8. Limite de falência: quantidade mínima de nutrientes para sustentar a vida.

Os agentes desta classe podem ainda carregar energia com sinal positivo ou negativo, dependendo do tipo de nutriente sintetizado (metabolismo). Presas cuja energia tem sinal negativo representam toxinas presentes no ambiente. A identificação dos tipos de presas no ambiente virtual é feita pela cor. Alimentos, com energia positiva, são representados por esferas verdes, e toxinas, com energia negativa, são representados por esferas vermelhas.

Ainda, as seguintes observações podem ser feitas sobre as presas:

- Não possuem movimento próprio, exceto durante a reprodução;
- Reproduzem-se por meiose, gerando dois agentes semelhantes;
- Na reprodução, alguns parâmetros podem sofrer variação;
- Durante a reprodução, agentes podem ser lançados para novas áreas;
- São responsáveis por toda a geração de nutrientes no ambiente;

Agentes do tipo "predador" (*Hunter*):

O agente predador não possui qualquer mecanismo de geração de energia a partir da radiação ou temperatura, portanto, como a maioria das espécies predadoras, depende de uma fonte externa de energia para desempenhar suas funções. Os agentes do tipo predador possuem os seguintes parâmetros:

1. Geração: grau de parentesco com os agentes criados no início do experimento;
2. Energia: nutrientes armazenados;
3. Taxa metabólica: velocidade com a qual consome os nutrientes;
4. Limite reprodutivo: reserva de nutrientes necessária para reprodução;
5. Limite de falência: quantidade mínima de nutrientes para sustentar a vida;
6. Sensibilidade: raio de sensibilidade ao ambiente;
7. Força: medida quantitativa para determinação de troca de nutrientes;
8. Agilidade: quantidade de movimento produzido por turno;
9. Temperatura: temperatura preferida ou característica;
10. Tolerância: tolerância a variações de temperatura;
11. Resistência: resistência ao contato com toxinas;
12. W_1: Filtro Vermelho (R);
13. W_2: Filtro Verde (G);
14. W_3: Filtro Azul (B).

O filtro RGB age como um circuito integrador, ou um neurônio simplificado, que avalia através dos pesos W_1, W_2 e W_3 qual ação tomar frente a um estímulo luminoso recebido. O filtro também define a cor da membrana celular do agente, permitindo diferenciar, dentre as várias instâncias da classe, os indivíduos de diferentes castas.

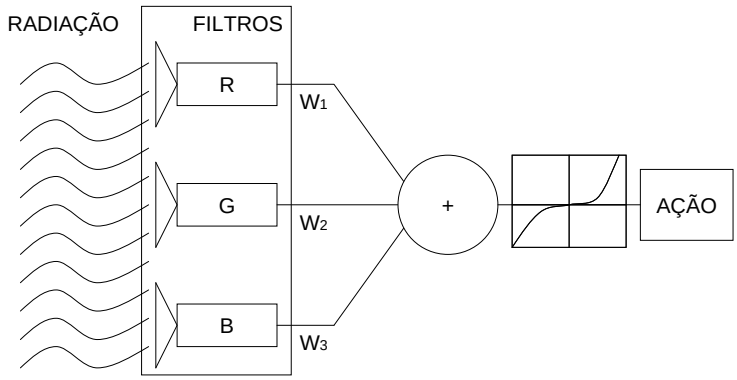

Figura 31. Esquema do funcionamento dos filtros de radiação.

Baseado na informação luminosa recebida dos agentes dentro do campo radial de sensibilidade, que é processada pelo filtro apresentado no esquema da Figura 31, o predador toma uma decisão sobre sua ação para o turno, podendo permanecer parado, preservando assim suas reservas de energia, ou gerar um impulso, gastando, para tanto, uma parcela de sua energia acumulada, proporcional a força aplicada no processo de aceleração.

Algumas observações podem ser feitas a respeito de agentes do tipo predador:

- Possuem movimento próprio e voluntário;
- Não possuem tempo de vida definido;
- Reproduzem-se por meiose;
- Durante a reprodução, geram um indivíduo semelhante;
- A reprodução envolve mutação no genoma;
- A reprodução envolve variação (pequenas mudanças);
- A sensibilidade determina o volume do ambiente que estimula o agente;
- A temperatura influencia a taxa metabólica;

- A resistência, bem como os filtros, podem assumir valores entre {+1, -1}.

A transferência de nutrientes se dá pela comparação da força: o agente mais fraco perde nutrientes com velocidade proporcional à diferença entre a força dos agentes. Os agentes do tipo presa não possuem o parâmetro “força”, portanto retornam zero ao chamado do parâmetro e, assim, são consumidos rapidamente.

O ambiente

O ambiente desenvolvido para o experimento apresenta três parâmetros, que controlam a temperatura, radiação e o atrito, respectivamente. Os parâmetros podem ser ajustados pelas barras de rolagem na interface de controle. A temperatura pode ser ajustada para se avaliar as conseqüências de mudanças ambientais ao longo do tempo. A radiação determina a quantidade de energia que é introduzida no ambiente, e pode ser aproveitada pelas “presas” fotossintéticas. O atrito anula o movimento contínuo (inércia) dos objetos da cena, e pode ser ajustado para simular uma variedade de densidades de líquidos diferentes. O atrito também afeta a quantidade de energia que os predadores gastam para se locomover, implicando numa menor quantidade de movimento por impulso.

Ao ser iniciado, o ambiente do experimento contém um número de predadores proporcional ao número de presas, que é proporcional ao volume do ambiente do experimento e limitado pela carga populacional máxima de agentes estipulada no objeto de configuração. A figura mostra um experimento em execução.

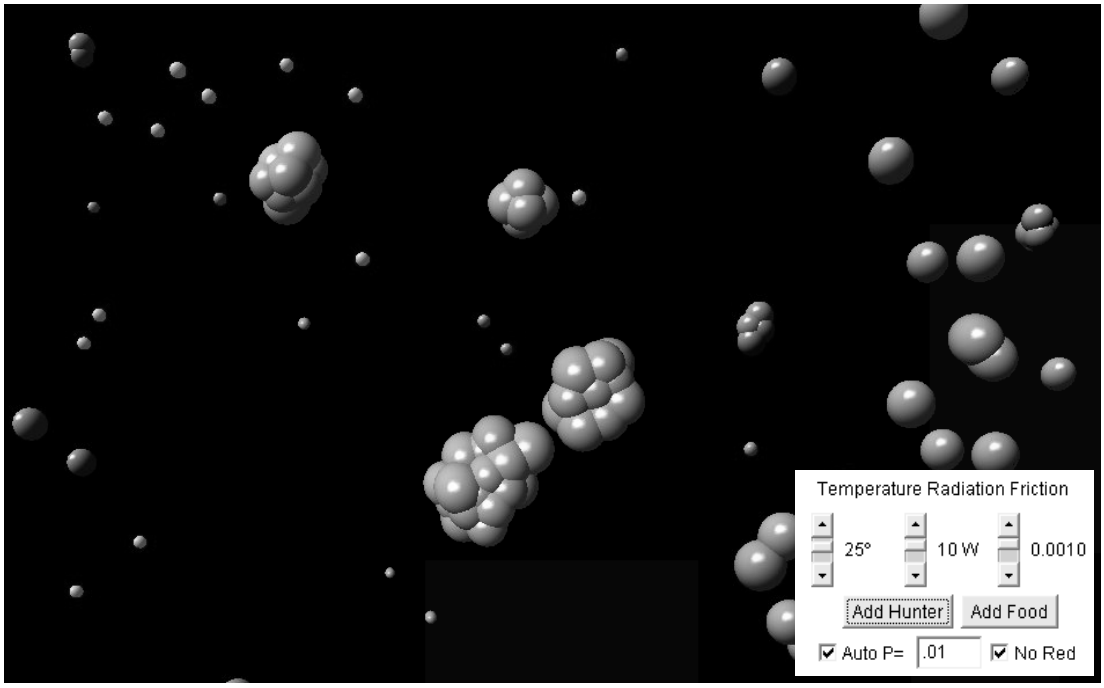

Figura 32. Tela do experimento e sua interface de controle.

Nos primeiros instantes do experimento, a seleção natural cuida de eliminar os indivíduos que nascem com genótipos desfavoráveis. Indivíduos que perseguem as toxinas morrem rapidamente; outros com uma tolerância maior sobrevivem; porém, os mais bem adaptados acabam levando uma vantagem natural.

Alguns predadores que apresentam afinidade cromática acabam por se aglomerar em colônias, o que nem sempre representa uma vantagem. Em alguns casos, um predador menos adaptado que se sente atraído por outro mais bem adaptado, pode acabar por atrapalhá-lo na busca por alimento. Alguns empurram constantemente a colônia, tornando difícil à busca por alimentos. Em alguns casos, porém, a colônia se adapta de tal maneira a trabalhar “em equipe”, de forma que todos os indivíduos agem coordenadamente para buscar os alimentos. Isso representa uma vantagem evolutiva, sendo que a colônia serve como uma reserva de nutrientes para tempos de escassez, em que os indivíduos mais fracos da colônia são devorados, lentamente, garantindo a sobrevivência da casta, especialmente dos indivíduos mais fortes.

Como é natural se esperar, com a evolução os indivíduos selecionados vão ficando cada vês mais fortes, mas ágeis, mais especializados e resistentes. Mas,

curiosamente, a adaptação em ambientes fechados leva a uma redução do raio de sensibilidade ao ambiente, levando os agentes a enxergar cada vez volumes menores, fazendo com que eles busquem apenas concentrações de nutrientes mais próximas.

Por “ambiente fechado”, entende-se: uma região ou volume finito, limitado, de onde não há entrada de agentes externos ou fuga de agentes internos. Num ambiente fechado, os agentes presentes em qualquer momento da simulação são os agentes, ou descendentes diretos dos agentes, presentes no ambiente no início da simulação.

Observações:

A evolução desenfreada de espécies predatórias em ambientes fechados geralmente, termina com o consumo de todos os recursos disponíveis no ambiente, levando as espécies presentes à completa extinção, após extinguir os recursos dos quais dependem. Geralmente, o tamanho do ambiente virtual determina o tempo de duração do experimento. Quanto menor o volume testado, mais rápido os predadores saturam o ambiente esgotando os recursos.

Um dos casos demonstrados nas equações de Lotka-Voltera, onde as presas e predadores apresentam ciclos alternados e regulares de crescimento populacional, só pôde ser observado por curtos períodos de tempo, em ambientes relativamente vastos, onde os predadores se concentram em áreas super-habitadas, deixando sementes de presa em outras áreas menos habitadas, que reiniciam o ciclo. Porém, em algum momento, uma mutação (egoísta e voraz) pode surgir desestabilizando o sistema e levando novamente ao caso anterior.

O acúmulo de toxinas torna-se um fator agravante. Não havendo espécies capazes de sintetizar o resíduo derivado da ação dos predadores, as toxinas acabam por se acumular no ambiente, impossibilitando que os predadores se locomovam livremente sem esbarrar nelas, levando à morte dos mesmos e posterior extinção dos predadores. A Figura 33 mostra a variação populacional em um experimento que apresentava

ciclos estáveis na população de predadores, até que o acúmulo de toxinas no ambiente da simulação levou à extinção os predadores.

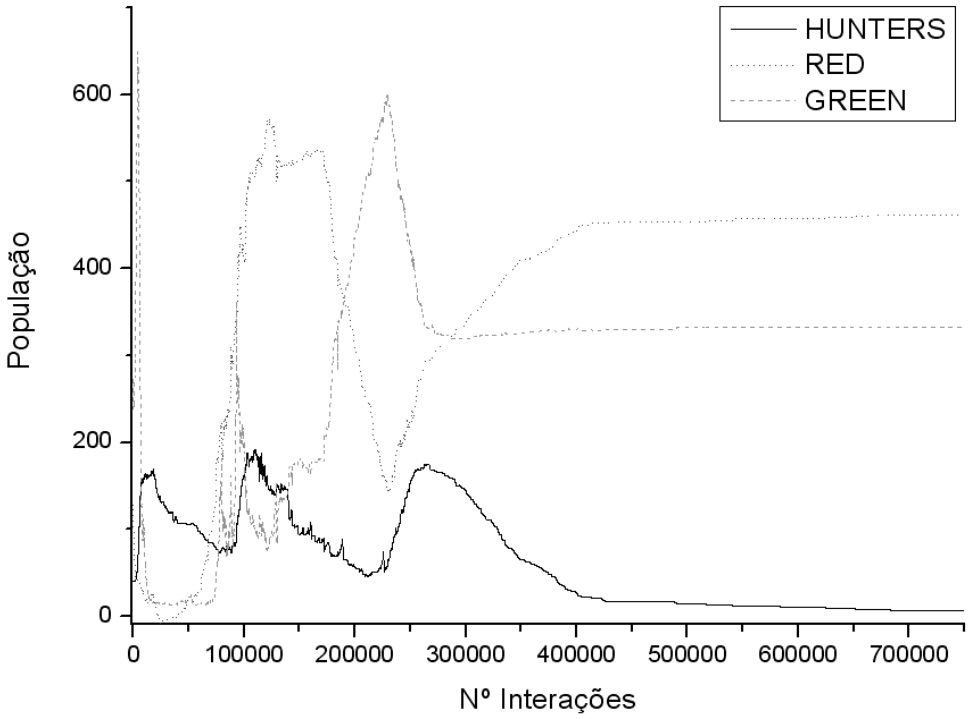

Figura 33. Gráfico da variação populacional no tempo, para um experimento realizado, no caso de extinção por acúmulo de resíduos tóxicos no ambiente.

O experimento demonstra a importância de um ecossistema completo e de manter seu equilíbrio. Pode-se observar, claramente, como a sutil alteração de parâmetros envolvidos no equilíbrio do ecossistema altera radicalmente a evolução dos organismos, podendo causar um colapso total ou parcial de várias espécies que o habitam. Nos experimentos desenvolvidos em ambientes fechados, onde os predadores dependem do sucesso reprodutivo da presa para se desenvolver, foi observado ser questão de tempo até que um salto evolutivo gerasse uma espécie altamente predatória, i.e., eficiente na busca e consumo dos recursos, multiplicando-se rapidamente, de forma que sua prole, em pouco tempo, esgota completamente os recursos disponíveis no ambiente, causando, posteriormente, sua própria extinção e das demais espécies que habitam o ambiente. A Figura 34 mostra uma extinção causada pelo surgimento de uma “mutação egoísta”.

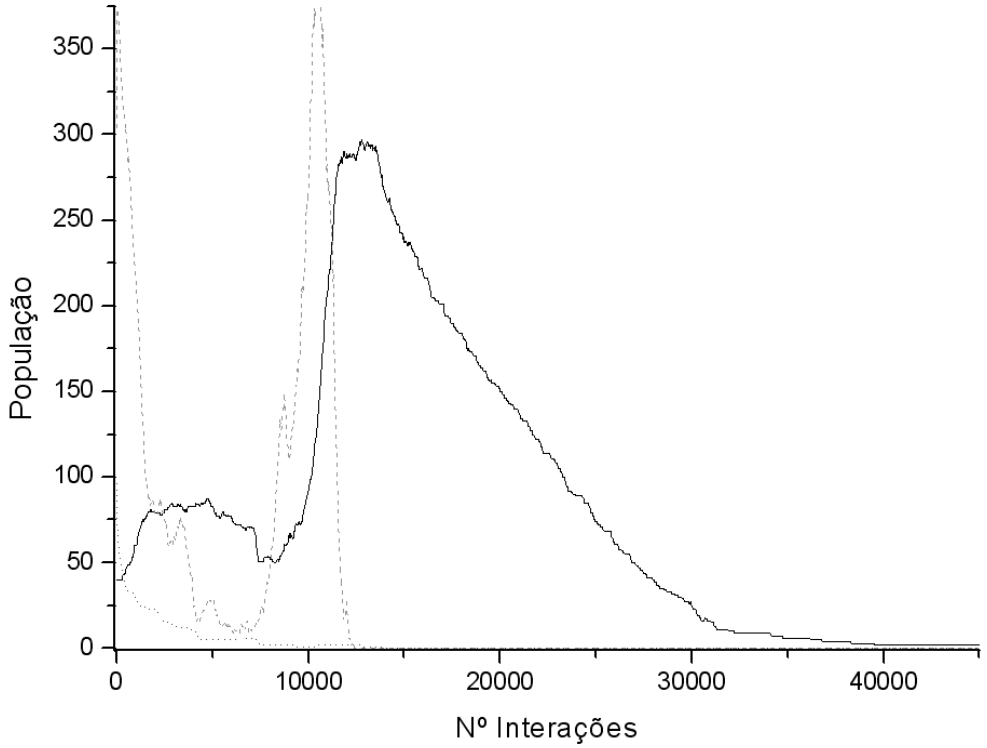

Figura 34. Gráficos de população no tempo para o caso em que ocorre extinção em massa.

Uma alternativa ao experimento foi proposta para determinar o que é inevitável na evolução dos predadores, considerando um sistema não totalmente fechado. Para tanto, um mecanismo de auto-abastecimento de alimento foi implementado, com o qual é possível ajustar a probabilidade com que o alimento é introduzido no ambiente.

Quando o tipo de alimento não é discriminado, i.e., tanto alimentos com energia positiva como toxinas podem surgir, o acumulo de toxinas no ambiente acaba levando todos os organismos presentes (predadores) ao colapso, como apresentado na Figura 33 .

Para considerar o caso ideal, apenas alimentos com energia positiva são adicionados. Nesta condição, os experimentos podem durar indefinidamente, porém, em todos os testes realizados, pode-se observar que, após um tempo de simulação, a tendência é que uma única espécie mais evoluída domine o ambiente e seus recursos, sobrepujando todas as demais espécies, que acabam perecendo (devoradas ou

famintas), até ser esta própria espécie dominante sobrepujada por uma casta descendente mais adaptada. A Figura 35 mostra a evolução de um experimento nestas condições.

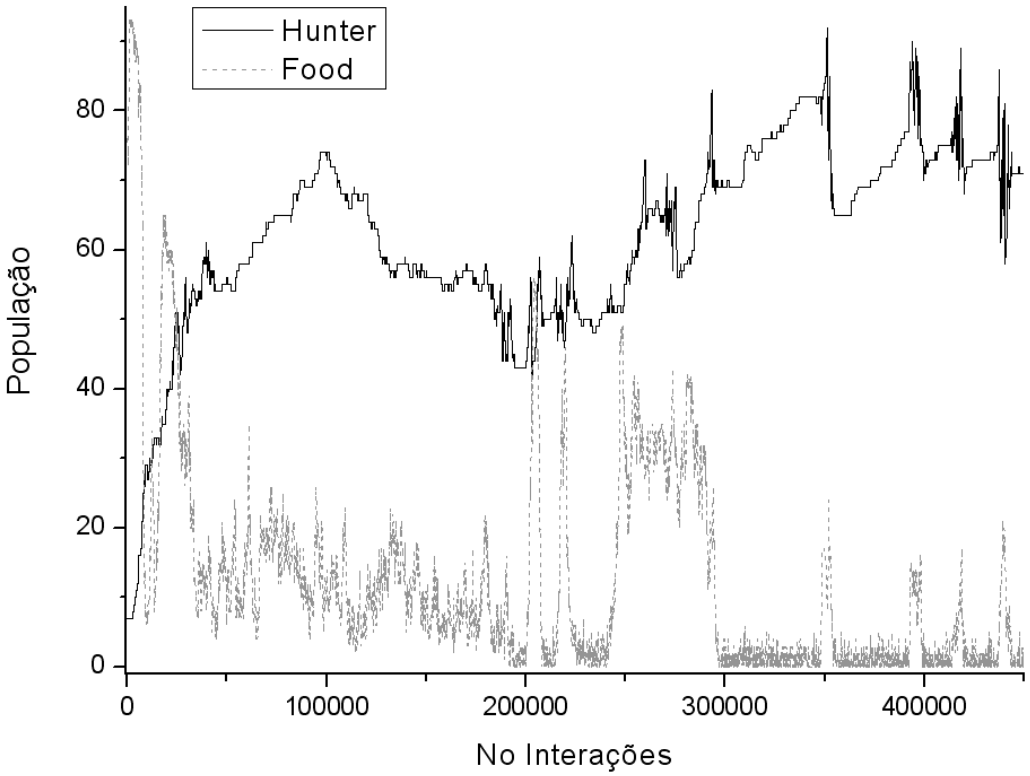

Figura 35. Variação populacional no tempo para o caso onde o sistema é abastecido com novas presas com energia positiva (alimento).

Observando os experimentos, pode-se concluir que, em um ambiente fechado e com recursos limitados, caso a espécie dominante não possua um mecanismo de auto-regulação, ou forma de administrar os recursos, ela e as demais espécies dependentes desses recursos estão fadadas a um colapso total em algum momento da sua evolução. Esta observação leva a crer que a racionalidade é um requisito obrigatório para o sucesso de uma espécie em longo prazo, visto ter sido constatado pelos experimentos que a simples evolução sem a emergência do fator cognição, leva ao inevitável fracasso.

5.3. Células Humanas

Algumas demonstrações envolvendo dinâmica de células humanas em atividade foram implementadas para demonstrar o uso do sistema para fins didáticos.

Células humanas são, na grande maioria, estáticas como as células da epiderme, ou flutuam na corrente sanguínea, transportando oxigênio como os glóbulos vermelhos, mas algumas apresentam características fascinantes. Linfócitos, por exemplo, são caçadores de proteínas, especializados em caçar padrões específicos, circulam na corrente sanguínea até encontrarem algum intruso que possua a chave correspondente a sua fechadura, i.e., sua infecção especial. Uma simulação da dinâmica dos linfócitos foi implementada para exibir a interação entre estas células e alguns tipos de invasores, como bactérias e vírus.

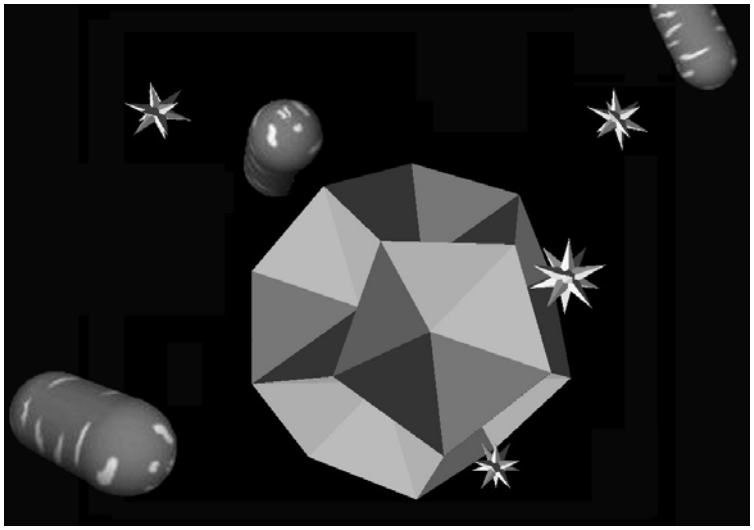

Figura 36. Tela do experimento de demonstração da ação do linfócito.

A Figura 36 mostra a demonstração executando com dois tipos de agentes infecciosos: bactérias e vírus. Os linfócitos são especializados em bactérias e os vírus simulam a ação do HIV, contaminando os glóbulos brancos e utilizando seu mecanismo interno para se replicar. O experimento apresenta apenas a interface de controle padrão.

Observações:

O experimento foi desenvolvido unicamente para ilustrar a aplicação da plataforma para fins didáticos. Para tanto, foi empregada uma filosofia de desenvolvimento *top-down*, onde os organismos foram pragmaticamente programados para apresentar o comportamento desejado e produzir, assim, os resultados esperados. Como era de se esperar, no experimento, os linfócitos "lutam" contra a infecção de bactérias continuamente, até que a população de vírus cresça demasiadamente, exterminando os linfócitos presentes. Bactérias e vírus prevalecem.

5.4. Aglomeração em Bandos (Flocking)

O comportamento conhecido como aglomeração é observado na natureza em diversas espécies, como abelhas em um enxame, peixes em um cardume ou um bando de gaivotas que sobrevoam um cardume de peixes no oceano.

O comportamento foi obtido através da evolução das espécies, demonstrando a eficiência da cooperação entre indivíduos da espécie para o desenvolvimento das mesmas. Indivíduos de espécies mais evoluídas, como a humana, aprendem rapidamente que juntos, resolvem melhor seus problemas. Um grupo de gaivotas sobrevoando um ponto específico do oceano serve como referência a pescadores para localizar um cardume. Da mesma maneira, outras gaivotas se beneficiam da informação de que um bando sobrevoando um ponto específico indica a presença de alimentos. Os peixes de diversas espécies se agrupam em cardumes, visando alimentação, reprodução, ou ainda como forma de evitar predadores, talvez dando a impressão de juntos, aparentarem ser um peixe maior. A evolução levou muitas espécies a desenvolverem um desejo inato de estar próximo de seus semelhantes. Na própria raça humana se observa o desejo de socialização. A necessidade de se relacionar com outros indivíduos, em grupos ou em locais de aglomeração, só não é compartilhada por eremitas, indivíduos reclusos, ou que apresentam desvios de personalidade, considerando que o compartilhamento de informação é um fator essencial no desenvolvimento humano.

O experimento demonstra o comportamento apresentado por um grupo de agentes (representados por gaivotas), guiados apenas pelo desejo de se aproximarem de seus semelhantes. Uma rápida visão do ambiente dá ao agente a sensação de quantidade, fazendo com que ele regule suas ações motoras para tentar se aproximar do ponto de maior aglomeração.

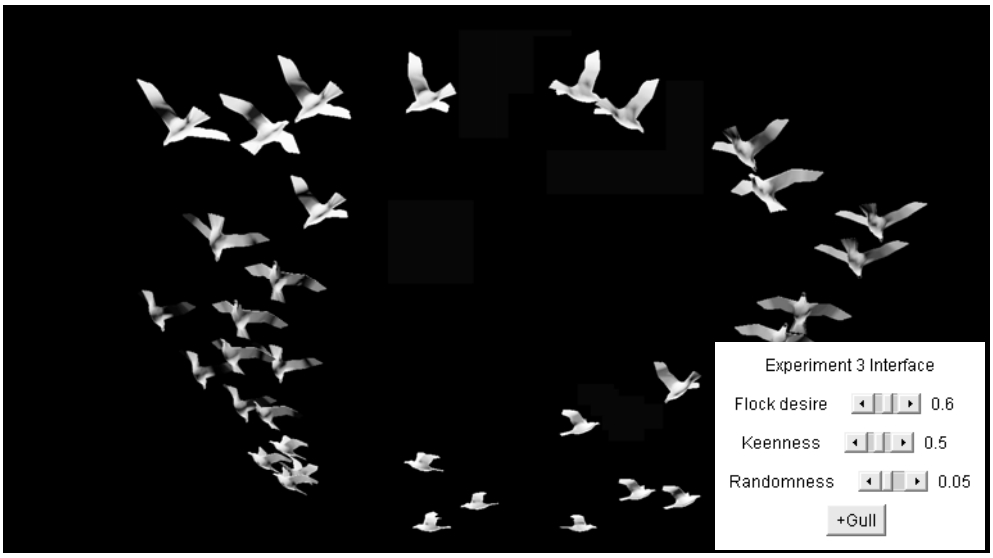

Figura 37. Tela do experimento e sua interface de controle.

A Figura 37 mostra o experimento em execução. É possível se obter diversos padrões diferentes de aglomeração variando-se os parâmetros na interface de controle, intitulados, respectivamente, “desejo de aglomerar”, “precisão” e “aleatoriedade”, e que representam probabilidades associadas ao comportamento do bando. A analogia entre o experimento e observações na natureza vai além das gaivotas representadas, para algumas combinações de valores de parâmetros, os agentes passam a se aglomerar como em um enxame de abelhas ou como um cardume de peixes, demonstrando a similaridade entre os comportamentos.

Simulações de comportamento coletivo ou *flocking*, que estudam o comportamento de animais em bandos, são comuns em programas de VA. Entre os mais conhecidos estão: Enxame de Abelhas (*bees*), Cardume de Peixes, as simulações conhecidas como *Floyds* e *Boyds*, muitos destes, que possuem implementações rodando em *Applets* na Internet [DOLAN SITE] [REYNOLDS SITE]. O grupo “Swarm” [SWARM SITE] também desenvolve pesquisas relacionadas ao tema. Os

experimentos demonstram como seres artificiais podem apresentar ou desenvolver comportamentos coletivos a partir de regras muito simples de interação.

5.5. Cardume de Peixes

O experimento cardume de peixes tem embasamento em um projeto realizado pelo grupo de Vida Artificial em conjunto com o grupo de Ciências Cognitivas [ARTLIFE SITE], que visa estudar a relação entre cognição e evolução nos seres virtuais, permitindo comparar o comportamento in-nato, atingido através da evolução, ao adquirido com o acúmulo de conhecimento ao longo da vida, avaliando quais as vantagens e desvantagens relacionadas a cada um dos métodos de aquisição de conhecimento.

A princípio, qualquer agente poderia ser usado para implementar o experimento, porém, a simplicidade do sistema nervoso individual e o comportamento errático dos agentes na fase de aprendizado, levam a uma analogia natural com um cardume de peixes, uma vez que peixes já possuem um comportamento instintivo definido através da evolução e seus primitivos sistemas nervosos sofrem muito pouca alteração ao longo de suas vidas.

Para implementar o conceito, os agentes utilizam um módulo de rede neural multicamada para representar as relações entre sentido e comportamento, ou formalmente, entrada e saída. Cada indivíduo é criado com uma rede neural iniciada aleatoriamente, que constitui parte de seu genoma, podendo esta sofrer alterações ao longo da vida, que são mantidas caso o resultado da alteração seja vantajoso. Quando o agente se reproduz, a rede neural contida no genoma é passada, e não a que se encontra atualmente em uso pelo agente.

Isto significa que:

- Cada agente possui um conhecimento *in nato*, representado pela rede neural *N*, contida em seu genoma;

- Quando nasce, sua rede neural *M* é iniciada com os valores W_{ij} contidos na rede neural *N*;
- Durante sua vida, a rede neural *M* sofre mudanças, que são mantidas, caso o resultado das mudanças se demonstre vantajoso;
- No ato da reprodução, o novo agente recebe dos pais uma combinação de seus genomas, que representam um cruzamento das suas redes neurais *N*, dado também um fator de mutação.

O experimento compreende a propagação de conhecimento na população de agentes, significando que um agente pode treinar sua rede neural observando o comportamento, ou seja, as relações entre entrada e saída de outros agentes próximos. Neste caso, a rede *M* pode sofrer dois tipos de alteração durante sua vida: devido a flutuações aleatórias e devido a observações do ambiente. Em ambos os casos, as mudanças só são mantidas caso apresentem algum benefício.

Uma função de auto-satisfação permite ao agente avaliar quantitativamente o sucesso obtido com o uso da nova rede neural, em detrimento da antiga. A nova rede neural modificada é mantida então por um determinado número de ciclos, sendo mantida após este período caso o valor médio da função satisfação seja superior ao obtido com o uso da rede antiga. Em termos, o peixe esquece o conhecimento inútil adquirido ou experimentado.

Além da rede neural inicial, o genoma contém os seguintes campos.

1. Rapidez: velocidade máxima que o peixe desenvolve;
2. Precisão: medida quantitativa do grau de acerto;
3. Taxa de aprendizado: afeta as mudanças na rede neural;
4. Variância comportamental: determina a quantidade de alterações que a rede sofre durante a vida;
5. Freqüência de variância: determina a probabilidade de sofrer uma mudança comportamental;
6. Tempo de avaliação: número de ciclos para que a nova rede seja avaliada;

7. Idade de reprodução: numero de ciclos antes que possa se reproduzir;
8. Tempo de vida: número de ciclos metabólicos;

O experimento é atualmente objeto de estudo e já possui resultados parciais. A Figura 38 mostra a tela do experimento em execução.

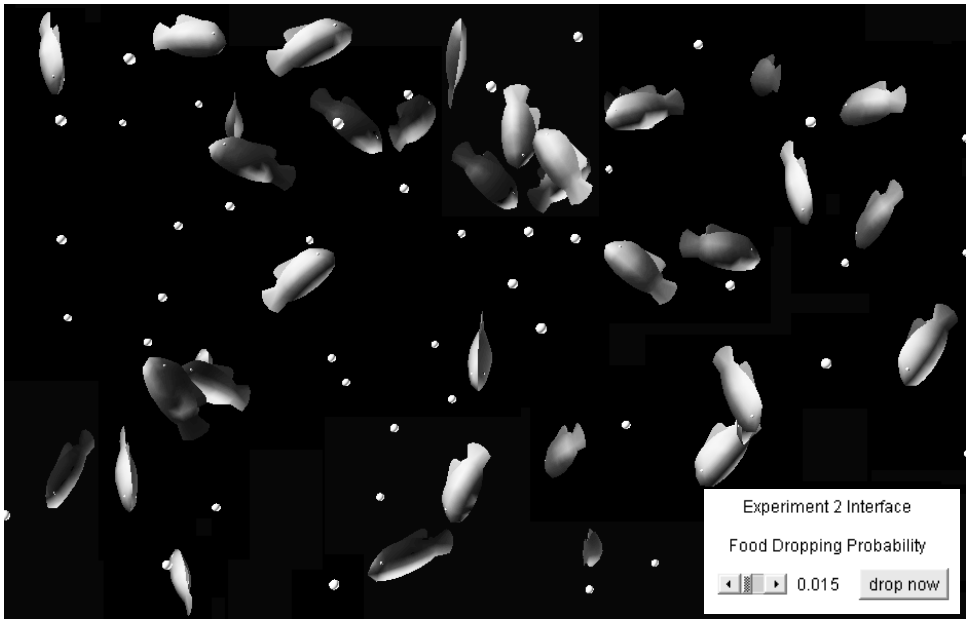

Figura 38. Tela do experimento cardume de peixes, no detalhe a interface para controle da alimentação.

Observações

Observações em longo prazo são necessárias para comparar aspectos evolutivos e cognitivos, e avaliar as vantagens do aprendizado em relação à simples evolução comportamental ou instinto. Porém, em algumas horas de simulação já é possível notar que os agentes que já nascem com um comportamento instintivamente eficiente, levam vantagem em relação aos que tem que aprender com a experiência, e acabam por desperdiçar parte do seu tempo de vida neste processo. Por outro lado, as espécies que nascem com um conhecimento menor e aprendem mais rápido, levam vantagem em experimentos em que ocorrem mudanças drásticas no ambiente ou no padrão de abastecimento de alimento, se adaptando às novas condições, enquanto os indivíduos mais dependentes do instinto, acabam morrendo por não se adaptarem.

6. CONCLUSÕES FINAIS

Durante a implementação do projeto, este sofreu diversas modificações. Porém, as idéias iniciais foram basicamente mantidas. As modificações sofridas visavam tanto melhoras na arquitetura, quanto aumento da flexibilidade do programa. Espera-se que tais mudanças continuem ocorrendo, enquanto a plataforma experimental continuar em desenvolvimento. Também, a filosofia de código aberto permite que o projeto receba contribuições externas ao laboratório onde foi inicialmente criado.

Além do uso para estudos científicos, espera-se que o projeto sirva como ferramenta de divulgação para trabalhos em VA, através da disponibilidade na Internet de experimentos, juntamente com o código fonte, permitindo transformar qualquer computador pessoal em um laboratório de estudos em VA.

A seguir são feitas algumas considerações sobre as contribuições e aplicações do projeto desenvolvido, bem como algumas sugestões para trabalhos futuros a serem desenvolvidos com base na pesquisa atual.

6.1. Contribuições

Entre as contribuições realizadas pelo projeto (*achievements*), vale citar o reforço ao vínculo entre Vida Artificial e Realidade Virtual, vínculo, este, ainda pouco explorado por programas de VA e computação gráfica. O uso de tecnologias de imersão total em ambientes virtuais ainda reforça a aplicação de Realidade Virtual para visualização de experimentos científicos e exibição de massa de dados utilizando-se, para tanto, artifícios visuais comuns à área de computação gráfica, como texturas, transparência, coloração e movimento.

O desenvolvimento da plataforma como foi apresentada, ainda fornece uma ferramenta de rápida prototipação, não só para experimentos de VA, mas, também, para programas de computação gráfica, que venham a utilizar o contexto multi-agentes para manipulação de objetos em Realidade Virtual.

6.2. Aplicações

Aqui são sugeridas aplicações do tema em várias frentes, como pesquisa ambiental, educação média e superior e descoberta de conhecimento, entre outras.

Experimentos em Vida Artificial, especialmente num contexto multi-agentes, podem ser aplicados a uma larga gama de áreas das ciências exatas e biológicas, entendendo desde simulações didáticas, visando a ilustração de conceitos básicos de fenômenos biológicos até soluções de problemas em robótica e engenharia, como demonstrado em [NEVES, 2002].

Os conceitos estudados aqui poderão ainda ter relevância em estudos futuros, envolvendo áreas como nanotecnologia, onde se pode desejar que nano-robôs, constituídos de poucos átomos e que realizam um conjunto de regras simples, desempenhem em conjunto, tarefas pré-definidas de maior complexidade. Tais robôs podem evoluir em um ambiente simulado, até que atinjam uma operabilidade aceitável, para que sejam, posteriormente, construídos.

Simulações envolvendo sistemas genéticos e busca evolutiva podem ser igualmente beneficiadas, uma vez que computadores de alta-performance podem realizar, em minutos, interações que levariam milhares de anos. O exemplo de evolução de algas demonstra bem esta capacidade.

Exemplos didáticos foram desenvolvidos demonstrando fenômenos celulares como mitose e fagocitose. Ainda há exemplos interativos, que permitem a um aluno mais interessado modificar parâmetros dos experimentos, ou mesmo criar novos experimentos, combinando agentes pré-definidos e observando suas interações. O exemplo de agentes patogênicos ilustra bem essa possibilidade, onde células comuns podem ser misturadas a agentes nocivos em um experimento, observando-se a evolução da infecção de maneira análoga a experimentos *in vitro*.

O programa disponibilizado na Página do Projeto [ALIVE SITE] possibilita o acesso remoto e execução de *Applets* através de páginas interativas, permitindo que estudantes e entusiastas experimentem em casa simulações em VA, bastando para isso possuir instalado o suporte à linguagem Java e ao API Java3D.

O aplicativo desenvolvido possibilita a visualização de experimentos em imersão total no ambiente virtual, utilizando desde óculos 3D até a CAVERNA [CAVERNA SITE] entre outros, utilizando, também, dispositivos de rastreamento (*tracking*), que permitem ao usuário interagir com o experimento em tempo de execução, alterando parâmetros dos agentes simulados.

Futuramente, agentes complexos poderão desempenhar tarefas de alto nível nos ambientes virtuais, como a construção de cenários, crescimento de vegetação ou até construção de arquiteturas baseadas em um conjunto de regras pré-estabelecidas (como colméias, formigueiros, casas ou outras obras baseadas em engenharia evolutiva).

Agentes “inteligentes” podem efetuar buscas no espaço virtual, com auxílio do usuário. Para tanto, vetores de um espaço de busca podem ser mapeados nas coordenadas *x*, *y*, *z* e em atributos visuais, nos quais os agentes possam basear sua busca, recebendo dicas do usuário, que observa sua evolução.

Visando explorar a interação com os seres humanos, agentes cognitivos com características humanas podem ser implementados, aplicando uma série de técnicas de VA e IA. Pesquisas envolvendo agentes cognitivos e com características humanas estão atualmente sendo desenvolvidas utilizando o contexto da plataforma e já apresentam alguns resultados [CAVALHIERI SITE].

6.3. Propostas para trabalhos futuros

Aqui são descritos alguns aprimoramentos e funcionalidades que podem ser implementadas futuramente no projeto, bem como experimentos que podem ser realizados utilizando-se, para sua implementação, o contexto desenvolvido.

Entre os aprimoramentos a serem desenvolvidos no programa, consta um ajuste no sistema de manipulação de agentes no universo virtual, especialmente em sistemas imersivos, como a CAVERNA, permitindo a movimentação de agentes utilizando-se dispositivos apontadores. As funcionalidades de acesso a arquivo, que envolvem leitura e gravação de experimentos, agentes e configurações, não foram implementadas por implicarem em limitações no funcionamento do programa em modo *Applet*. *Applets* apresentam uma série de limitações relativas à segurança. Uma, em especial, impede que *Applets* e scripts executando em navegadores tenham acesso à leitura e gravação no disco do usuário. É sugerido que, futuramente, o projeto seja dividido em dois módulos independentes, sendo um voltado à exibição e divulgação na Internet, em modo *Applet*, e um para uso como aplicativo, apresentando uma série de ferramentas adicionais, entre elas, métodos para armazenamento e recuperação de informação em disco.

Entre os experimentos a serem desenvolvidos, é sugerido o desenvolvimento de uma demonstração envolvendo sistemas com morfologia variável, para observar como o aspecto físico é influenciado pela evolução e regras de encaixe (*fitness*). O genoma, neste caso, representaria um programa contendo códigos simples que coordenariam as ramificações, informando os comprimentos e ângulos, garantindo uma variedade na obtenção de formas construídas por suas variações. Uma função de satisfação é definida permitindo escolher qual critério será considerado como vantajoso no processo evolutivo, como: maior volume, maior área de superfície, maior comprimento, relação entre altura e largura, ou uma combinação entre estes e outros. O experimento permitiria observar como as formas variam através da evolução, dependendo das regras e critério definidos (função de encaixe, satisfação ou *fitness*).

BIBLIOGRAFIA

Referências Bibliográficas

[ADAMI, 1998]
ADAMI, CHRISTOPH
"An Introduction to Artificial Life"
Springer Verlag / Telos (1998)
ISBN: 0-387-94646-2

[BEDAU, 2000]
BEDAU, MARK A. ET AL.
"Open Problems in Artificial Life"
Artificial Life 6, MA, 363-376 (2000)
Também disponível em:
<http://mitpress.mit.edu/journals/ARTL/Bedau.pdf >, acesso em 15/8/2003

[BENTLEY, 1999]
BENTLEY, PETER J. (EDITOR)
"Evolutionary Design by Computers"
Morgan Kauffmann (1999)
ISBN: 1-55860-605-X

[DARWIN, 1859]
DARWIN, CHARLES
"A Origem das Espécies"
Editora Hemus; 5ª Edição (2000)
ISBN: 8-528-90134-3

[DAWKINS, 1976]
DAWKINS, RICHARD
"O Gene Egoísta"
Editora Gradiva (1976)
ISBN: 972-662-127-5

[DEITEL, 2002]
DEITEL, HARVEY M.; DEITEL, PAUL J.
"Java, como programar"
Bookman Companhia Ed., 4ª Edição (2002)
ISBN: 853-630-123-6

[DYSON, 1999]
DYSON, FREEMAN
"Origins of Life"
Cambridge University Press (1999)
ISBN: 0-521-62668-4

[ECKEL, 2002]
ECKEL, BRUCE
"Thinking in Java"
Pentice Hall (2002)
ISBN: 013-100-287-2
Também disponível em:
<http://www.mindview.net/Books/TIJ/>, acesso em 15/8/2003

[FRANKLIN, 1996]
FRANKLIN, S.; GRAESSER, A.
"Is it an Agent or just a Program? A Taxonomy for Autonomous Agents"
In Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages
Springer-Verlag (1996)

[FRANKLIN, 1997]
FRANKLIN, STAN
"Artificial Minds"
MIT Press (1997)
ISBN: 0-262-56109-3

[HAYES, 1995]
HAYES-ROTH, B.
"An Architecture for Adaptive Intelligent Systems"
Artificial Intelligence: Special Issue on Agents and Interactivity, 72, 329-365 (1995)

[HAYKIN, 1998]
HAYKIN, SIMON S.
"Neural Networks: A Comprehensive Foundation"
Prentice Hall; 2nd edition (July 6, 1998)
ISBN: 0-132-73350-1

[KAPLAN, 1995]
KAPLAN, DANIEL; GLASS, LEON
"Understanding Nonlinear Dynamics"
Springer (1995)
ISBN: 0-387-94440-0

[KOVACS, 1997a]
KOVÁCS, ZSOLT L.
"Redes Neurais Artificiais"
Collegium Cognitio / Edição Acadêmica (1997)
ISBN: 0-262-13316-4

[KOVACS, 1997b]
KOVÁCS, ZSOLT L.
"O Cérebro e sua Mente"
Ed. Zsolt Kovács / Edição Acadêmica (1997)

[KREYSZIG, 1998]
KREYSZIG, ERWIN
"Advanced Engineering Mathematics"
John Wiley & Sons; 8th edition (January 1999)
ISBN: 0-471-15496-2

[LANGTON, 1989]
LANGTON, G. C. (EDITOR)
"Artificial life: The proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems"
Held September, 1987, in Los Alamos, New Mexico
Addison-Wesley Pub. Co., Advanced Book Program (Redwood City, Calif.) (1989)
ASIN: 0201093561

[LANGTON, 1995]
LANGTON, G. C.
"Artificial Life: An Overview (Complex Adaptive Systems)"
MIT Press; Reprint edition (January, 1997)
ISBN: 0-262-62112-6

[LEA, 1999]
LEA, DOUG
"Concurrent Programming in Java™: Design Principles and Pattern"
Addison-Wesley Pub Co; 2nd edition (November 5, 1999)
ISBN: 0-201-31009-0

[LEE, 1990a]
LEE , CHUEN CHIEN,
"Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part I"
IEEE Transactions on Systems, Man, and Cybernetics, vol 20, nº 2, March/April 1990.

[LEE, 1990b]
LEE, CHUEN CHIEN
"Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part II"
IEEE Transactions on Systems, Man, and Cybernetics, vol 20, nº 2, March / April 1990.

[MAES, 1995]
MAES, PATTIE
"Artificial Life Meets Entertainment: Life like Autonomous Agents"
Communications of the ACM, 38, 11, 108-114 (1995)

[MARGULIS, 1998]
MARGULIS, LYNN; SAGAN, DORION
"What is Life?"
University of Califórnia Press (1998)
ISBN: 85-7110-641-X

[MICHALEWIC, 1996]
MICHALEWIC, ZBIGNIEW
"Genetic Algorithms + Data Structures = Evolution Programs"
Springer Verlag (1996)
ISBN: 3-540-60676-9

[MITCHELL, 1997]
MITCHELL, MELANIE
"An Introduction to Genetic Algorithms"
MIT Press (1997)
ISBN: 0-262-13316-4

[MORIE, 1994]
MORIE, J.F.
"Inspiring the Future: Merging Mass Communication, Art, Entertainment and Virtual environments"
Computer Graphics, 28(2):135-138, May 1994.

[NEVES, 2002]
NEVES, ROGÉRIO P. O.; NETTO, MARCIO L.
"Evolutionary Search for Optimization of Fuzzy Logic Controllers"
1st International Conference on Fuzzy Systems and Knowledge Discovery
Volume I, on Hybrid Systems and Applications I (2002)
ISBN: 981-04-7520-9
Também disponível em:
<http://www.lsi.usp.br/~rponeves/work/fuzzy/>, acesso em 1/9/2003

[RASBAND, 1990]
RASBAND, S. NEIL
"Chaotic Dynamics of Nonlinear Systems"
A Wiley-Interscience publication, John Wiley & Sons (1990)
ISBN: 0-471-63418-2

[RUSSEL, 1995]
RUSSELL, STUART; NORVIG, PETER
"Artificial Intelligence: A Modern Approach"
Englewood Cliffs, NJ: Prentice Hall; 2nd edition (December, 2002), page 33
ISBN: 0-137-90395-2

[SCHRÖDINGER, 1943]
SCHRÖDINGER, ERWIN
"O que é Vida?"
Editora Unesp (1943)
ISBN: 8571391610

[SHLOAM, 1998]
SHLOAM, Y.
"Agent-Oriented Programming"
Readings in Agents - editado por M.N. Huhns e M.P. Singh pags. 329-349 - Morgan & Kaufmann, S. Francisco (1998)

[SMITH, 1994]
SMITH,D.C.; CYPHER,A.; SPOHRER,J.
"KIDSIM: Programming Agents Without a Programming Language"
Communications of the ACM, v.37, n.7, pags. 55-67 (1994)

[WOOLDRIDGE, 1995]
WOOLDRIDGE, MICHAEL; NICHOLAS R. JENNINGS,
"Agent Theories, Architectures, and Languages: a Survey"
Wooldridge and Jennings Eds., Intelligent Agents
Berlin: Springer-Verlag, 1-22 (1995)

[WOLFRAM, 2002]
WOLFRAM, STEPHEN
"A New Kind of Science"
Wolfram Media, Inc.; (May, 2002)
ISBN: 1579550088

[ZADEH, 1973]
ZADEH, LOTFI ASKER
"Outline of a New Approach to the Analysis of Complex Systems and Decision Processes"
IEEE Transactions on Systems, Man, and Cybernetics, vol SMC3, nº 1, January 1973.

Referências On-Line

[ALIVE SITE]
NEVES, ROGÉRIO P. O.
Página oficial do projeto A.L.I.V.E.
Disponível em: <http://www.lsi.usp.br/~alive/>, acesso em 1/9/2003
Pagina do responsável pelo projeto
Disponível em: <http://www.lsi.usp.br/~rponeves/research>, acesso em 1/9/2003

[ALIFEVIII SITE]
ARTIFICIAL LIFE VIII WEB SITE
The 8th International Conference on the Simulation and Synthesis of Living Systems
Disponível em:
<http://parallel.hpc.unsw.edu.au/complex/alife8/>, acesso em 15/8/2003 ???

[ARTLIFE SITE]
LOBO, MARCIO
Página do grupo de vida artificial ARTLIFE
Disponível em: <http://www.lsi.usp.br/~artlife>, acesso em 15/8/2003

[ADAMI SITE]
ADAMI, CHRISTOPH
"Cris Adami home page"
Disponível em: <http://www.krl.caltech.edu/~adami/>, acesso em 15/8/2003
"The Digital Life Laboratory"
Disponível em: <http://dllab.caltech.edu/>, acesso em 15/8/2003
"Avida Home Page"
Disponível em: <http://dllab.caltech.edu/avida/>, acesso em 15/8/2003

[BROOKS SITE]
BROOKS, RODNEY
Rodney Brooks home at MIT AI Lab
Disponível em: <http://www.ai.mit.edu/people/brooks/>, acesso em 15/8/2003

[CAVALHIERI SITE]
CAVALHIERI, MARCOS
Projeto com humanos virtuais
Disponível em: <http://www.lsi.usp.br/~mac/>, acesso em 1/9/2003

[CAVERNA SITE]
NÚCLEO DE REALIDADE VIRTUAL DO LSI-USP
"Página da CAVERNA digital"
Disponível em: <http://www.lsi.usp.br/~rv/>, acesso em 1/9/2003

[CESTA SITE]
CESTA, ANDRÉ AUGUSTO
Tutorial: "A Linguagem de Programação Java"
Disponível em: <http://www.dcc.unicamp.br/~cmrubira/aacesta/>, acesso em 24/8/2003

[DOLAN SITE]
DOLAN, ARIEL
Floyds - social, territorial artificial life creatures
Disponível em: <http://www.aridolan.com/>, acesso em 24/8/2003

[FONER SITE]
FONER, LENNY
"The Foner Agent"
Disponível em: <http://foner.www.media.mit.edu/people/foner/>,
acesso em 15/8/2003

[LOBONETT SITE]
NETTO, MÁRCIO LOBO
Página Pessoal
Disponível em: <http://www.lsi.usp.br/~lobonett>, acesso em 15/8/2003

[RAY SITE]
RAY, THOMAS
"Thomas Ray Home Page"
Disponível em: <http://www.isd.atr.co.jp/~ray/>, acesso em 15/8/2003
"Tierra Web Site"
Disponível em: <http://www.isd.atr.co.jp/~ray/tierra/>, acesso em 15/8/2003

[REYNOLDS SITE]
REYNOLDS, CRAIG
Craig Reynolds Home Page
Disponível em: <http://www.red3d.com/cwr/>, acesso em 15/8/2003

[RV SITE]
KIRNER, CLAUDIO
Tutorial "Sistemas de Realidade Virtual"
Disponível em:
<http://www.realidadevirtual.com.br/publicacoes/tutorial_rv/tutrv.htm>,
acesso em 15/8/2003

[SIMS SITE]
SIMS, KARL
"Karl Sims Work bio on Biota.org"
Disponível em: <http://www.biota.org/ksims/>, acesso em 15/8/2003

[SUN SITE]
SUN MICROSYSTEMS JAVA GROUP
"Sun's Java Site", "Java Online Manual", "Java API Specification"
Disponível em: <http://java.sun.com/>, acesso em 1/9/2003
"The Java3D Introduction", "Java3D API Specification", and more on-line material
Disponível em: <http://java.sun.com/products/java-media/3D/collateral/>,
acesso em 1/9/2003

[SWARM SITE]
SWARM DEVELOPMENT GROUP
Swarm.org
Disponível em: <http://www.swarm.org/>, acesso em 1/9/2003

Bibliografia recomendada

[BERG, 1983]
BERG, HOWARD C.
"Random Walks in Biology"
Princeton University press (1983)
ISBN: 0-691-08245-6

[BROOKS, 1991a]
BROOKS, RODNEY A.
"Intelligence without Representation"
Artificial Intelligence 47, (pags. 139-159) (1991)
Também disponível em:
<http://www.ai.mit.edu/people/brooks/papers/AIM-1293.pdf >, acesso em 15/8/2003

[BROOKS, 1991b]
BROOKS, RODNEY A.
"Intelligence Without Reason"
Proceedings of the 12th International Joint Conference on Artificial Intelligence ({IJCAI}-91),(A.I. Memo n. 1293), MIT Artificial Intelligence Laboratory
Morgan Kaufmann publishers Inc.: San Mateo, CA, USA (April, 1991)
ISBN: 1-55860-160-0
Também disponível em:
<http://www.ai.mit.edu/people/brooks/papers/representation.pdf >,
acesso em 15/8/2003

[COLOMBETTI, 1994]
COLOMBETTI, MARCO; DORIGO, MARCO
"Training Agents to Perform Sequential Behavior"
International Computer Science Institute (ICSI), Berkeley, CA
Adaptive Behavior v.2, n.3, pags. 247-275
MIT Press (1994)

[CRUZ-NEIRA, 1992]
CRUZ-NEIRA, C. ET AL.
"The CAVE Audio Visual Experience Automatic Virtual Environment"
Communication of the ACM, 35(6):64-72, June 1992

[LEJTER, 1996]
LEJTER, MOISES; DEAN, THOMAS
"A Framework for the Development of Multiagent Architectures"
IEEE Expert, pags. 47-61, December (1996)

[LEVY, 1992]
LEVY, STEVEN
"Artificial Life: The Quest for a New Creation"
Panteon Books (1992)
(Um dos mais completos sumários jornalísticos publicados até a data)
ASIN: 0-679-40774-X

[LEWONTIN, 1998]
LEWONTIN, RICHARD
"A Tripla Hélice: Gene, Organismo e Ambiente"
Editora Companhia das Letras (1998)
ISBN: 85-359-0259-7

[MIRANDA, 2001a]
MIRANDA, FABIO R. ET AL
"An Artificial Life Approach for the Animation of Cognitive Characters"
Computers&Graphics - an international journal on computer graphics and applications, Special Issue: Virtual Life – Towards New Generation of Computer Animation, volume 25, issue 6 (december 2001)
Elsevier Science

[MIRANDA, 2001b]
MIRANDA, FABIO R. ET AL
"Arena and WoxBOT: First Steps Towards Virtual World Simulations"
SIBGRAPI 2001 - XIV Brazilian Symposium on Computer Graphics and Image Processing, Florianópolis, Brazil (october 2001)
IEEE Computer Society Press

[MURPHY, 1995]
MURPHY, P, O'NEILL, L.
"O que é Vida? 50 anos depois"
Editora Unesp (1995)
ISBN: 85-7139-168-8

[NETTO, 2001]
NETTO, MARCIO L.; KOGLER JR., JOÃO E. (EDITORS)
Special Issue: "Virtual Life – Towards New Generation of Computer Animation"
Computers & Graphics – an international journal on computer graphics and applications, volume 25, issue 6 (december 2001)
Elsevier Science

[PONNAMPERUMA, 1972]
PONNAMPERUMA, CYRIL
"The Origins of Life"
Thames and Hudson Ltd; (1972)
ASIN: 0500100128

[SCHOEDER, 1997]
SCHOEDER, W.; MARTIN, K.; LORENSEN, B.
"The Visualization Toolkit"
Prentice Hall Computer Books (1997)
ISBN: 0-139-54694-4

[THAGART, 1998]
THAGART, PAUL
"MIND - Introduction to Cognitive Science"
MIT Press (1998)
ISBN: 0-262-20106-2

[WATT, 1997]
WATT, A.; POLICARPO, F.
"The Computer Image"
Addison Wesley (1997)
ISBN: 0-201-42298-0

[ZADEH, 1979]
ZADEH , LOTFI ASKER
"Linear System Theory"
Krieger Publishing Company; Reprint edition (February 1979)
ISBN: 0882758098

[ZADEH, 1999]
ZADEH , LOTFI ASKER
"Computing With Words in Information/Intelligent Systems 2: Applications"
Studies in Fuzziness and Soft Computing, Vol. 33-34
Springer Verlag; (November 15, 1999)
ISBN: 3790812188

APÊNDICE A. TABELAS E GRÁFICOS ADICIONAIS

A.1. Sistema Presa-predador

Seguem alguns gráficos de variação da população em função do tempo de experimento, obtidos nos experimentos realizados. A saída gerada pelo programa tem o formato mostrado na Tabela 5. Os símbolos da última coluna dão uma idéia dos processos ocorridos no ambiente através dos caracteres exibidos. Sempre que nasce um predador, o símbolo “O” é impresso, “+/-” indica o nascimento de uma presa com sinal de energia positivo ou negativo, o símbolo “x” é impresso toda vez que uma presa morre e “X” (maiúsculo) sempre que um predador morre.

Tabela 5. Exemplo da saída gerada pelo experimento.

Started: Thu Aug 21 15:25:52 BRT 2003

T	Hnters	Food	Green	Red	
0	90	900	597	303	xxxxxxxxxxxxxxx...
134	90	676	475	201	xxxxxxxxxxxxxxx...
268	90	605	429	176	xxxxxx+xxxxxx+x+xx+xxxx+xxxxx
2680	96	102	65	37	xxx+x+x+x+x+xx+xxxxx
2814	96	94	60	34	+x+xXo+x+++x+xxxxx+xxox+x
2948	97	90	57	33	xxXxx+xx+
3082	96	86	54	32	+xxxxXxx+x
3216	95	81	51	30	+xo+xx+xx+x
3350	96	79	50	29	x+++xxxx++x+
3484	96	79	50	29	xxxx+++-+x+-o
3618	97	81	52	29	xxx++x++x
3752	97	80	51	29	x-x+X+x-+x
3886	96	81	51	30	x-xx+x+xxxxx-Xx
4020	95	75	44	31	xX++-X+xx
4154	93	76	44	32	+xX++xxx+++x++
4690	91	75	41	34	ox-xxx+x
4824	92	72	38	34	xxxxx
4958	92	67	34	33	x++x--xX+
5092	91	69	34	35	x-xxxx+
5226	91	66	32	34	++xx+X+
5360	90	68	34	34	++++xxx--x
5494	90	70	34	36	+x+x+x
5628	90	70	35	35	xoxxXX+xx
5762	89	66	32	34	+++++xXx+xx-x++x
5896	88	69	35	34	xxxx+ox+
6030	89	66	32	34	Xx+x+X+xxx
6298	87	62	28	34	x+x+Xx
6432	86	61	28	33	x-x-+x+xxxx
6566	86	58	23	35	++++--x
6700	86	63	26	37	Xx+X+--+x-xxx
6834	84	64	24	40	+++X
7102	83	66	26	40	++x+
7236	83	68	28	40	---+
7504	83	76	30	46	--+--+
7638	83	82	32	50	++++++x++++
7772	83	91	41	50	x--+X+x--+xx
7906	82	94	40	54	x--+
8040	82	96	40	56	X++
8442	81	110	52	58	+++++++
8576	81	117	59	58	x++++++-+
8710	81	124	65	59	+XX+++X+
14472	66	406	230	176	xoxxx+x+

Finished: Thu Aug 21 15:29:01 BRT 2003

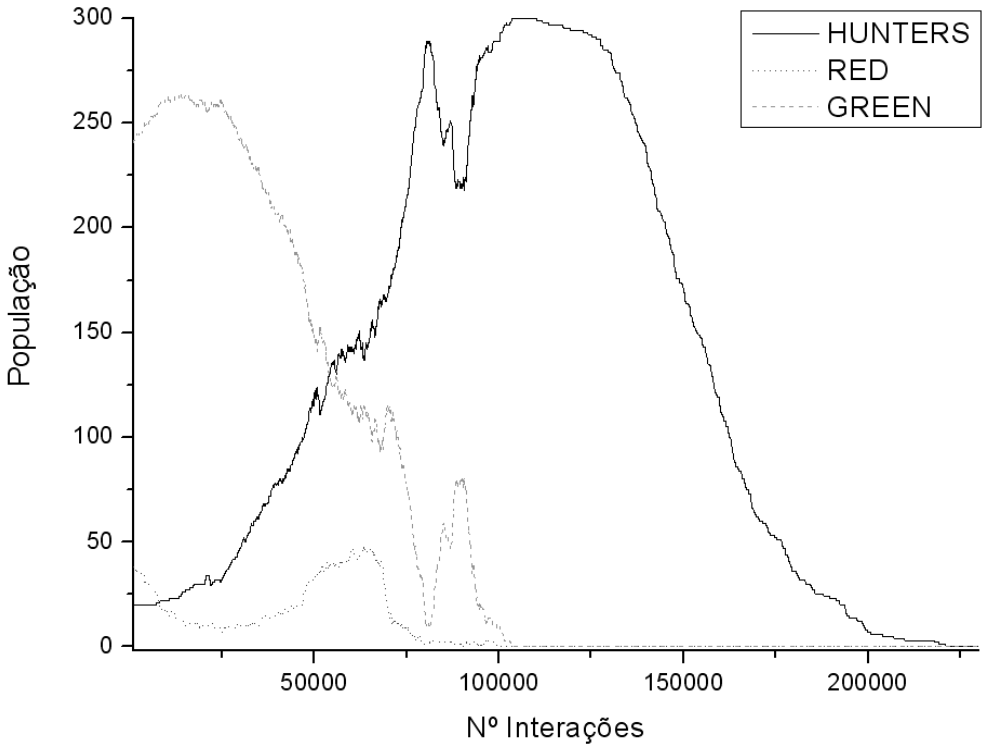

Figura 39. Extinção devido ao consumo total dos recursos.

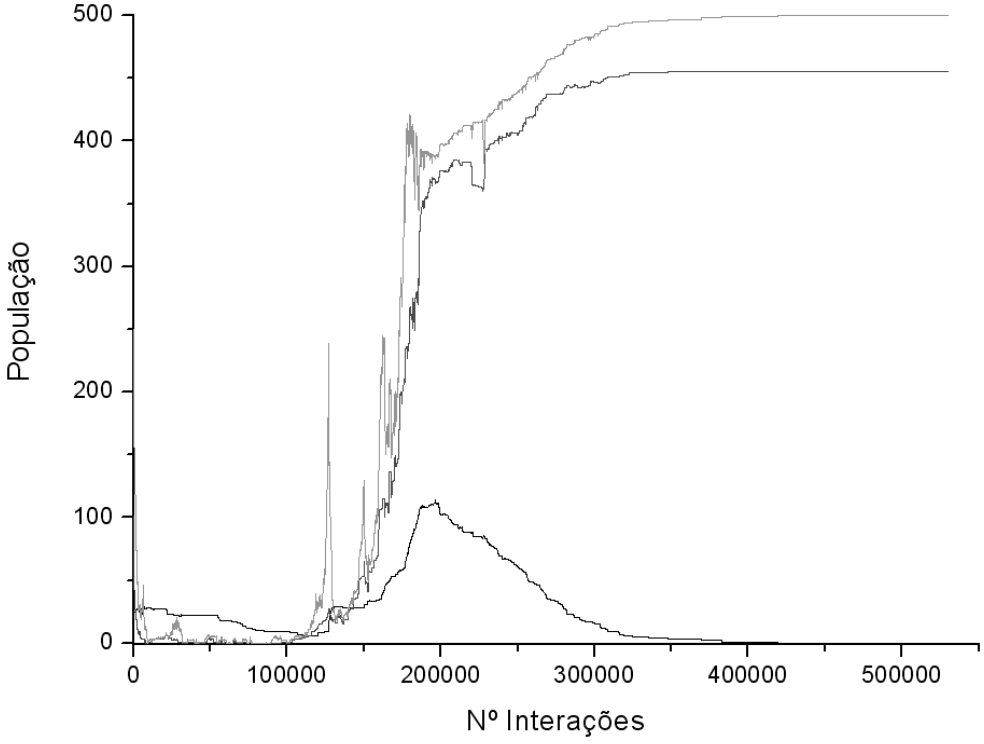

Figura 40. Extinção devido ao acumulo de toxinas no ambiente.

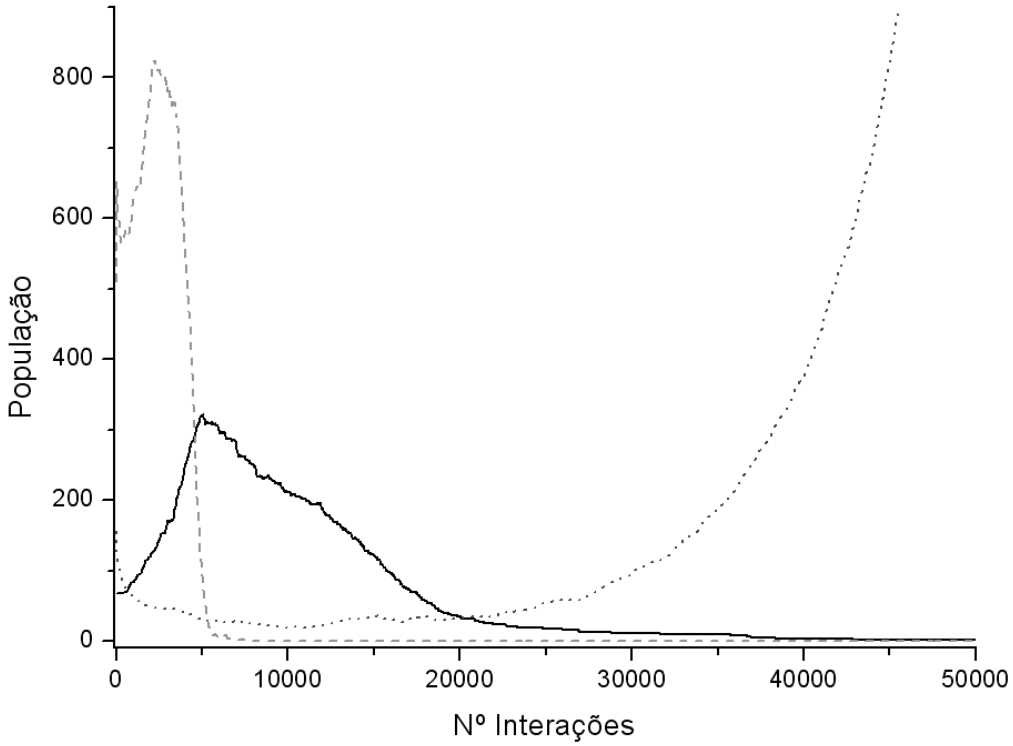

Figura 41. Extinção devido ao acumulo de toxinas no ambiente.

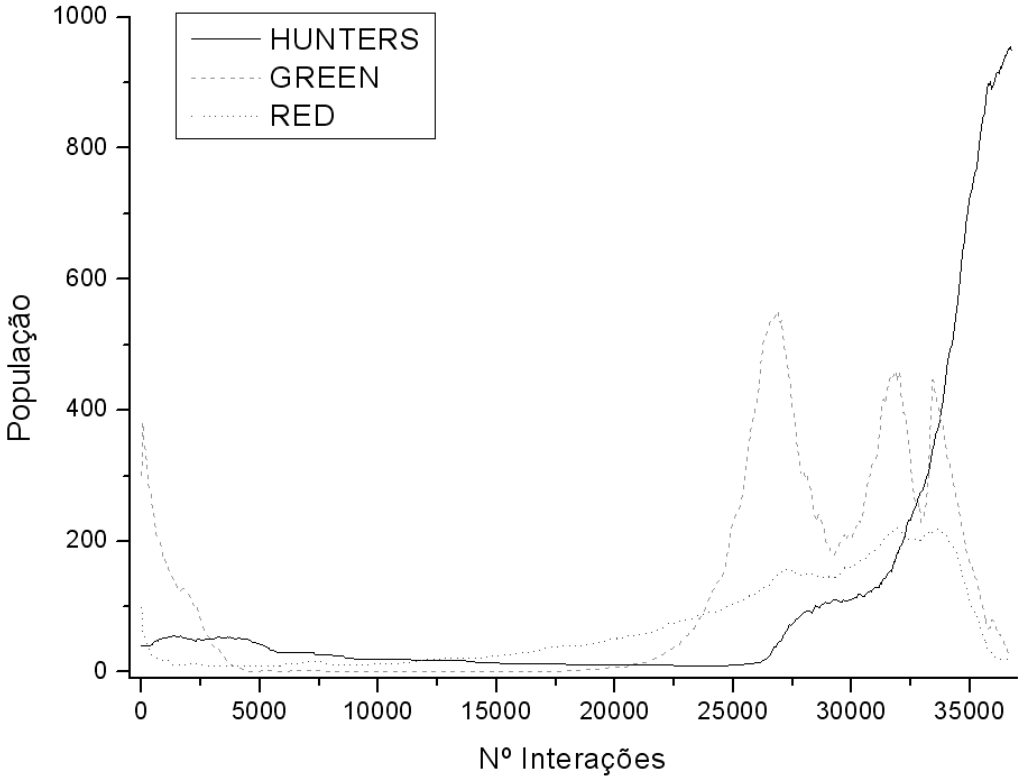

Figura 42. Explosão populacional devido a abundância de alimentos.

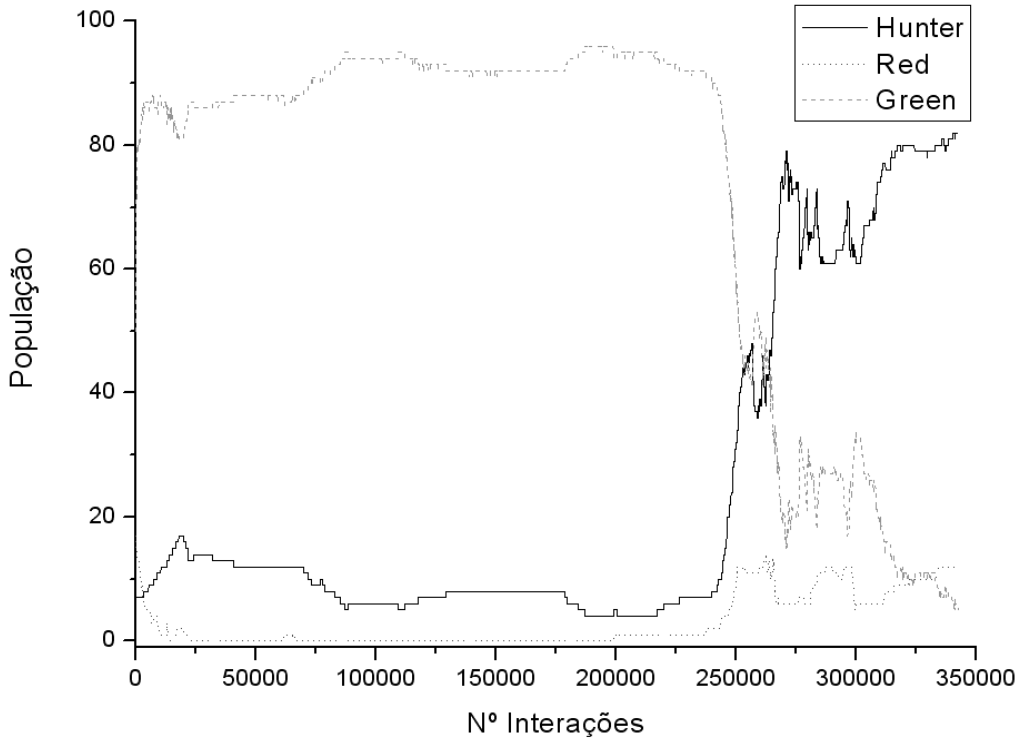

Figura 43. Figura mostra um salto evolutivo (~24000) para um sistema sem abastecimento.

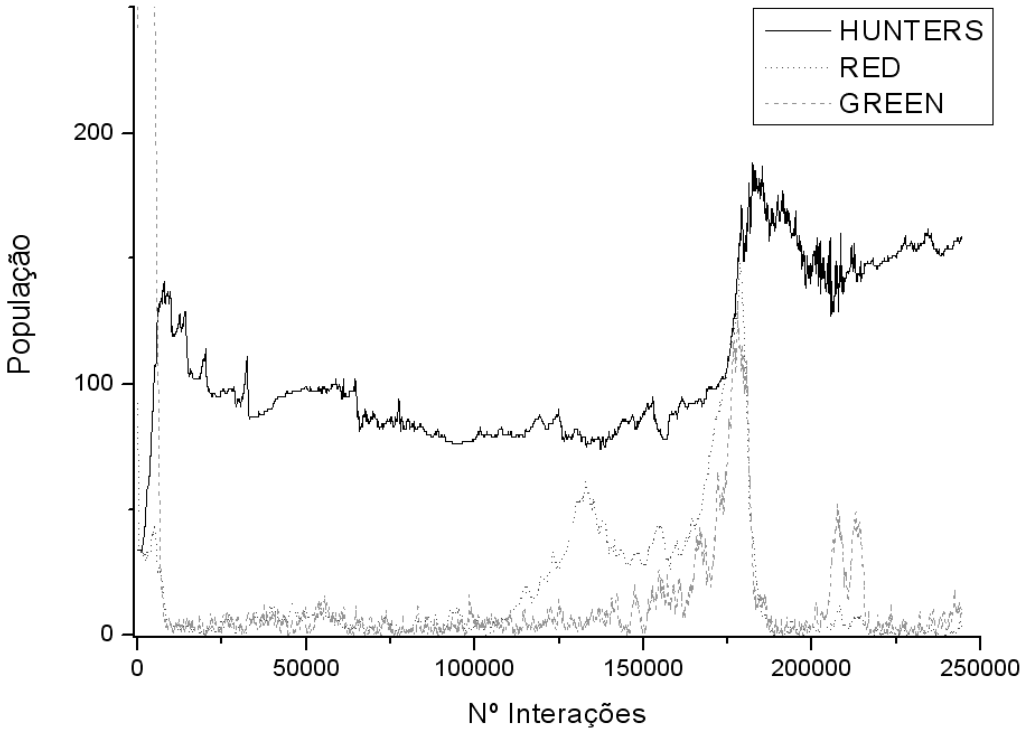

Figura 44. Para um sistema abastecido de presas, Gráfico da variação populacional com o tempo no caso de equilíbrio entre presa e predador. A região em torno de 17500 mostra o crescimento populacional devido a um salto evolutivo.

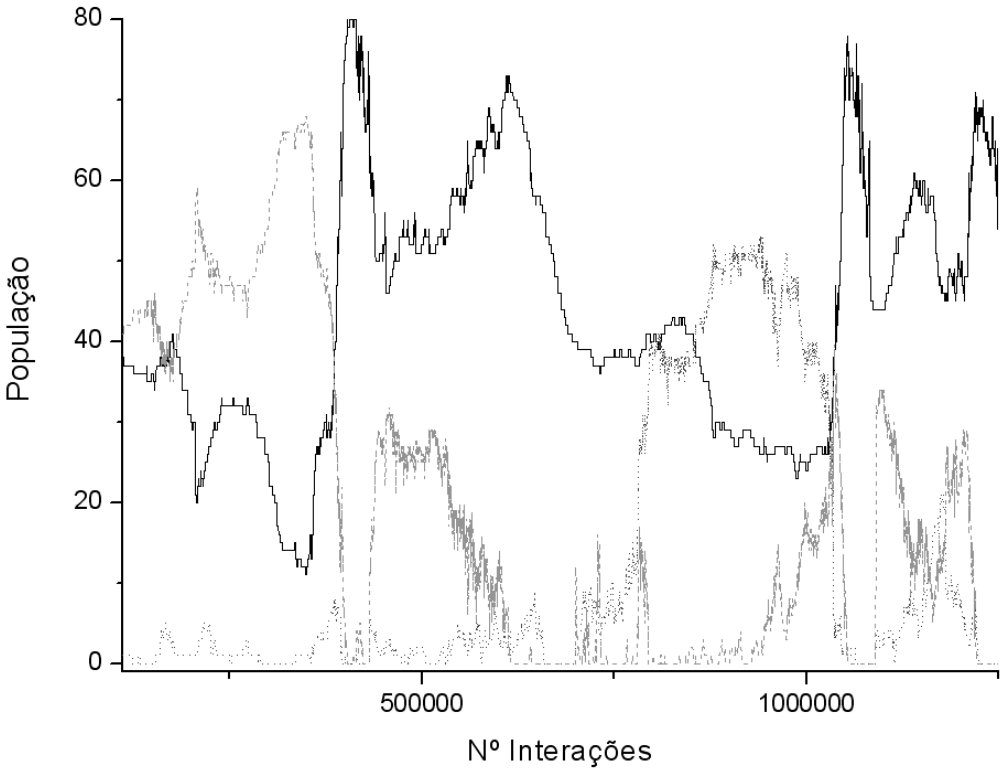

Figura 45. Variação das três populações em um experimento típico, com abastecimento automático não descriminado entre presas do tipo +/- (alimentos e toxinas).

APÊNDICE B. ESPECIFICAÇÃO DAS SUPERCLASSES

Aqui são apresentados os diagramas estruturais das classes base do projeto. Diagramas estruturais em notação UML consistem de blocos descritivos das classes e objetos que compõe o projeto. No caso de um diagrama de classe na notação UML, o formato é apresentado como segue:

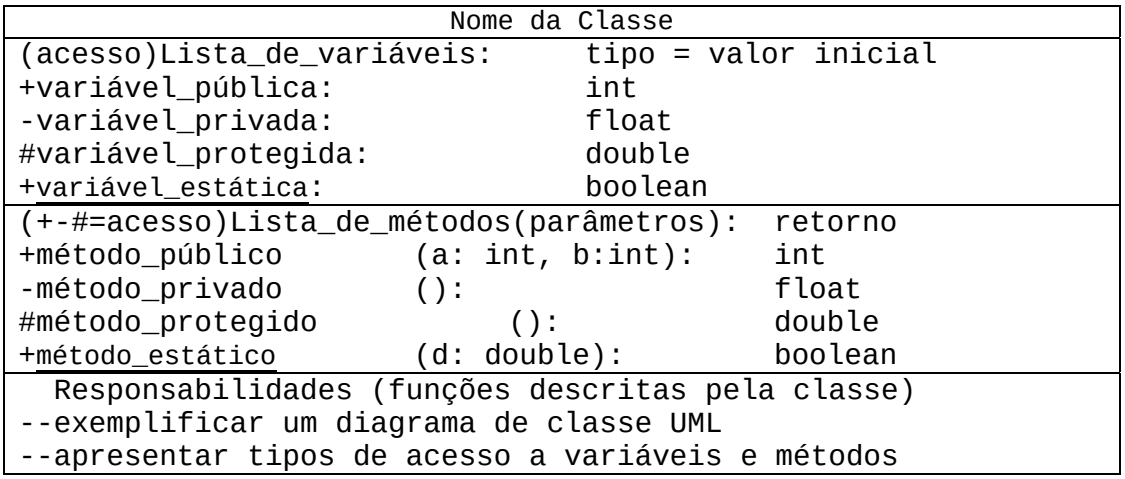

Figura 46. Exemplo de diagrama de classe na notação UML.

Os diagramas apresentados aqui exibem apenas métodos e variáveis públicas e protegidas, ocultando os pertencentes aos mecanismos internos de funcionamento da plataforma.

B.1. Superclasse Ambiente

A Superclasse Ambiente (**Environment.class**) contém os métodos de visualização e controle do ambiente. O diagrama de classe é apresentado no tópico a seguir. As variáveis públicas disponíveis são explicadas posteriormente. Em seguida, as substituições que devem ser efetuadas pelo código implementado pelo usuário são apresentadas juntamente com a utilização dos demais métodos.

B.1.1. Diagrama de Classe

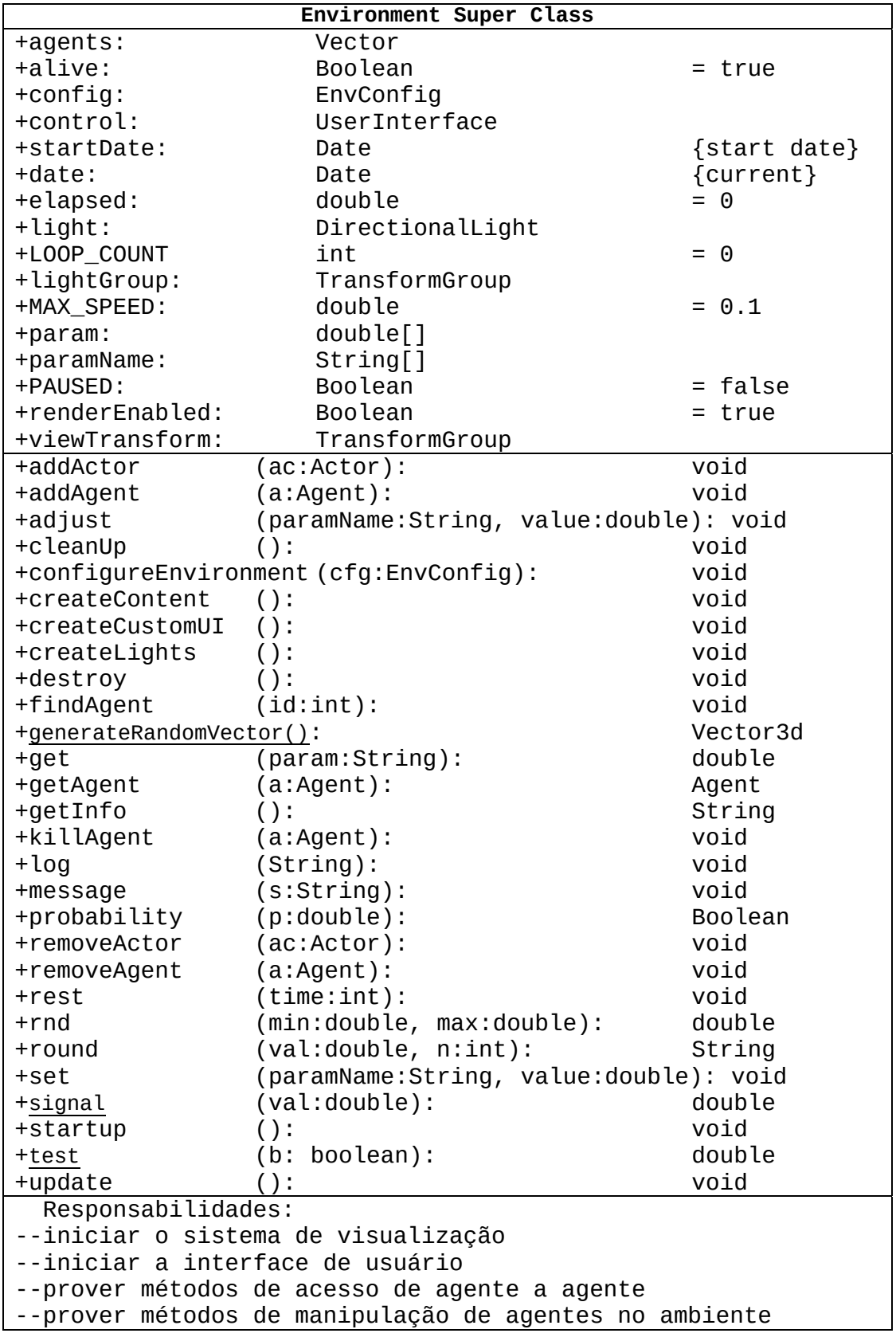

Figura 47. Diagrama de classe da superclasse ambiente.

B.1.2. Descrição das Variáveis:

`agents: Vector`

Lista contendo referências para todos os agentes em execução no ambiente, por ordem de criação.

`alive: Boolean`

Atividade do ambiente. Se Verdadeiro, o ambiente está executando; se falso, a simulação terminou.

`config: EnvConfig`

Referência ao objeto de configuração do ambiente que contém a especificação do ambiente simulado.

`control: UserInterface`

Referência ao objeto de interface de usuário.

`startDate: Date`

Data e hora do início da execução da simulação.

`date: Date`

Data e hora atual.

`elapsed: double`

Número de ciclos (loops) executados desde o início da simulação.

`light: DirectionalLight`

Referência a iluminação padrão. Permite mudar atributos da iluminação.

`MAX_SPEED: double`

Velocidade máxima que objetos podem atingir no ambiente virtual. O valor deve ser diminuído para evitar distorções e aberrações no cálculo das interações.

`paramName: String[]`

Vetor contendo a lista com os nomes dos parâmetros do ambiente. As variáveis podem representar, por exemplo, “temperatura”, “viscosidade” ou concentrações químicas específicas.

`param: double[]`

Vetor contendo os valores dos parâmetros especificados pela lista acima.

`PAUSED: Boolean`

Estado da simulação. Verdadeiro = simulação parada.

`renderEnabled: Boolean`

Visualização habilitada. Falso indica sem saída visual.

`viewTransform: TransformGroup`

Grupo raiz dos objetos de visualização contendo posição e orientação das câmeras. Pode ser operado pelo código de usuário para acompanhar um determinado ator.

B.1.3. Métodos a Serem Substituídos por Código do Usuário*

(**Override required*)

`createContent(): void`

Método que especifica os objetos e atores, que são adicionados à simulação no início do experimento.

`update(): void`

O método define as regras de atualização para o ambiente especificado em tempo de execução.

B.1.4. Métodos que Podem ser Substituídos por Código do Usuário**

(***Override allowed*)

`cleanUp(): void`
O método é executado quando a simulação termina.

`createCustomUI(): void`
Método que adiciona uma interface de controle para parâmetros do experimento no espaço reservado da interface de controle.

`createLights(): void`
Método que adiciona iluminação à cena virtual. No método padrão, apenas uma luz direcional é criada.

`destroy(): void`
Método padrão para limpar as variáveis de ambiente.

`getInfo():String`
Retorna informações sobre os parâmetros da simulação.

`startUp(): void`
O método é executado quando a simulação começa.

B.1.5. Métodos Finais de Acesso e Controle***

(****Final methods: Override forbiden*)

`addActor(Actor): void`
Adiciona ator à cena virtual.

`addAgent(Agent):`
Adiciona o agente ao ambiente de simulação.

`adjust(paramName: String, value: double): void`
Ajusta o parâmetro especificado com o valor fornecido.

`configureEnvironment(cfg: EnvConfig): void`
Configura o ambiente inicial utilizando o objeto de configuração cfg.

`findAgent(id: int): void`
Executa busca de agente por número de identificação.

`get(param: String): double`
Retorna o valor do parâmetro especificado.

`getAgent(pos: int): Agent`
Retorna o agente na posição pos do vetor de agentes.

`killAgent(pos: int): void`
Finaliza o agente na posição pos do vetor de agentes.

`log(String): void`
Grava no arquivo de registro a mensagem passada.

`message(String): void`
Exibe na saída padrão a mensagem passada.

`removeActor(Actor): void`
Remove o ator especificado da cena virtual.

`removeAgent(Agent): void`
Remove o agente especificado da simulação.

`set(paramName: String, value: double): void`
Define valor para o parâmetro com nome especificado.

`set(n:int, paramName: String, value: double): void`
Define valor e nome para o parâmetro com o número especificado.

B.1.6. Métodos de Auxílio

(Utilitários ou *Helper methods*)

`generateRandomVector(): Vector3d`
Gera um vetor aleatório normalizado.

`probability(P: double): Boolean`
Retorna verdadeiro, com probabilidade P (entre 0 e 1).

`rest(tempo: int): void`
Permanece inativo pelo tempo (ms) especificado.

`rnd(min:double, max: double): double`
Retorna um valor aleatório no intervalo determinado pelo máximo e mínimo.

`rnd(min:int, max: int): double`
Retorna um valor aleatório no intervalo determinado pelo máximo e mínimo.

`round(val:double, n:int): String`
Retorna uma String contendo o valor arredondado com n casas decimais.

`signal(var: double): double`
Retorna +1 se positivo ou -1 se negativo.

`test(test: boolean): double`
Retorna 1d se verdadeiro ou 0d se falso.

B.2. Superclasse Agente

A superclasse Agente (**Agent.class**) é utilizada na criação de qualquer agente desenvolvido no contexto da plataforma. Seu funcionamento é similar à superclasse ambiente, exceto por possuir métodos abstratos que necessitam substituição na classe

filha. O diagrama, descrição das variáveis e utilização dos métodos é apresentado a seguir.

B.2.1. Diagrama de Classe

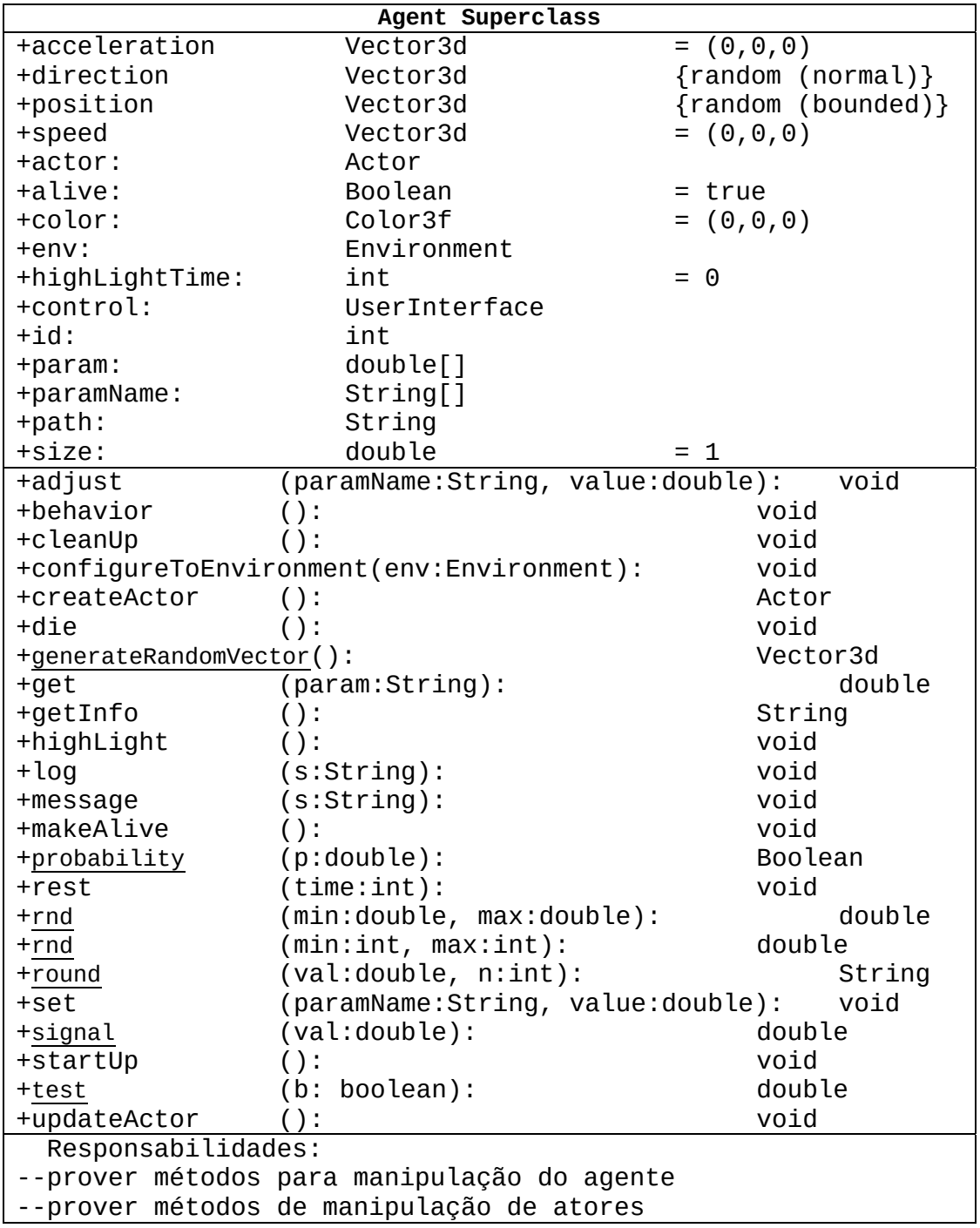

Figura 48. Diagrama de classe da superclasse Agente.

B.2.2. Descrição das Variáveis

`actor: Actor`
Referência para objeto ator que representa o agente na cena.

`alive: Boolean`
Atividade do agente. Se verdadeiro, o agente está executando; se falso, o agente encerrou a execução.

`color: Color3f`
Cor do agente, para uso público ou em conjunto com atributos visuais do ator.

`highlightTime: int`
Marca o tempo em que o objeto (ator) fica em evidência, quando o agente é selecionado.

`id: int`
Número único de identificação do agente, atribuído seqüencialmente por ordem de criação.

`paramName: String[]`
Vetor contendo a lista com os nomes dos parâmetros do agente. As variáveis representam atributos próprios do agente, como cromossomos, temperatura ou concentrações químicas específicas.

`param: double[]`
Vetor contendo os valores dos parâmetros especificados pela lista acima.

`path: String`
Caminho para acesso aos arquivos do pacote.

`size: double`
tamanho do agente.

`acceleration: Vector3d`
Vetor aceleração do agente.

`direction: Vector3d`
Vetor direção do agente.

`position: Vector3d`
Vetor indicando a posição atual do agente.

`speed: Vector3d`
Vetor velocidade do agente.

B.2.3. Métodos a Serem Substituídos por Código do Usuário*

(**Abstract methods: Override required*)

`behavior(): void`
Método que contém as operações feitas com o agente em tempo de execução.

`createActor(): Actor`
O método deve retornar um ator válido para representar o agente na cena, não sendo executado se o sinalizador (*flag*) `env.config.RENDER` estiver desabilitado.

`updateActor(): void`
Opera alterações visuais no ator em tempo de execução. O método não é executado se qualquer um dos sinalizadores (*flags*) de síntese de imagens, `env.renderEnabled` ou `env.config.RENDER`, estiverem desabilitados.

B.2.4. Métodos que Podem ser Substituídos por Código do Usuário**

(***Override allowed*)

`cleanUp(): void`
O método é executado quando a simulação termina.

`die(): void`
O método destrói as variáveis criadas pela instância, remove o ator da cena e o agente do ambiente.

`getInfo():String`
Retorna informações sobre os parâmetros e vetores do agente.

`startUp(): void`
O método é executado quando a simulação começa.

B.2.5. Métodos de Acesso e Controle***

(****Final: Override forbiden*)

`adjust(paramName: String, value: double): void`
Ajusta o parâmetro especificado com o valor fornecido.

`configureToEnvironment(env: Environment): void`
Configura o agente para o ambiente fornecido.

`get(param: String): double`
Retorna o valor do parâmetro especificado.

`highLight(): void`
Marca o ator com alto contraste por um intervalo de tempo determinado.

`log(String): void`
Grava no arquivo de registro a mensagem passada.

`message(String): void`
Exibe a mensagem passada no console.

`makeAlive(): void`
Inicia a execução do agente.

`removeAgent(Agent): void`
Remove o agente especificado da simulação.

`set(paramName: String, value: double): void`
Define o valor para o parâmetro com nome especificado.

`set(n:int, paramName: String, value: double): void`
Define o valor e nome para o parâmetro com o índice especificado.

B.2.6. Métodos de Auxílio

(Utilitários ou *Helper Methods*)

`generateRandomVector(): Vector3d`
Gera um vetor aleatório normalizado.

`probability(P:double): Boolean`
Retorna verdadeiro, com probabilidade P (entre 0 e 1).

`rest(tempo:int): void`
Permanece inativo pelo tempo (ms) especificado.

`rnd(min:double, max:double): double`
Retorna um valor aleatório no intervalo especificado.

`rnd(min:int, max:int): double`
Retorna um valor aleatório no intervalo especificado.

`round(val:double, n:int): String`
Retorna uma String contendo o valor arredondado com n casas decimais.

`signal(var: double): double`
Retorna +1 se positivo ou -1 se negativo.min-max.

test(b: boolean): double

Retorna 1 se verdadeiro ou 0 se falso.

B.3. Superclasse Ator

A superclasse ator é responsável pela padronização de acesso e operação dos atores criados no contexto da plataforma.

B.3.1. Diagrama de Classe

Actor Superclass		
+emissiveColor:	Color3f	= (0,0,0)
+ambientColor:	Color3f	= (.6,.6,.6)
+diffuseColor:	Color3f	= (.6,.6,.6)
+specularColor:	Color3f	= (1,1,1)
+shiness:	float	= 100
+id:	int	
+model:	String	""
+scale:	double	= 0.01
+texturefile:	String	""
#tg:	TransformGroup	
#dir	Vector3d	= (0,0,0)
#pos	Vector3d	= (0,0,0)
+black:	Color3f	= (0,0,0)
+blue:	Color3f	= (0,0,1)
+cyan:	Color3f	= (0,1,1)
+green:	Color3f	= (0,1,0)
+magenta:	Color3f	= (1,0,1)
+red:	Color3f	= (1,0,0)
+white:	Color3f	= (1,1,1)
+yellow:	Color3f	= (1,1,0)
+build	():	void
+createGeometry	():	void
+getColor	():	Color3f
+getMaterial	():	Material
+getTransform	():	Transform3D
+getTransformGroup	(): TransformGroup	
+set	(pos:Vector3d, size:double):	double
+set	(pos, dir, size):	String
+setAllColors	():	void
+setColor	(r:float, g:float, b:float):	void
+setColor	(c:Color3f):	void
+setTexture	(filename:String):	void
+signal	(val:double):	double
Responsabilidades:		
--prover métodos para manipulação de atores		

Figura 49. Diagrama de classe da superclasse Ator.

B.3.2. Descrição das Variáveis

`emissiveColor: Color3f`
Cor emissiva (própria) do material de que constitui o ator.

`ambientColor: Color3f`
Cor ambiente (uniforme) do material de que constitui o ator.

`diffuseColor: Color3f`
Cor difusa (fosca) do material de que constitui o ator.

`specularColor: Color3f`
Cor especular (lisa) do material de que constitui o ator.

`shiness: float`
Brilho do material de que constitui o ator.

`id: int`
Número único de identificação do agente proprietário.

`model: String`
String descrevendo o modelo utilizado para o ator, para uso em síntese distribuída.

`scale: double`
Escala do objeto ou ator.

`texturefile: String`
String com o nome do arquivo de imagem utilizado na textura, para síntese distribuída.

`tg: TransformGroup`
Grupo princcipal de transformação que opera a geometria do ator.

`dir: Vector3d`
Vetor direção do ator.

`pos: Vector3d`
Vetor contendo a posição do ator na cena.

`+black, blue, cyan, green, magenta, red, white, yellow: Color3f`
Cores pré-definidas para uso dos agentes e atores.

B.3.3. Métodos a Serem Substituídos por Código do Usuário*

(**Override required*)

`createGeometry(): void`
O método deve adicionar uma forma geométrica ao grupo principal de transformação `tg`.

B.3.4. Métodos que Podem ser Substituídos por Código do Usuário**

(***Override allowed*)

`set(position:Vector3d, scale:double): void`
Define somente posição e escala do ator na cena.

`set(position:Vector3d, direction:Vector3d, scale:double): void`
Define posição, direção e escala do ator na cena.

B.3.5. Métodos de Acesso e Controle***

(****Override forbiden*)

`build(): void`
Constrói o ator com base na geometria contida no método `createGeometry()`.

`getColor(): Color3f`
Retorna a cor emissiva do ator.

`getMaterial(): Material`

Retorna uma referência para o objeto de material utilizado no ator.

`getTransform(): Transform3D`

Retorna a matriz de transformação do ator.

`getTransformGroup(): TransformGroup`

Retorna uma referência ao nó de transformação do ator.

`setColor(r:float, g:float, b:float): void`

Define a cor emissiva do ator, com base nos valores RGB passados.

`setColor(c:Color3f): void`

Estabelece a cor emissiva do ator com as componentes do objeto passado.

`setAllColors():void`

Atualiza as referências do objeto de material para o conjunto de cores e brilho determinados pelas variáveis internas do ator.

`setTexture(filename: String): void`

Define a figura informada pelo nome do arquivo para ser usada como textura no ator.

`setTransparency(transparency: float): void`

Define a transparência do ator para o valor passado, que deve estar entre 0 e 1.

APÊNDICE C. CONCEITOS DE PROGRAMAÇÃO

C.1. Conceitos Fundamentais da Linguagem Java

Para utilizar a plataforma, o usuário deverá escrever algum código, e, portanto, deverá possuir algum conhecimento da linguagem Java. Um conhecimento elementar do paradigma de POO é requerido, devendo o leitor recorrer a qualquer das literaturas supracitadas para dados mais detalhados sobre tal paradigma, bem como sobre a linguagem Java e sua utilização. Aqui são descritas apenas algumas características essenciais da linguagem Java, necessárias e suficientes para o entendimento dos segmentos de código que são apresentados nas próximas seções.

A linguagem Java é estruturalmente similar a suas linguagens precursoras, C e C++. Nos programas desenvolvidos em Java, C e C++, cada expressão (*statement*) é separada por um ponto-e-vírgula.

Os blocos de códigos são separados por chaves “{}”, que definem um conjunto de expressões pertencentes a um mesmo escopo. Expressões conectadas do tipo se... então, podem ser ligadas em uma só expressão, dispensando o ponto-e-vírgula e a chave, desde que apenas uma avaliação ou operação seja efetuada.

```
/* Comentário
   código exemplo */
public void método(boolean booleanFlag, int soma) {
   // Escopo do método
   double valorDouble = (double)soma; // type cast
   if (booleanFlag) soma+=1; // 1 expressão condicional
   if (soma==20) { // 2 expressões condicionais
       // Escopo da expressão condicional
       soma=0;
       booleanFlag=false;
   }   // Fim do escopo da expressão condicional
}  // Fim do escopo do método
```

Os métodos são similares a funções em linguagens funcionais como Pascal e Fortran, tendo seus nomes precedidos pelo tipo de acesso, este podendo ser público, privado

ou protegido, e o tipo de variável de retorno, “void” indicando sem retorno. Após a declaração do nome do método, entre parênteses, estão as variáveis passadas como parâmetros, separadas por vírgulas e precedidas pela especificação do seu tipo.

O texto precedido por “//” ou entre os símbolos “/*” e “*/” são considerados comentários e não são processados pelo compilador.

Os tipos de acesso às classes e métodos são: público (*public*), privado (*private*) e protegido (*protected*), como explicado no capítulo anterior. Para se acessar um método em um objeto diferente do local (*this*) deve-se especificar o objeto referenciado seguido de um ponto e do método evocado no objeto referido. Para indicar métodos e variáveis locais, ou seja, que pertencem ao próprio objeto operado, a palavra reservada “*this*” pode ser utilizada para indicar localidade, embora seja desnecessária se o método ou variável local possuir escopo global e não houver variáveis de escopo local que a ocultem.

```
public void run() {
  // Chama condicionalmente um método no objeto number
  if (booleanFlag) number.add(5); // parâmetro = 5

  // Chama um método local
  método(); // O mesmo que this.método()

  // Chama um método alheio
  outroObjeto.método();
}
```

Os tipos de variáveis em Java são:

Tabela 6. Tipos de variáveis na linguagem Java

boolean	Variável binária com valores verdadeiro ou falso (*true*, *false*), diferente da representação em C++, não possui conversão para outros tipos. Representa resultados de testes lógicos
byte	Inteiro de 8-bits , complemento de 2, assume valores de -128 a 127
short	Inteiro "curto" de 16-bits complemento de 2, assume valores de -32768 a 32767
int	Inteiro de 32-bits complemento de 2, assume valores de -2147483648 a 2147483647
long	Inteiro "longo" de 64-bits complemento de 2, assume valores de -9223372036854775808 a 9223372036854775807
float	Ponto-flutuante de 32-bits (IEEE754), representa números com precisão de (+/-)1.4E-45 a (+/-)3.4028235E38
double	Ponto-flutuante de 64-bits (IEEE754), representa números com precisão de (+/-)4.9E-324 a (+/-)1.7976931348623157E308

Não existem especificadores de tipos "*unsigned*", como na linguagem C e C++, todos os tipos inteiros possuem sinal. Um valor especificado sem ponto é considerado "*int*" como padrão, para atribuir esta constante a um "*long*", basta fazer um "*type cast*":

```
double valorDouble = 718123.1231;
long number;
number=(long)valorDouble; // (type cast)
//number = 718123;
```

As iniciais dos tipos inteiros e de ponto-flutuante também podem ser utilizadas como terminação para indicar o tipo do número especificado. Um valor contendo ponto, do tipo 3.14159, é considerado "*double*" como padrão, para atribuir esta constante a um "*float*", basta fazer um "*type cast*" ou utilizar a terminação "*f*" após o número.

```
double valorDouble = 718123.1231;
long valorLong = 123456L;
float valorFloat = 11.11f;
float valorFloat2 = (float)valorDouble;
```

Para indicar valores octais basta antecedê-los com um zero (0421), e para valores hexadecimais, com um 0X ou 0x. Exemplo: 0xBCD4.

Uma introdução à linguagem de programação Java em português pode ser obtida gratuitamente pela Internet em [CESTA SITE].

C.2. Utilização do API Java3D

O API Java3D fornece classes de manipulação de figuras geométricas, vetores, objetos compostos e áudio espacial, através de três pacotes incluídos na sua instalação:

- javax.media.j3d.*
- javax.vecmath.*
- com.sun.j3d.utils.*.

Para utilizar em um código, por exemplo, o pacote de cálculo vetorial, deve-se incluir o pacote no início do programa.

```
package meu_projeto

import java.applet.Applet;
import javax.media.j3d.*;
import javax.vecmath.*

public class MinhaClasse {
  //  Meu código
}
```

No contexto da plataforma, são regularmente utilizados cálculos vetoriais para a manipulação dos objetos e agentes. O acesso direto a objetos estruturais do Java3D foi mascarado por métodos de mais alto nível que visam facilitar a manipulação dos objetos na cena virtual. Para operar a posição de um agente, sua velocidade e aceleração, basta modificar os vetores correspondentes no objeto.

Tabela 7. Operações com vetores em Java3D com o pacote vecmath.

void absolute()	Deixa todos os componentes do vetor "*this*" absolutos
double angle(*Vector3d v1*)	Retorna o ângulo em radianos entre o vetor "*this*" e v1
void add(*v1*)	Adiciona v1 a "*this*"
void add(*v1, v2*)	Armazena em "*this*" a soma de v1 e v2
void cross(*v1, v2*)	Armazena em "*this*" o produto vetorial de v1 e v2
double dot(*v1*)	Retorna o produto escalar de "*this*" e v1
boolean equals(*v1*)	Retorna verdadeiro se "*this*" e v1 forem iguais
double length()	Retorna o comprimento do vetor "*this*"
double lengthSquared()	Retorna o comprimento ao quadrado do vetor "*this*"
void negate()	Nega o valor do vetor "*this*"
void negate(*v1*)	Nega o valor de v1 e armazena em "*this*"
void normalize()	Normaliza o vetor
void normalize(*v1*)	Normaliza o vetor v1 e armazena em "*this*"
void scale(*double s*)	Multiplica o vetor "*this*" pelo escalar s
void scale(*double s, v1*)	Multiplica o vetor v1 por s e armazena em "*this*"
void scaleAdd(*s, v1*)	"*this*" = s * "*this*" + v1
void scaleAdd(*s, v1, v2*)	"*this*" = s * v1 + v2
void set(*x, y, z*)	Define "*this*" para os valores double x, y e z
void set(*v1*)	Define a coordenada do vetor "*this*" a mesma de v1
void sub(*v1*)	Subtrai v1, "*this*" = "*this*" - v1
void sub(*v1, v2*)	Subtrai v2 de v1 e armazena em "*this*" = v1 – v2
String toString()	Retorna uma string do tipo "(x, y, z)"

A Tabela 7 sintetiza a utilização da classe Vector3d do pacote javax.vecmath. Nela, os objetos *vn* (*v1*, *v2*) pertencem a classe *Vector3d*, e *s*, *x*, *y* e *z* são variáveis numéricas do tipo *double*.

Para realizar operações mais complexas, que requerem modificação estrutural das formas em tempo de execução ou construção de atores e representações dinâmicas de agentes, o usuário deve recorrer à especificação da plataforma Java3D e sua documentação, disponível gratuitamente em [SUN SITE].