BRUNO VIEIRA GUERRA ALVES
TÁCIO SILVA DIOGO

Desenvolvimento de um sistema multiagente em NetLogo

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do título de Bacharel em Ciência da Computação do curso de Ciência da Computação da Universidade Federal Fluminense.

Orientador: ANA CRISTINA BICHARRA GARCIA

Niterói
2010

BRUNO VIEIRA GUERRA ALVES
TÁCIO SILVA DIOGO

Desenvolvimento de um sistema multiagente em NetLogo

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do título de Bacharel em Ciência da Computação do curso de Ciência da Computação da Universidade Federal Fluminense.

BANCA EXAMINADORA

Profª. Drª. Ana Cristina Bicharra Garcia – Orientadora
UFF

Prof. Dr. Inhaúma Neves Ferraz
UFF

Profª. Drª. Viviane Silva
UFF

Niterói
2010

Agradecimentos

A todos os professores e seus convidados pelo carinho, dedicação e entusiasmo demonstrado ao longo do curso.

Aos colegas de classe pela espontaneidade e alegria na troca de informações e materiais numa rara demonstração de amizade e solidariedade.

E, finalmente, às nossas famílias pela paciência em tolerar a nossa ausência.

Resumo

Sistemas multiagentes são parte de uma tecnologia relativamente nova da inteligência artificial. Existem diversas características e obstáculos que influenciam no desenvolvimento tanto de agentes como de ambientes e simuladores. Neste trabalho nós apresentamos uma visão geral do campo e das propriedades de sistemas multiagentes, assim como analisamos uma ferramenta específica, o NetLogo, para a criação e melhoria de um ambiente e dos agentes em um problema de fuga, e mostramos análises de desempenho.

Palavra chave: Sistemas multiagentes, simulação, NetLogo, inteligência artificial.

Abstract

Multiagent systems are part of a relatively new technology of artificial intelligence. There are numerous features and obstacles that influence the development of agents, environments and simulators. In this paper we introduce an overview of the field and properties of multiagent systems, as well as analyze a specific tool, NetLogo, for the creation of an environment and agents in an escape problem, and show performance analysis.

Keywords: Multiagent systems, simulation, Netlogo, artificial intelligence.

Sumário

Lista de Figuras

1. Introdução

Nos últimos anos diversas pesquisas foram feitas na área de sistemas multiagentes, nos campos de governança, sistemas de reputação, planejamento, raciocínio e adaptação. Estes sistemas ainda são parte de uma tecnologia relativamente nova no campo da inteligência artificial e suas técnicas podem ser utilizadas como auxílio para a solução de diversos tipos de problemas das mais diversas áreas [Russel, S., 2003].

Neste trabalho mostraremos uma ferramenta de criação de ambientes de simulação e agentes chamada NetLogo. Esta ferramenta pode ser utilizada para a criação de simulações nas mais diversas áreas, desde biologia, química, física até mecânica e computação. Com o NetLogo é possível desenvolver facilmente, através de programação, ambientes prontos para simulações e também é possível interagir com esses ambientes ao mesmo tempo em que a simulação ocorre. Outra vantagem se dá no fato de existirem soluções prontas para medir o desempenho dos agentes envolvidos na simulação, seja através de gráficos e outros componentes de interface de fácil configuração.

Com as facilidades oferecidas pelo NetLogo na criação de sistemas multiagentes, usamos e modificamos um ambiente de simulação específico para o problema de fuga em um labirinto de salas. O ambiente de simulação foi inicialmente desenvolvido por [Ferreira, Y., 2009]. Nosso trabalho teve como objetivo modificar o ambiente já criado para adequá-lo a novos tipos de simulação e estudos sobre o comportamento de agentes. Mostraremos no decorrer desta monografia os problemas inerentes deste tipo de ambiente e do desenvolvimento de agentes heterogêneos atuando de forma simultânea nas simulações. Analisaremos as características de cada agente desenvolvido e sua atuação no ambiente e definiremos as métricas utilizadas para analisar o desempenho dos agentes criados, nas diversas simulações executadas.

Este trabalho tem como objetivo aprofundar o conhecimento de seus integrantes na área de inteligência artificial, mais especificamente em sistemas multiagentes, de forma

prática, através do desenvolvimento de um ambiente e de agentes em um programa de criação e manipulação destes sistemas.

2. Agentes e ambientes

2.1. Definição de agente

Um agente, do latim *agere* (significa fazer), por definição é algo que age. Agentes no âmbito da computação podem ser definidos como programas de computador que agem. No entanto espera-se que um agente computacional seja mais do que um programa de computador que responde a comandos de um controlador, muitas vezes chamado de agente humano. Um agente computacional deve ser capaz de receber estímulos externos através de sensores e agir no ambiente em que se encontra, através de atuadores, de forma autônoma e independente de comandos [Russel, S., 2003]. Em outras palavras, é um programa que tem como princípio básico "pensar" e tomar decisões com o objetivo de buscar a melhor solução para um determinado problema.

A partir de agora, quando usarmos o termo "agente", estaremos nos referindo exclusivamente aos agentes computacionais.

Antes de caracterizar os diversos tipos de agentes temos que identificar os ambientes que estes manipulam, para que possamos melhor entender os problemas que eles enfrentam e as características que devem ter para manipular melhor o ambiente em que se encontram.

2.2.Tipos de ambientes

Os ambientes nos quais os agentes atuam são essencialmente os “problemas” para os quais os agentes computacionais são as “soluções”.

É através do ambiente, também chamado de mundo, que os problemas se “apresentam” aos agentes e é por conta dele que os problemas existem. Por isso a primeira afirmação deste tópico coloca o ambiente como sendo essencialmente o problema e os agentes, os solucionadores deste problema.

Segundo [Russel, 2003] os ambientes podem ser classificados de acordo com as seguintes propriedades:

- **Completamente ou parcialmente observável.**

  Se os sensores de um agente conseguem captar todo o estado do ambiente a cada instante nós consideramos o ambiente como completamente observável. Caso contrário, quando o agente não tem a todo o momento todas as informações sobre o ambiente este é considerado um ambiente parcialmente observável. A não completude de informações interfere diretamente no desempenho do agente já que este terá de tomar decisões e agir sobre um ambiente no qual ele não conhece totalmente.

- **Determinístico ou estocástico.**

  Quando o próximo estado de um ambiente pode ser completamente determinado pelo estado anterior e pelas ações tomadas pelos agentes neste ambiente, este já pode ser considerado um ambiente determinístico. Quando a ação de um agente interfere no ambiente de uma forma em que não é possível identificar qual será o próximo estado este é um ambiente estocástico ou não-determinístico. Um agente não precisaria, em princípio, se preocupar com a incerteza em um ambiente determinístico, porém se o ambiente for parcialmente observável ele poderá parecer estocástico ao agente, já que este não terá como determinar o estado completo do ambiente a partir de suas ações. Da mesma forma, quando há mais de um agente no ambiente e um deles em particular não conhece as ações dos outros agentes no ambiente, este não poderá considerar o ambiente como determinístico.

- **Episódico ou seqüencial.**

Em um ambiente episódico o agente atua de acordo com episódios independentes que vão da percepção do agente à sua ação no ambiente. A ação do agente em um determinado momento não interfere na próxima decisão a ser tomada por ele. Os episódios que descrevemos aqui são como problemas independentes que o ambiente "apresenta" ao agente, e cada ação que o agente toma corresponde a uma solução do problema apresentado. Quando falamos em diversos problemas independentes não estamos afirmando que o agente foi desenvolvido para solucionar problemas com uma só ação no ambiente, mas sim que ele só precisa se preocupar com a ação a ser tomada no episódio atual para que com uma seqüência de ações ele possa solucionar o problema para o qual ele foi desenvolvido.

- **Estático ou dinâmico.**

Ambientes estáticos são aqueles que não se alteram com o passar do tempo, a menos que recebam uma ação de um agente. Estes ambientes são mais fáceis de manipular por possuir maior previsibilidade que os que estão fora desta categoria. Aqueles ambientes que mudam enquanto o agente ainda está deliberando sua próxima ação são chamados de ambientes dinâmicos. A idéia geral sobre os ambientes dinâmicos é que eles estão continuamente perguntando ao agente o que ele deseja fazer, caso não haja resposta é considerada a decisão de não fazer nada, e enquanto o tempo passa o ambiente continua se modificando. Quando o ambiente não muda com o passar do tempo mas o desempenho do agente muda, consideramos o ambiente como semidinâmico. Um bom exemplo de ambiente semidinâmico é um jogo de xadrez jogado com limite de tempo, conforme o tempo passa o ambiente não muda, porém o agente fica com o tempo mais escasso para fazer sua jogada, portanto seu nível de desempenho muda.

- **Discreto ou contínuo.**

As categorias "discreto" e "contínuo" podem ser aplicadas em relação ao ambiente, ao tempo e às ações do agente. Em um jogo de damas ou xadrez, o ambiente pode assumir uma quantidade finita de estados, logo ele é considerado discreto. Um agente andando por um labirinto de corredores, portas e salas, estará em um ambiente contínuo, já que não existe um número finito de estados em que os agentes no labirinto estejam. Cada agente pode estar em uma posição do corredor, ou de uma sala e realizar um caminho entre salas e corredores de forma contínua, podendo até esbarrar em outros

agentes pelo caminho, tendo de mudar sua trajetória para atravessar um corredor. As ações desses agentes também serão contínuas, indicando a direção que eles seguirão, já que sua movimentação é feita de forma contínua pelos aposentos do labirinto.

- **Agente único ou multiagente.**

Esta definição é bastante trivial. Ambientes com um único agente são chamados de ambientes de agente único. Quando um ambiente apresenta mais de um agente é chamado de ambiente multiagente. Em ambientes multiagente pode acontecer dos agentes serem competitivos ou cooperativos, esta característica acontece de acordo com a natureza do ambiente, em um jogo de xadrez os agentes são competitivos, porque a ação de um agente visa minimizar a chance de sucesso do outro agente e maximizar o seu próprio desempenho. Em um labirinto com vários agentes cujo objetivo é encontrar a saída os agentes são considerados cooperativos já que eles podem trocar informações de forma a se ajudar mutuamente a encontrar a saída.

2.3.Tipos de agentes

Como já foi dito no tópico anterior os agentes são as soluções dos problemas apresentados pelo ambiente, logo, um agente deve ser especificado de forma a solucionar o problema da forma mais vantajosa possível. Com isso os agentes são desenvolvidos com algumas características que visam lhes dar vantagens na solução dos problemas. Algumas destas características são descritas a seguir:

- **Agentes reativos simples**

Agentes reativos são aqueles que respondem intuitivamente às percepções que eles eles fazem do ambiente. Estes agentes não se baseiam no seu histórico de ações e acontecimentos passados para tomar decisões no presente. O processo de decisão agente reativo simples é puramente baseado nas percepções atuais sobre o regras pré-determinadas de ação-reação. Um bom exemplo disto é um agente em um labirinto de salas tentando encontrar a saída, seu processo de decisão pode ser visto na Figura 1 - Representação agente reativo simples

, exibida a seguir:

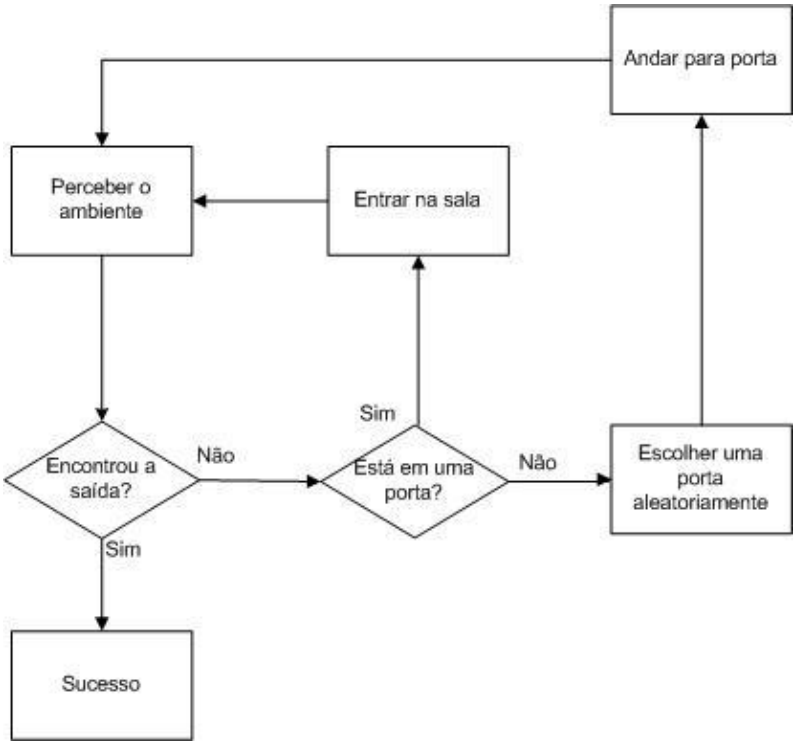


Figura 1 - Representação agente reativo simples

A cada momento o agente percebe o ambiente e recolhe informações sobre o mesmo, caso ele tenha encontrado a saída ele obteve sucesso em seu objetivo de sair do labirinto. Caso contrário o agente irá verificar se está em uma porta e atravessá-la para a próxima sala se for o caso. Quando o agente está em uma sala que não tem a porta de saída, ele simplesmente escolhe uma porta de forma aleatória e anda até esta porta, começando de novo o ciclo com uma nova percepção do ambiente à sua volta. Note que o agente só toma suas decisões baseado nas informações coletadas durante a última percepção do ambiente, ele não armazena nenhuma informação passada e mesmo que armazenasse não as usaria para tomar sua decisão.

Agente reativo simples é o tipo de agente mais simples e de fácil implementação, porém são também os agentes com inteligência mais limitada.

- **Agentes reativos baseados em modelo**

Para aumentar o nível de inteligência de agentes reativos simples é necessário que eles guardem um histórico ou informações sobre o ambiente em que estão, e essas informações podem ser representadas através de modelos. Os modelos são nada mais do que “imagens” do mundo para o agente, ou seja, como o agente imagina que o mundo

esteja no momento atual. Esses agentes que se utilizam de modelos são chamados de agentes baseados em modelo.

A cada instante que o agente recupera informações do ambiente ele deve usar essas informações para atualizar o seu modelo de mundo, tentando assim se manter o mais próximo possível da realidade. O modelo que o agente mantém é usado como base para que ele possa tomar suas decisões. O exemplo a seguir mostra como um agente baseado em modelos pode ser utilizado para encontrar a saída em um labirinto de salas.

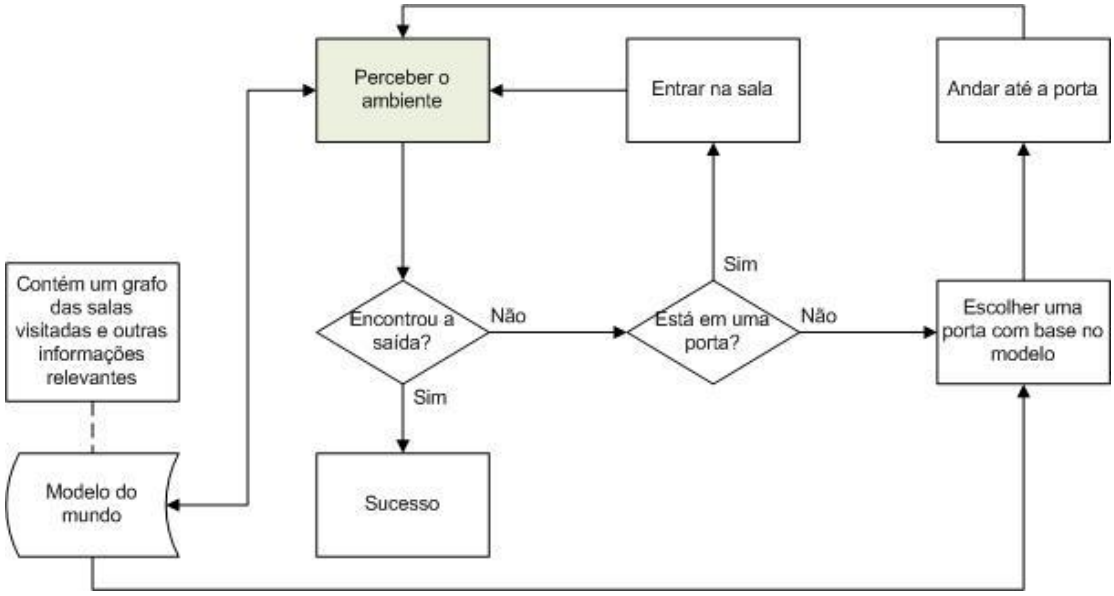


Figura 2 – Representação reativos baseados em modelos

A cada ciclo o agente recolhe informações do ambiente e atualiza sua base de dados, representada na figura pela caixa “Modelo do mundo”. Após esse passo, todo o ciclo de decisão do agente acontece como vimos no agente reativo simples, a diferença mais importante é que este agente usa as informações do modelo para escolher uma porta da sala em que está, de forma que ele não entre em uma sala já visitada, e busque regiões de borda ainda não exploradas. Desta forma, o agente reativo baseado em modelos consegue, na maioria dos casos, encontrar a saída do labirinto mais rapidamente que um agente reativo simples.

- **Agentes baseados em objetivos**

Agentes baseados em objetivos não são regidos por regras fixas como os já mostrados aqui, estes agentes manipulam o ambiente de forma a alcançar objetivos. O objetivo de um agente será alcançado através da execução de um conjunto de ações que

ele determina, analisando as informações coletadas no ambiente e o conhecimento que ele possui sobre como o mundo funciona.

Em outras palavras, quando o agente baseado em objetivos define um objetivo a ser alcançado, ele tenta continuamente criar seqüências de ações, que eles acreditem que sejam as ações necessárias para alcançar este objetivo. A seqüência de ações a que nos referimos é também chamada de plano. Planejamento é um subcampo da IA dedicado à criação de técnicas para encontrar seqüência de ações que alcancem os objetivos do agente.

- **Agentes baseados em utilidade**

Definir somente uma seqüência de ações que levarão o agente a alcançar seu objetivo nem sempre é eficaz. Um exemplo disto é um agente que tem como objetivo sair de um labirinto de salas, mas só pode fazê-lo dentro de um determinado período de tempo. Esse tempo limite para que o agente saia do labirinto pode ser visto como uma necessidade para o agente já que no exemplo ele se encontra em um prédio em desmoronamento e passados esses preciosos minutos o prédio desabará e o agente terá falhado em alcançar seu objetivo. Assim, escolher o caminho mais curto para a saída é uma necessidade, e por isso, criar um plano para alcançar a saída não é suficiente. O agente terá que escolher aquele plano, entre vários possíveis, que o leva até a saída no menor caminho possível.

Mas como escolher um entre diversos planos? É necessário que cada possível plano do agente se transforme em um valor para que ele possa ser comparado aos outros e para que o agente possa escolher o plano de maior valor, ou de maior utilidade. A função que transforma cada possível escolha do agente em um número é chamada de função de utilidade, e é uma das principais ferramentas desse tipo de agente.

2.4. Sistemas multiagentes

Sistemas multiagentes são aqueles nos quais atuam em um mesmo ambiente, diversos agentes, de forma autônoma. O termo autônomo é usado neste contexto para definir que os agentes atuam e existem de forma independente de outros agentes. Isto quer dizer que cada agente tem seus próprios meios de captar informações e de atuar sobre o ambiente de forma independente. Uma das principais características de sistemas multiagentes é a capacidade dos

agentes de cooperar e/ou competir com outros com o objetivo de aumentar suas chances de sucesso ou alcançar o sucesso global com a solução do problema.

Em muitos casos não é preciso que cada agente seja individualmente inteligente para que o sucesso global seja alcançado, um exemplo disso é o comportamento de formigas ou abelhas. Cada agente atua reativamente, na forma estímulo-resposta (ação-reação) e de forma independente, porém o trabalho individual feito por uma grande comunidade de agentes gera um comportamento global que leva ao sucesso. Essa característica é chamada muitas vezes de auto-organização.

Por outro lado, agentes inteligentes podem ser capazes de se comunicar com outros agentes com o objetivo de obter informações que os ajudem. A comunicação entre agentes é um complexo subcampo da teoria de agentes, já que é necessário levar em consideração que outros agentes podem não ser confiáveis a ponto de passar informações que prejudiquem o agente. Em outros casos agentes podem firmar acordos de cooperação e não honrar o compromisso. Com todas essas variáveis, diversos estudos têm sido realizados nos últimos anos com o propósito de avançar neste campo, desde padronizações de linguagens de comunicação universal entre agentes, como a FIPA-ACL e KQML, a estudos sobre governança de agentes, e frameworks de certificação de confiabilidade de agentes, através de mecanismos de reputação e testemunhos.

Alguns exemplos famosos da aplicação dos conceitos de sistemas de reputação são sites como eBay e Amazon.

No tópico 3 descreveremos um sistema de simulação chamado NetLogo, que usamos para implementar um simulador multiagente, e explicaremos como os diversos tipos de agentes atuam nesse ambiente específico.

3. NetLogo

O NetLogo é uma linguagem de programação simples e adaptada à simulação de fenômenos naturais e/ou sociais [Wikipedia, 2010]. É baseado no paradigma de agentes. Um agente no NetLogo:

- É um indivíduo autônomo, munido de um conjunto de características e regras que governam o seu comportamento e com a capacidade de tomar decisões.
- Interage com um conjunto de agentes e/ou com o meio ambiente obedecendo a um conjunto de regras.
- São flexíveis e têm a capacidade de aprenderem e de adaptarem o seu comportamento baseados na experiência. Assim requer alguma forma de memória.
- Podem ainda ter regras para mudar as suas regras de comportamento.

O NetLogo suporta basicamente três tipos de agentes:

- "Turtles", que são agentes que se movimentam pelo mundo. Nesta versão do NetLogo o mundo é bidimensional e está dividido em um grid de "patches".
- Cada "patch" é um pedaço quadrado do "terreno" do mundo, sobre o qual as "turtles" podem se movimentar.
- O "observador" é um agente único que não tem localização. Ele pode ser imaginado como o "criador", que interage com o mundo, formado pelas "turtles" e pelos "patches".

A linguagem NetLogo é bem adequada à modelagem de sistemas complexos, que evoluem no decorrer do tempo, e também permite explorar as interações entre os indivíduos, assim como os padrões emergentes destas interações. Estas características tornaram o NetLogo um ambiente bastante popular, e as aplicações criadas para ele vão desde modelos do comportamento dos agentes no seu conjunto de ações, à previsão da propagação de

epidemias, ou à modelagem de evasão fiscal, ou à modelagem da evacuação de multidões, e muitas outras [NetLogo, 2009].

É também um ambiente de programação fácil de usar para criar e testar tais modelos.

- Permite executar e experimentar simulações.
- Permite criar modelos para testar hipóteses sobre sistemas descentralizados.
- Vem munido de uma grande biblioteca contendo simulações em ciências naturais e/ou sociais, que podem ser usadas e modificadas.
- Os modelos são construídos usando uma linguagem simples, mesmo para aqueles que estão a aprender a sua primeira linguagem de programação.
- Possui uma interface gráfica fácil de usar.

Tais aspectos justificam a escolha do NetLogo como linguagem de programação utilizada neste trabalho.

4. Evolução do simulador

O ambiente de simulação criado inicialmente por [Ferreira, Y., 2009] tem capacidade de configuração de alguns parâmetros de simulação como a modificação do número de agentes, variação do espaço de simulação, gravação de arquivo de log, entre outras características.

Uma das principais tarefas deste trabalho foi modificar o ambiente de simulação para capacitá-lo a aceitar mais opções de configuração dando uma maior capacidade de estudo do comportamento e do desempenho de agentes. Além disso, adicionamos mais opções de visualização das variáveis de simulação, através de gráficos e outros componentes do NetLogo.

Outras opções de configuração da simulação foram adicionadas para que o usuário pudesse escolher o posicionamento dos focos de incêndio iniciais, e para salvar e carregar o contexto de uma simulação para que ela pudesse ser executada diversas vezes com os mesmos parâmetros.

Durante a evolução do sistema percebemos que a criação de salas e portas deveria ser reformulada para que as novas funcionalidades pudessem ser implementadas de forma mais fácil. Porém, decidimos não refazer este trecho do código por não dispormos de tempo suficiente para tal.

Durante o desenvolvimento tivemos que realizar mudanças significativas no código da simulação. Assim, vimos que seriam necessárias aplicações de diversas refatorações. Uma melhor descrição das mudanças efetuadas está descritas no apêndice B.

Modificamos também o código referente à implementação dos diversos tipos de agentes. Estas implementações serão descritas no tópico 6.

5. Domínio e ambiente

O problema escolhido para que planejássemos a criação do ambiente e dos agentes foi um caso específico do problema de fuga. Este grupo de problemas consiste em apresentar ao agente um ambiente com diversas opções de locomoção e várias possibilidades de caminhos a escolher, e o objetivo é encontrar a saída do ambiente. Normalmente o problema se apresenta como um labirinto onde só existe uma saída, no nosso caso o ambiente é composto de um conjunto de salas geminadas, com portas interconectando-as, e as saídas se encontram na região de borda do mesmo. Nenhum agente tem conhecimento da geografia do ambiente e o objetivo deles é encontrar a saída no menor tempo possível. Para isto cada agente obedece a uma estratégia que tem a ver com suas habilidades cognitivas.

5.1. Problema de fuga

O ambiente de simulação proposto pode ser visto como um andar de prédio de onde os agentes deverão sair. Este prédio trás como motivação para o desejo de fuga dos agentes um fator limitador de tempo, que se dá na forma de um incêndio. Durante a simulação, focos de incêndio aparecem no ambiente, e conforme o tempo passa o incêndio se espalha pelas salas diminuindo cada vez mais a chance de sucesso dos agentes. Logo, os agentes devem se preocupar também em evitar entrar em salas onde o incêndio já tenha se espalhado, porque a proximidade com o fogo irá feri-los rapidamente até a sua morte. Imagens do ambiente podem ser vistas na Figura 3 – Propagação do fogo e na Figura 4– Propagação do fogo estágio avançado:

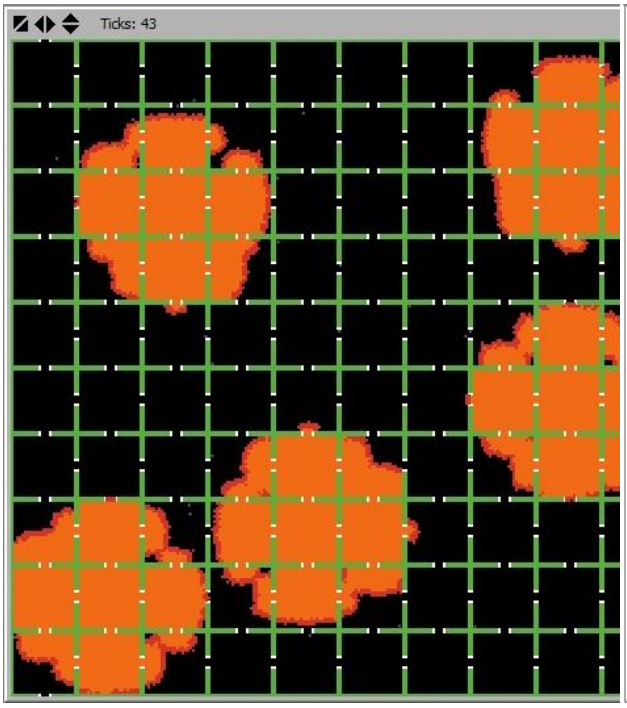


Figura 3 – Propagação do fogo

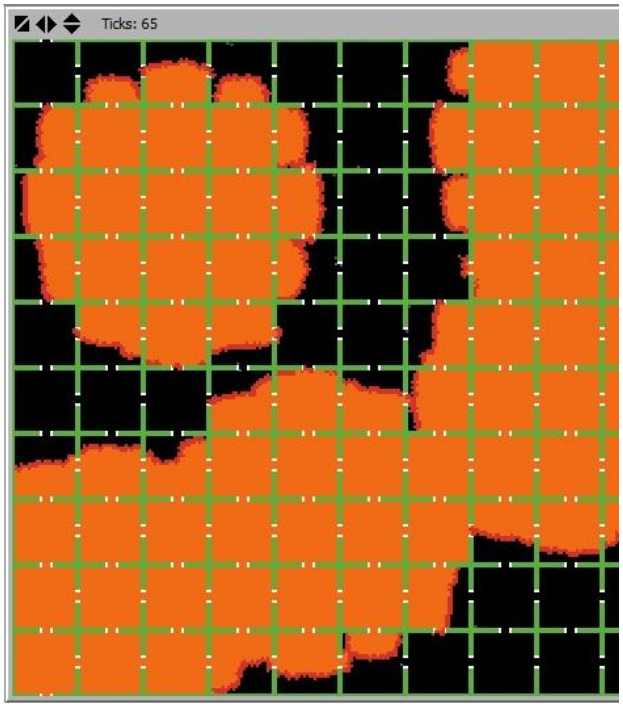


Figura 4– Propagação do fogo estágio avançado

As imagens **Erro! Fonte de referência não encontrada.** mostram a organização das salas, representadas em verde, e suas portas, representadas em branco. É possível notar também a evolução do incêndio (em laranja) no decorrer de uma simulação.

5.2.Ambiente

O ambiente da simulação foi criado de forma que atendesse às especificações do problema proposto. Descreveremos nos próximos tópicos o processo de criação dos elementos do ambiente.

5.2.1. Salas

As salas do labirinto são idênticas umas às outras só apresentando diferenças nas áreas de borda, onde a quantidade de portas é menor. A disposição das salas e portas segue sempre o mesmo padrão entre diversas simulações facilitando assim a comparação de desempenho dos agentes.

Cada sala é criada de forma independente das outras. Isto na prática quer dizer que cada parede de uma sala é geminada à outra parede da sala seguinte. Como regra geral cada parede tem uma porta, representada em branco no ambiente, como podemos ver na **Erro! Fonte de referência não encontrada.**, logo, as portas normalmente ficam geminadas a outras portas também. Esta característica de certa forma simplifica os

cálculos no momento de criação das paredes e portas de cada sala, porém percebemos que esta estratégia é responsável por grande parte dos problemas encontrados durante a implementação dos agentes e outras funções auxiliares do ambiente. Muitas vezes os agentes precisam escolher uma porta para atravessar para a próxima sala, porém, como as portas são geminadas é necessário ter cuidado na implementação para que a porta certa seja escolhida, ou seja uma porta de outra sala,pois a implementação de portas geminadas pode levar a uma escolha errada e o agente corre o risco de continuar na mesma sala em que se encontra ao invés de atravessar para a próxima sala.

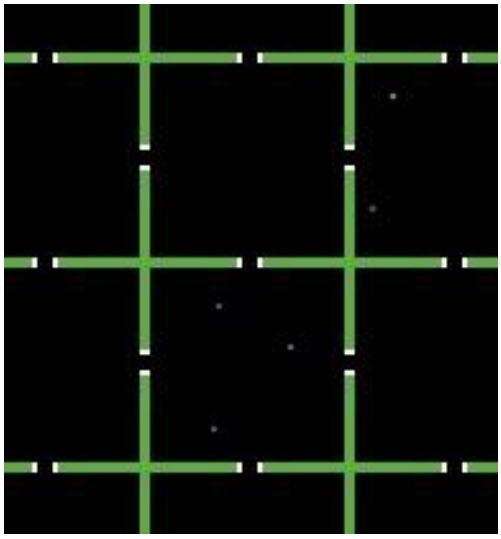


Figura 5 – Agentes em salas

Outra característica que vale ser ressaltada é que cada parte de uma parede é criada como sendo um agente fixo, mostrado em verde na **Erro! Fonte de referência não encontrada.**. Esta foi uma facilidade escolhida para que se pudesse ter um controle maior sobre as colisões entre agentes e paredes, porém, com um estudo mais aprofundado da linguagem percebemos que esta estratégia acarreta uma sobrecarga no sistema NetLogo, que passa a ter um número dezenas de vezes maior de agentes para simular, levando à simulação a saturar[1] com um número menor de agentes móveis. O mesmo resultado poderia ter sido alcançado sem a geração de diversos agentes fixos para cada parede, utilizando funções específicas para manipular patches, detectando e verificando assim as colisões entre agentes e paredes através das cores dos patches de parede.

[1]O termo saturar é utilizado aqui com o sentido de chegar ao limite superior de agentes em uma simulação. Cada agente adicionado além deste número tornaria a simulação mais lenta e impraticável.

Conforme evoluímos nos estudos sobre o ambiente criado, fomos percebendo aos poucos as características envolvidas na criação das salas e os problemas que estas características acarretavam na implementação dos agentes, porém, por se tratar de um projeto de curto prazo de implementação decidimos por manter as características já citadas. Outro fato relevante para esta decisão foi o fato das características das salas e portas serem questões essenciais para todo o código, estando fortemente acoplada às outras regiões, e com isso grande parte do código teria que ser alterado para entrar em conformidade com as mudanças na criação do ambiente, consumindo assim um tempo de implementação que não teríamos. Assim, decidimos nos concentrar na criação dos agentes e adicionar flexibilidade às configurações da simulação. Neste processo realizamos diversas atividades de refatoração no código para que os problemas identificados pudessem ser minimizados.

Maiores informações sobre refatoração e manutenção de código na linguagem NetLogo podem ser encontradas no Apêndice B deste trabalho.

5.2.2. Fogo

O fogo que se propaga pelo ambiente enquanto uma simulação ocorre é criado como um agente, na verdade uma série deles, que a cada intervalo de tempo se replicam, criando outros semelhantes nos patches que ainda não tem fogo. Desta forma o fogo se expande do centro para as bordas em forma circular, como pode ser observado na Figura 6 e na Figura 7.

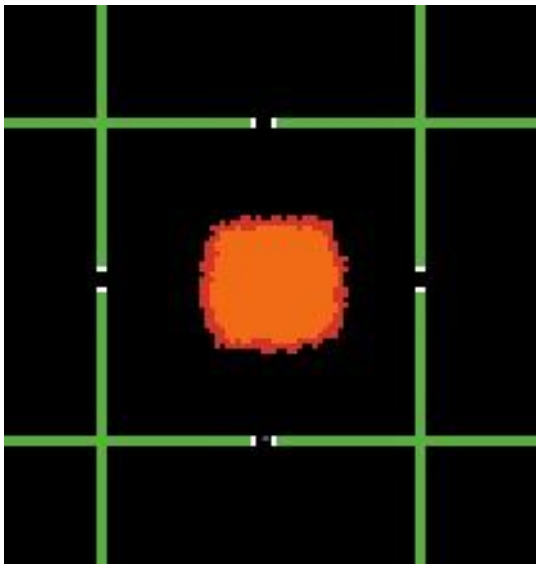


Figura 6 – Expansão radial

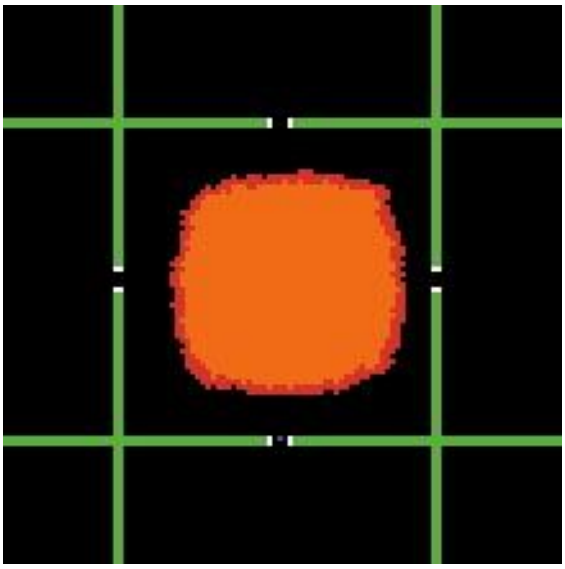


Figura 7 – Expansão radial

A expansão do fogo ocorre naturalmente e é implementada de forma bem simples já que uma das formas de criação de agentes no NetLogo é feita utilizando o comando “sprout”. Este comando faz “brotar” de um patch um novo foco de incêndio, logo, para que o algoritmo de expansão de focos funcione só é preciso que cada foco escolha alguns patches ao redor de sua localização e faça com que eles “brotem” suas duplicatas. O único cuidado que deve ser tomado é para evitar que focos brotem em posições onde já existe fogo, assim evitamos desperdício na criação de agentes na simulação.

Como a expansão do fogo se dá de forma radial, os focos mais internos já estão cercados de outros focos, logo eles não criarão mais focos de incêndio ao seu redor e sua existência passa a ser desnecessária, por isso esses focos se “matam” quando não podem mais se duplicar, porém antes disto eles mudam a cor do patch em que estão para que o fogo possa continuar a ser representado, na cor laranja. Esta característica pode ser notada na Figura 6 e Figura 7, onde os patches em laranja mais claro são os que já estão sem agentes. Esta estratégia visa evitar um grande crescimento do número de agentes de fogo durante a simulação.

O usuário tem a capacidade de escolher a posição de cada um dos focos no ambiente, assim é possível estudar padrões relacionados às posições dos focos.

5.3. Agentes

Como já foi dito no início deste tópico, o objetivo dos agentes principais da simulação (reativos, seguidores e cognitivos) é encontrar a saída do ambiente com a menor exposição ao fogo e no menor tempo possível. No problema apresentado, os agentes são capazes de perceber o ambiente à sua volta, se movimentar, identificar saídas do ambiente, focos de incêndio e outros agentes, porém não identificam o nível de inteligência de outros agentes. Eles têm a capacidade de se movimentar livremente dentro das salas, sem nenhuma restrição, e atravessar salas através de suas portas. É importante salientar que os agentes podem se movimentar em regiões onde há fogo, porém, fazendo isto suas “vidas” diminuem rapidamente os levando à morte e conseqüentemente falhando em alcançar seu objetivo. A Figura 8 mostra agentes em uma simulação, representados por pontos dentro das salas.

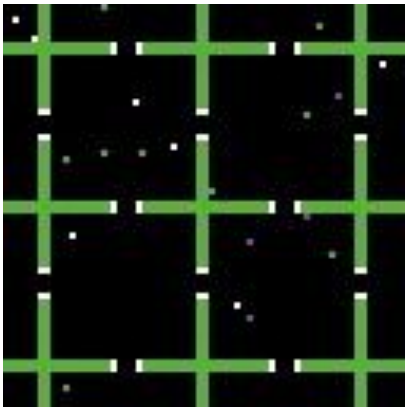


Figura 8 – Agentes em uma sala

Outras informações sobre como os agentes foram implementados e suas estratégias de atuação no ambiente serão apresentadas no tópico 6 deste trabalho.

5.4.Parâmetros de simulação

Para ajudar na composição do ambiente foram introduzidos alguns componentes gráficos para configuração de uma simulação. Essas configurações nos permitem aplicar um controle maior, ou até mesmo estudar um comportamento especifico para algum agente. Podemos aplicar a uma simulação algumas características particulares tais como:

- **Simulação encerrada por ticks**

  Uma simulação pode ser encerrada de duas formas diferentes. A primeira delas é quando não existem mais agentes no ambiente de simulação, seja por morte ou por não conseguir escapar do ambiente de perigo. A segunda, quando configurada na interface, permite escolher até qual tick (ciclo) a simulação irá ocorrer. Isto permite coletar dados interessantes do estado dos agentes e do ambiente em um determinado instante de simulação. Na Figura 9 é possível ver como esta opção pode ser configurada.

Figura 9 – Configuração de tick

- **Numero de simulações**

Representa o número de simulações que serão executadas seguidamente. Este parâmetro é importante, pois nos permite estudar comportamentos de um mesmo contexto definido para simulação.

- **Escolha de posição para focos**

Através da interface também podemos habilitar/desabilitar a posição na qual os focos começaram a propagar. Essa posição pode ser guardada para as n vezes que a simulação for repetida. Quando o usuário não desabilitar a opção de escolha da posição dos focos o sistema irá atribuir randomicamente uma posição para cada foco.

- **Número de focos**

É possível configurar o número de focos que estarão presentes na simulação.

O número de focos e o posicionamento dos mesmos podem ser vistos nas figuras a seguir:

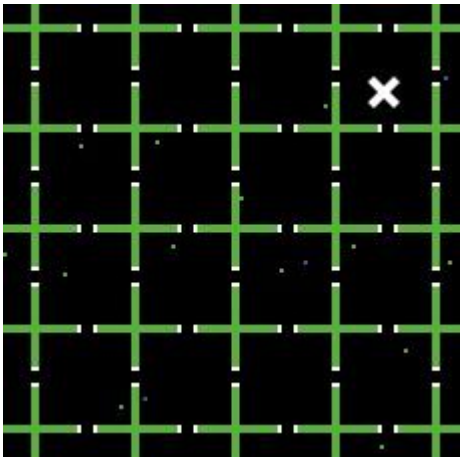


Figura 10 – Posicionamento de foco

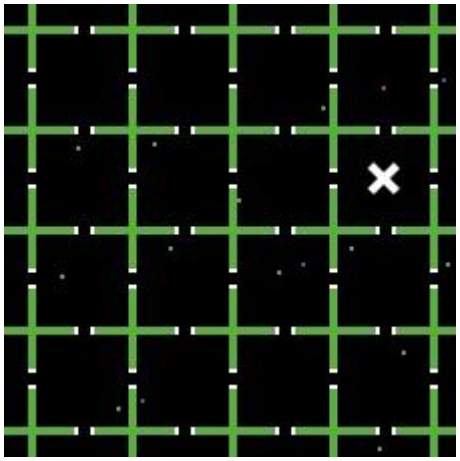


Figura 11 – Posicionamento de foco

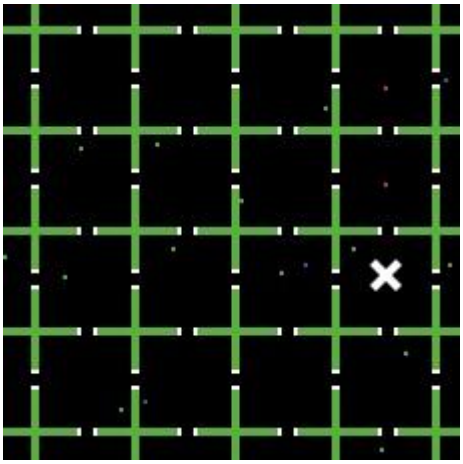


Figura 12 – Posicionando foco

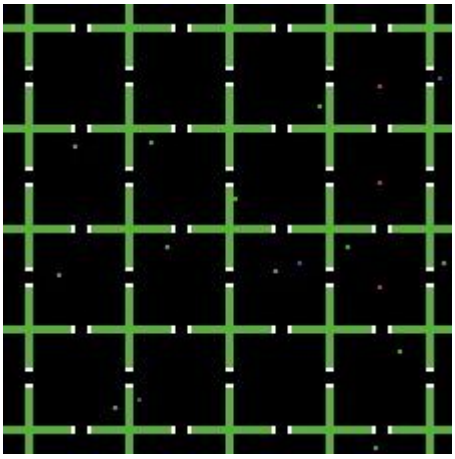


Figura 13– Inicio do incêndio

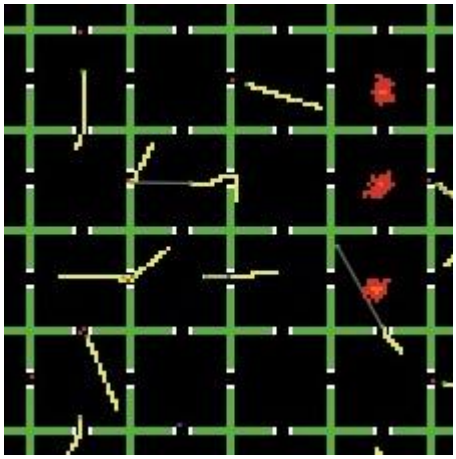


Figura 14 – Rastro de depuração

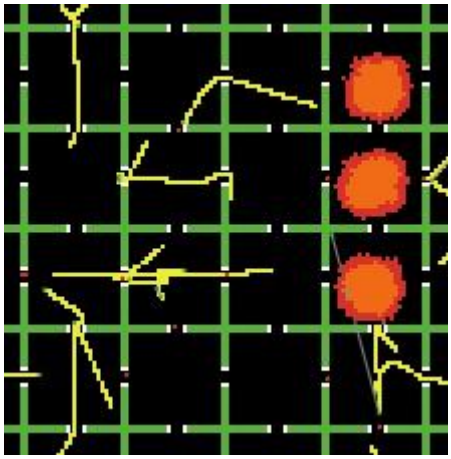


Figura 15– Rastro de depuração

Nas figuras acima, as imagens mostram como o usuário pode configurar, com o cursor do mouse, o posicionamento dos focos. As figuras: Figura 10, Figura 11 e Figura 13 mostram o início da simulação e a expansão dos focos posicionados pelo usuário. O rastro amarelo que pode ser visto na Figura 14 e Figura 15 mostra o caminho percorrido por um agente e é um recurso para depuração de erros na implementação dos agentes, utilizado durante a fase de desenvolvimento.

- **Tamanho do ambiente**

Ao modificar esta variável de configuração o ambiente sofre alterações, tais como tamanho de salas. Na Figura 16 podemos ver o controle de interface com opções de tamanhos para o ambiente.

Figura 16 – Configuração do tamanho do ambiente

- **Número de agentes**

Para cada série de simulações é permitido atribuir, uma quantidade, para cada tipo de agente presente na simulação. Existe um limite máximo de 200 agentes. Este limite foi estipulado afim de não tornar a simulação lenta, visto que muitos dos elementos de ambientes como salas e focos já são agentes. Na Figura 17 podemos ver os controles para ajuste da quantidade de agentes na simulação.

Figura 17– Configuração de agentes

- **Posição aleatória de Agentes**

Ao ligar esta chave, para cada série de simulações, a posição inicial de cada agente é salva. Na Figura 18 é possível ver a chave e a mensagem que confirma o salvamento da posição inicial dos agentes.

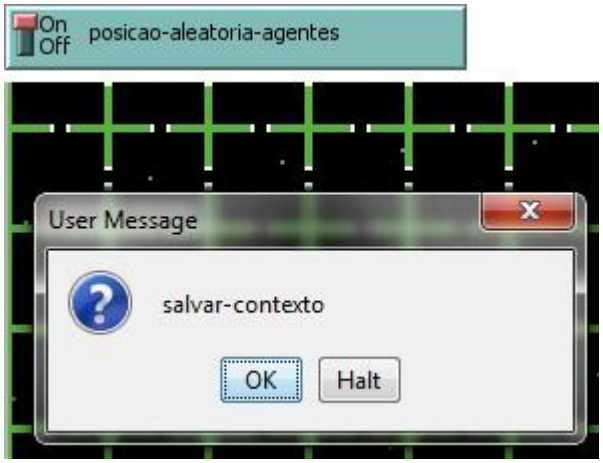


Figura 18 – Salvamento de contexto

- **Gráficos**

Ao longo da simulação podemos observar gráficos com informações em tempo real sobre o número de agentes em simulação e média de quantidade de vida. A Figura 19 ilustra os tipos de gráficos presentes no simulador.

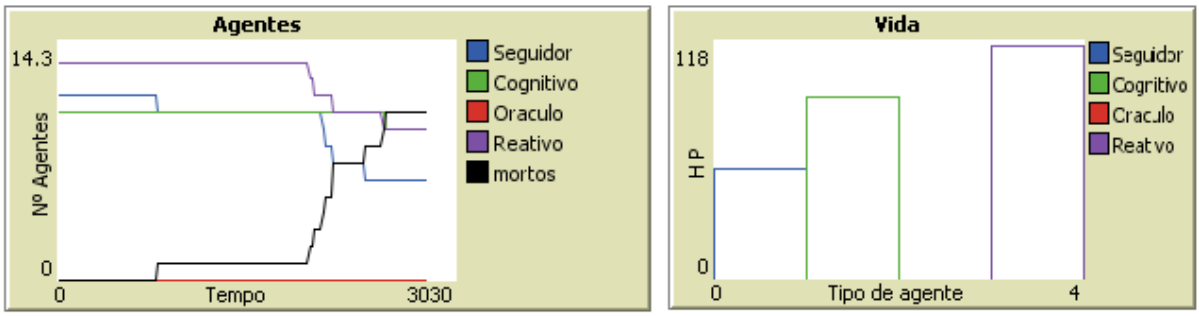


Figura 19 – Gráficos de simulação

- **Dados Estatísticos**

Além dos números de agentes em simulação e média de quantidade de vida é possível visualizar alguns resultados de uma simulação tais como: tempo médio de fuga, número de mortos por asfixia, número de agentes presos.

6. Implementação dos agentes

Neste capitulo serão apresentadas as estratégias definidas para cada tipo de agente presente na simulação. Os três tipos são: reativo, seguidor e cognitivo. Desejávamos representar comportamentos distintos para tipos diferentes de pessoas, por exemplo o reativo como uma pessoa em pânico, incapaz de tomar decisões mais refinadas, ou o cognitivo que seria uma pessoa mais calma e com capacidade de analisar melhor o ambiente. Todos objetivam a fuga do ambiente de risco, cada qual com suas prioridades de ações e características particulares.

6.1. Comportamentos comuns

Apesar das diferenças existentes entre os tipos de agente até aqui citados alguns comportamentos não tem qualquer tipo de dependência com a classificação do agente. Podemos destacar:

- **Verificação de sala de saída:**

Ao entra em uma nova sala um agente sempre verifica se esta é uma sala que possui uma saída do ambiente. Caso verdade, o agente imediatamente dirige-se a saída. Caso contrário o agente adotará o passo seguinte de sua estratégia.

- **Verificação de fogo**

Ao detectar um foco de incêndio em uma sala o agente verifica se sua a distancia para a próxima porta escolhida é menor que sua distancia para o foco. Caso verdade o agente irá se dirigir para a porta escolhida, caso contrário o agente toma uma direção oposta, ou seja, 180 graus ao foco e procura uma nova porta nessa nova direção.

- **Cone de Visão**

O NetLogo permite a utilização de uma função chamada: “in-cone” [NetLogo, 2009]. Esta função calcula um cone de visão, de origem no agente, com um ângulo escolhido pelo programador. Através deste cone de visão podemos determinar o que cada agente consegue enxergar, sendo assim, a princípio esta seria a principal função para auxiliar os agentes na escolha da próxima porta. Contudo, esta função não é trivial para o framework e acarreta grandes perdas de tempo dentro da simulação. Somado a esta descoberta verificamos que o uso do cone de visão torna-se desnecessário quando o agente esta entrando por uma porta já que possui a visão de todas as outras portas de uma sala.

Ainda assim o cone de visão tem sua importância dentro da simulação. Eventualmente quando um agente por algum motivo desorienta-se e fica em algum canto de sala essa função é essencial para ajudar o agente na escolha da próxima porta.

6.2.Reativo

A estratégia definida para o agente reativo baseia-se em uma premissa: a escolha da próxima porta. Este agente sempre escolhe aleatoriamente uma das portas da sala na qual ele se encontra.

Essa estratégia permite ao agente escolher rapidamente a próxima sala que irá visitar. Sendo assim o agente não perde vários ciclos para escolha de um ponto de interesse, fazendo a escolha no momento em que entra em uma nova sala.

Embora a premissa seja encontrar a próxima porta, assim como todos os outros agentes, primeiramente o reativo verifica se a sala atual tem uma porta de saída do ambiente e se ela possui algum foco de incêndio. Isto faz com que o agente execute uma ação como “sair do ambiente” ou “fugir na direção oposta ao fogo” antes da escolha da nova sala. A Figura 20 ilustra este comportamento.

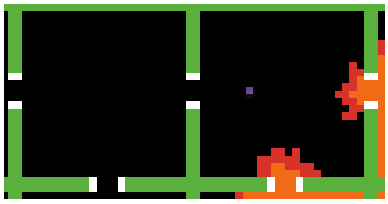


Figura 20 – Agente Reativo

Um agente reativo representa uma entidade que não possui memória, ou seja, este agente não será capaz de armazenar um histórico de lugares visitados, ou até mesmo guardar

pontos onde os focos de incêndio estão presentes, ele simplesmente sente o ambiente e atua de forma reativa.

A principal motivação da criação deste tipo de agente foi para representar uma pessoa em pânico, desorientada e incapaz de manter um raciocínio mais refinado sobre o ambiente onde se encontra, excetuando-se a fuga de uma sala em chamas o que seria instintivo. Outra motivação é que o desempenho deste agente sem memória pode servir como base de comparação com outros tipos de agentes.

Na figura a seguir podemos ver um fluxograma simplificado, descrevendo o algoritmo do agente reativo.

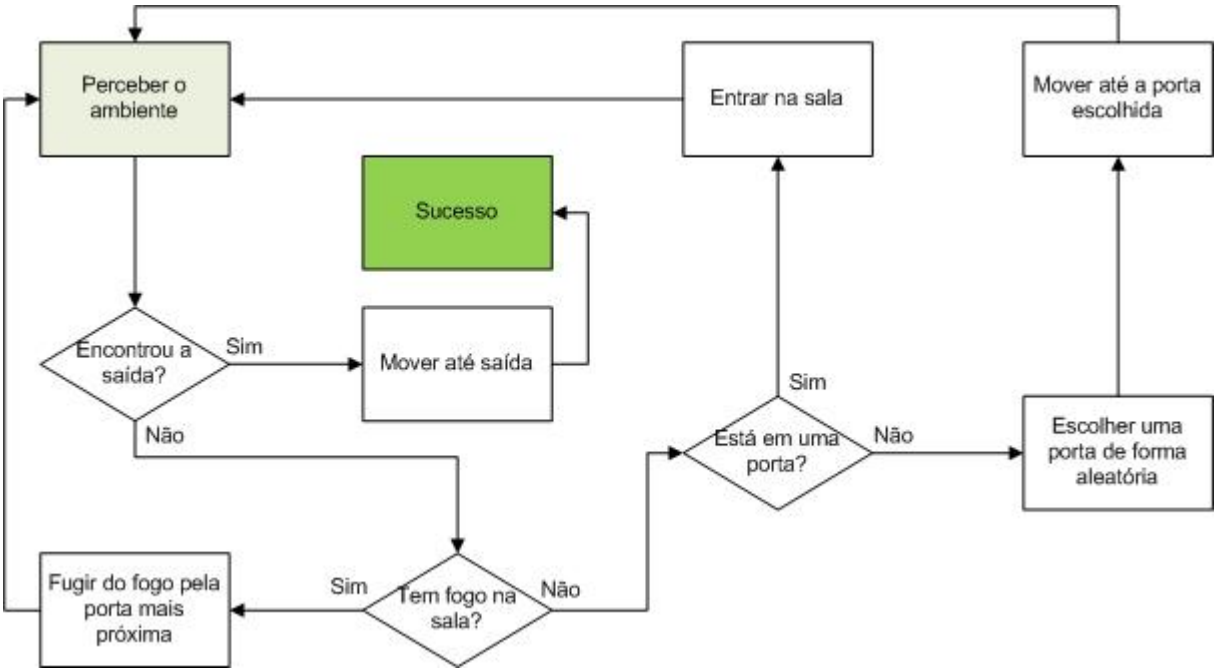


Figura 21– Fluxograma agente reativo

6.3.Seguidor

A estratégia definida para o agente do tipo seguidor é bem simples: “siga”. A cada ciclo de simulação este agente executa uma rotina de busca com interesse em localizar o agente dentro do seu campo de visão que possui a menor distância para ele. A este agente damos o nome de agente alvo. É importante reforçar que o agente alvo é um agente móvel qualquer, podendo ser qualquer um dos três tipos existentes na simulação.

Esta rotina de busca pode retornar os seguintes valores: uma referência para o agente alvo encontrado ou “nobody”, que é o termo da linguagem NetLogo para indicar a ausência

de agente. Quando um agente alvo não é encontrado pela rotina de busca o agente seguidor apresentará comportamento reativo, escolhendo uma porta qualquer da sala para atravessar. Quando a busca retorna um agente alvo, um link de identificação entre os agentes é criado e o seguidor passa efetivamente a seguir o agente alvo.

A Figura 22 ilustra o link criado entre o agente alvo e o agente seguidor.

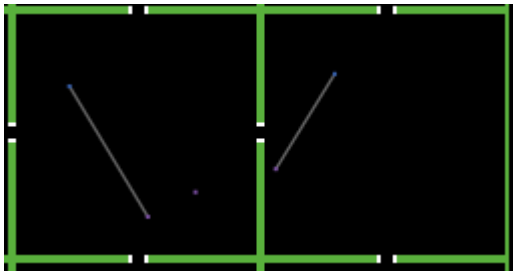


Figura 22 – Elo de ligação entre agente alvo e seguidor

Esta estratégia atenta para alguns pontos importantes:

- **Troca de seguidor**

  Um seguidor irá seguir o agente alvo enquanto este for o agente com menor distância, a partir do momento que outro agente assumir uma distância menor para o seguidor este será o novo agente alvo. Assim o link de identificação com o agente alvo antigo será apagado, o novo agente alvo será atribuído e um novo link identificação será criado.

- **Campo de visão**

  Quando um agente alvo sair do campo de visão do seguidor, ou seja, quando eles já não estiverem mais na mesma sala, o link de identificação é apagado e o agente seguidor se dirige à porta na qual o agente alvo passou. É importante ressaltar que durante o tempo de locomoção até a porta se algum outro agente na mesma sala do seguidor apresentar distância menor que a distância deste mesmo seguidor para a porta em questão, o novo agente alvo passa a ser o agente com menor distância.

- **Restrição de seguidor**

  Um problema surge quando um agente seguidor escolhe como agente alvo outro agente seguidor que já o está seguindo. Para melhor ilustrar, antes de criar o link e efetivamente seguir outro agente os agente seguidores fazem a seguinte pergunta: “estou tentando seguir quem está me seguindo?”. Se a respostar for negativa, o link de identificação é criado, caso contrário o agente seguidor corrente passa a ter comportamento reativo até a escolha de um novo seguidor.

- **Ciclo de seguidores**

  Um grande problema dos agentes seguidores é o ciclo de seguidores. Neste caso um agente segue um segundo agente, que segue um terceiro, que está seguindo o primeiro agente. Um ciclo contém pelo menos três agentes, mas este número pode aumentar bastante. De forma diferente de quando apenas um agente decide seguir quem o está seguindo, quando temos a configuração de ciclo de seguidores é muito lento realizar a verificação, seriam necessários vários ciclos para que um agente decida parar de seguir seu alvo, quebrando assim o ciclo, e impedir que isto aconteça seria impossível sem adicionar inteligência na implementação dos agentes seguidores, o que implicaria em uma contradição à sua especificação de seguidor “puro”.

Na figura a seguir pode ser visto o fluxograma simplificado do algoritmo de atuação do agente seguidor.

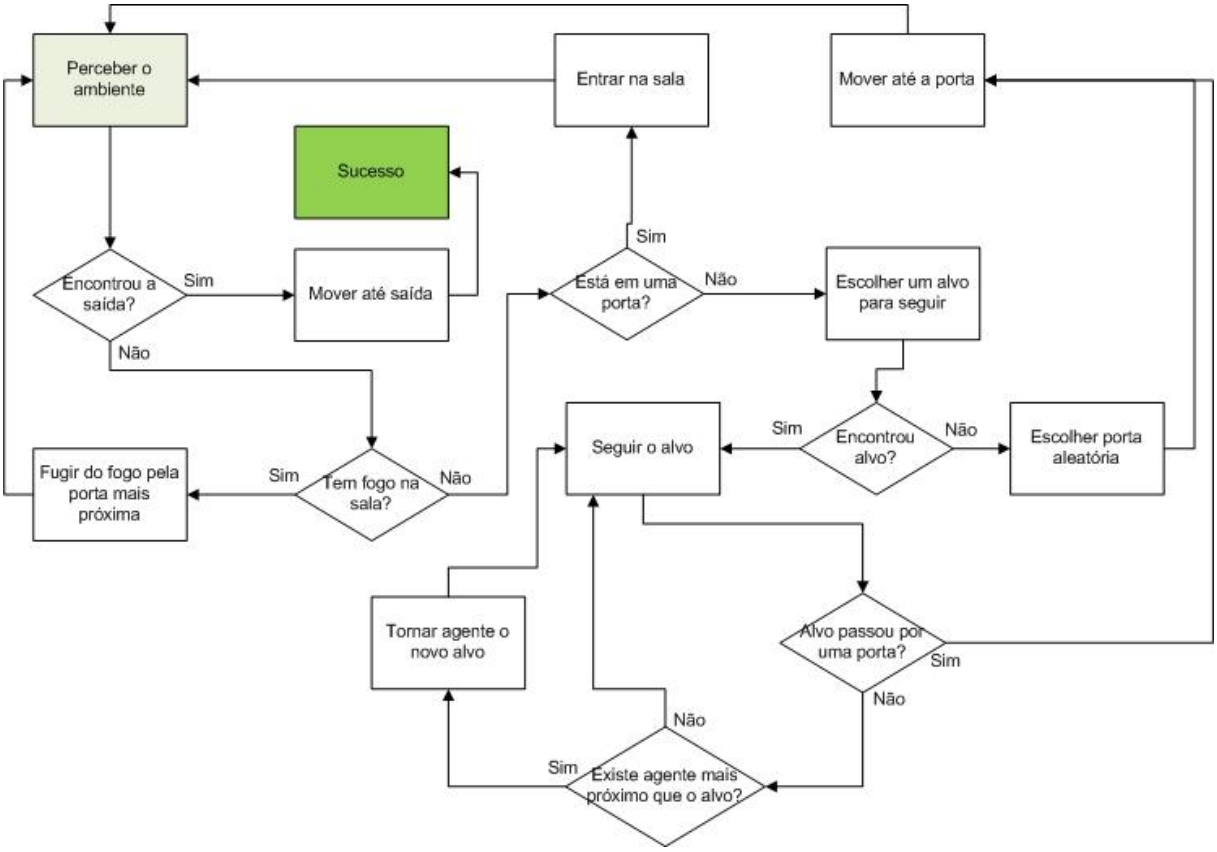


Figura 23 – Fluxograma agente seguidor

6.4. Cognitivo

O agente cognitivo foi pensado para ser o tipo mais desenvolvido dentre os existentes na simulação. Estes agentes têm a capacidade de guardar informações sobre as salas onde já estiveram, buscar regiões de fronteira em busca da saída global, planejar novas rotas e se comunicar com outros agentes inteligentes.

Um agente cognitivo age seguindo o plano criado por ele. Um plano nada mais é do que um conjunto de ações que o agente executa em uma determinada ordem. Cada ação é uma tarefa básica para o agente cognitivo, como seguir algum outro agente ou andar em alguma direção. Quando uma ação é executada de forma completa ela é removida do plano do agente.

No início da simulação o seu conhecimento sobre o mundo é mínimo e ele não tem um plano de atuação, assim, o processo de planejamento entra em ação. Neste processo o agente cria um plano baseado nas informações que ele guarda sobre o mundo, como a quantidade de salas visitadas, tempo desde a última comunicação, etc. O plano é reavaliado constantemente e pode ser alterado para que o agente consiga atingir seus objetivos. Por exemplo: O agente tem o objetivo de seguir outro (alvo) e seu plano contém a ação “seguir”. Porém, quando o

agente alvo sai do seu campo de visão, o agente cognitivo acrescenta a ação “atravessar porta x” para que ele possa atravessar a porta por onde o agente alvo passou e voltar a segui-lo.

Inicialmente o agente deve saber reconhecer a saída global do ambiente, por isto a primeira verificação que o agente faz é para saber se existe alguma saída global na sala em que ele está. Em caso afirmativo o agente renova o seu plano com uma ação para sair do ambiente, caso contrário o agente passa à escolha dos próximos passos de acordo com seus conhecimentos sobre o ambiente.

Tendo em vista que a configuração de ambiente escolhida para as simulações é um mundo com 100 salas, algumas considerações foram tomadas durante o desenvolvimento do agente cognitivo. Quatro faixas de atuação foram definidas, elas são descritas a seguir:

- Quando a simulação começa, nenhum agente tem conhecimento suficiente do ambiente para a troca de informações, por isso o melhor a fazer é que o agente explore o máximo possível o ambiente à procura da saída, se movendo aleatoriamente.
- Conforme a simulação evolui, os agentes cognitivos passam a escolher portas a seguir, de acordo com seu conhecimento sobre as salas já visitadas, evitando assim, a visitação de salas em que o agente já passou.
- Com o tempo mais avançado e o aumento da quantidade de salas visitadas os agentes podem passar a se comunicar de forma eficiente. Assim, os agentes cognitivos procuram outros agentes para se comunicar. A cada comunicação os agentes perdem a oportunidade de se mover, porém ganham informações úteis para encontrar a saída do ambiente.
- Muitas vezes com a escassez de agentes com capacidade de comunicação nas proximidades do agente cognitivo, este passa a procurar pela saída seguindo para regiões de borda, aumentando assim suas chances de encontrar a saída.

A figura a seguir mostra o fluxograma correspondente ao comportamento do agente cognitivo. Este agente claramente tem um algoritmo mais complexo do que os outros tipos de agentes, contendo inclusive uma memória interna com informações sobre as salas já visitadas.

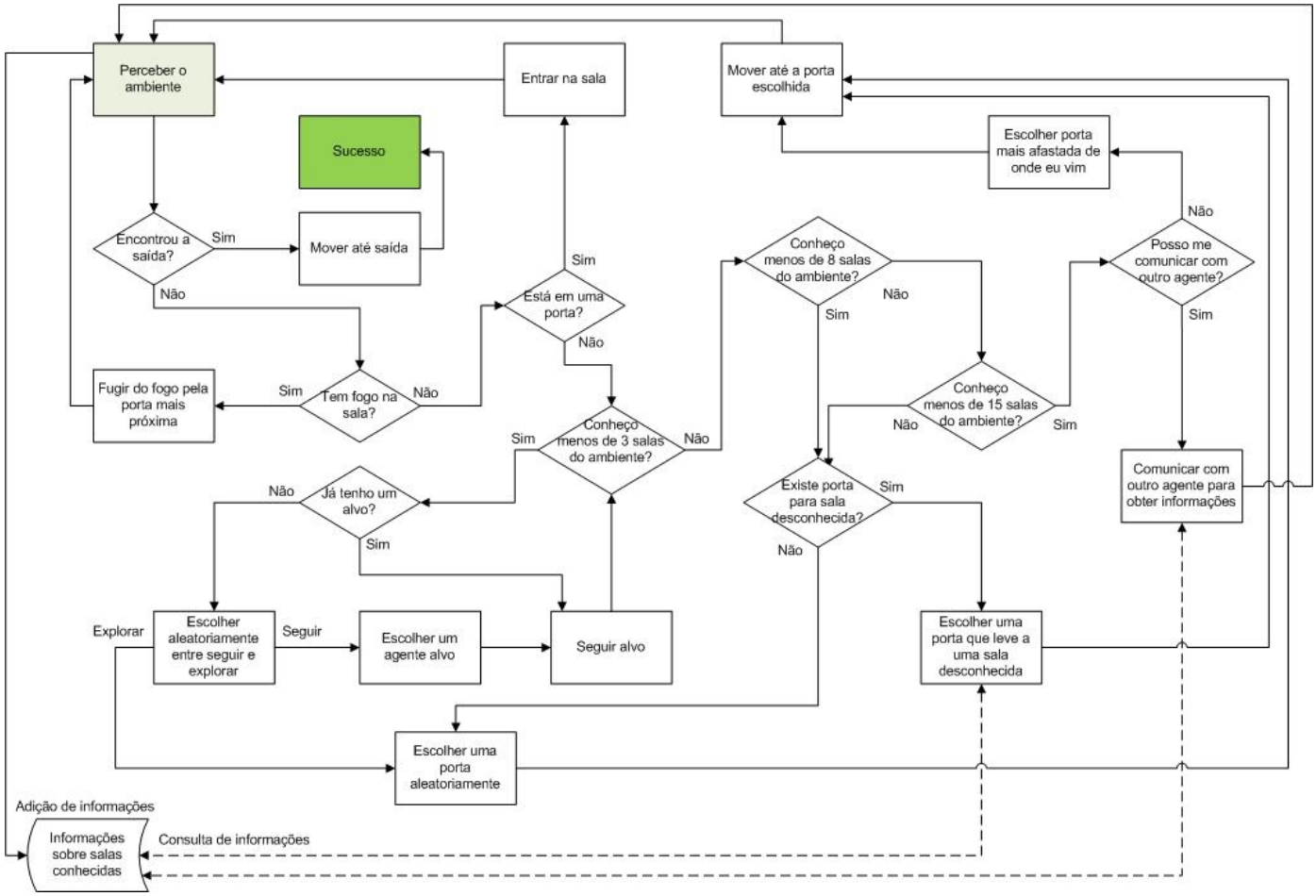


Figura 24 – Fluxograma agente cognitivo

7. Metodologia

O objetivo deste trabalho é estudar o comportamento dos diferentes tipos de agentes quando submetidos a simulações com parâmetros distintos. Por exemplo, poderíamos averiguar qual tipo de agente consegue sair mais rápido do ambiente, ou até mesmo aquele que consegue maior taxa de sobrevivência. Sendo assim para obter tais resultados é necessário realizar uma série de simulações possuindo configurações específicas para o que se deseja medir.

Para ratificar o estudo comportamental dos agentes foram agregadas duas funcionalidades importantes: a possibilidade de repetir uma mesma simulação diversas vezes, e o sistema de "log" para coleta dos resultados parciais e finais de cada simulação.

A possibilidade de realizar diversas simulações com a mesma configuração foi algo essencial para aumentar o grau de confiança dos resultados. Vale ressaltar que se fixarmos todos os parâmetros os resultados obtidos seriam sempre os mesmos, por isso para que um determinado estudo tenha sentido é necessário variar algum parâmetro, por exemplo, o número de focos ou o número de agentes.

O sistema de "log" consiste na gravação de todas as informações de variáveis ao longo da simulação e o resultado final, em um arquivo texto. É possível observar o comportamento do sistema multiagente a cada ciclo de simulação. Assim, a principal utilidade desse arquivo texto é extrair grandes quantidades de informação sem precisar acompanhar cada simulação.

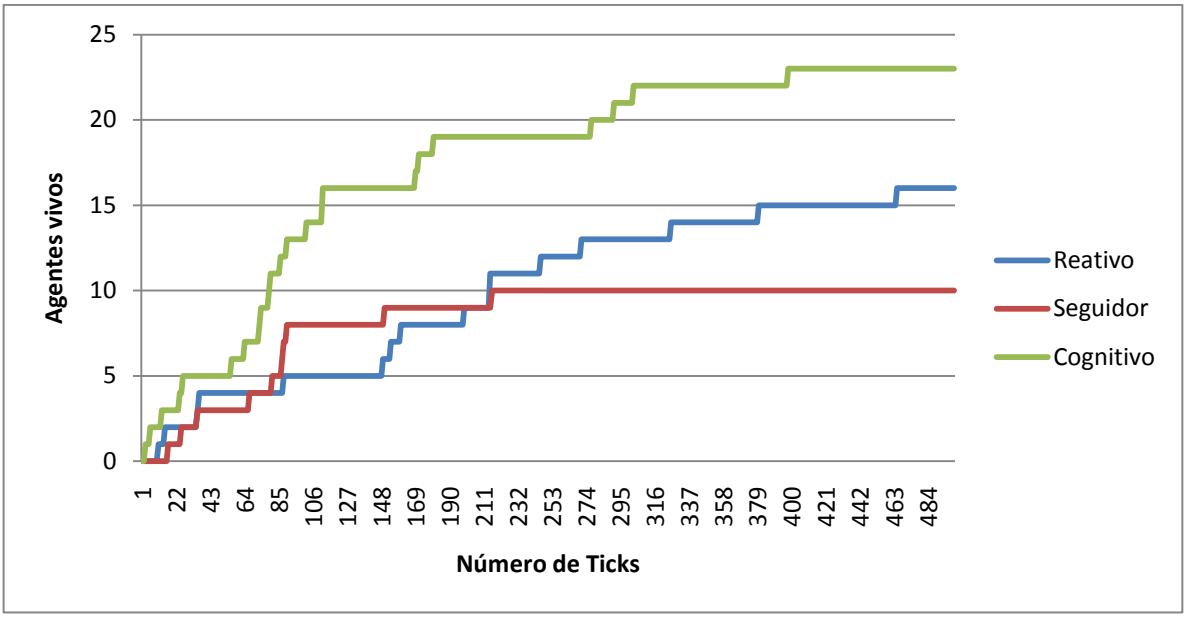
Todos os estudos descritos nos próximos tópicos tiveram como premissas: repetição e coleta de dados através de "logs" gerados por cada grupo de simulação.

8. Resultados

Para analisar o ambiente e o desempenho dos agentes sob diversos aspectos, realizamos várias simulações com parâmetros variados para que pudéssemos analisar os resultados através de gráficos. Realizamos ao todo quatro análises diferentes. No tópico 8.1 mostraremos o decorrer de uma simulação e o número de agentes sobreviventes em relação ao tempo. No tópico seguinte, 8.2, veremos o comparativo de desempenho entre os tipos de agentes em simulações com variados números de focos. O tópico 8.3 mostrará um comparativo semelhante ao anterior, porém com a quantidade de agentes variando proporcionalmente. No último tópico, 8.4, analisamos o desempenho de agentes reativos comparados com o desempenho de agentes seguidores em situações semelhantes e estudamos a influência dos agentes cognitivos nestas simulações.

8.1. Simulação trivial

Mostraremos a seguir o desempenho dos diversos tipos de agentes em termos do número de sobreviventes ao longo de uma simulação. Assim é possível verificar a morte de agentes no decorrer do tempo, e o desempenho de cada tipo de agente ao final da simulação, quando o fogo já se alastrou por todas as salas e alguns agentes conseguiram sobreviver enquanto outros morreram. O comportamento descrito pode ser visto na Figura 25



Erro! Fonte de referência não encontrada.

Esse tipo de gráfico é importante quando queremos medir o desempenho dos agentes ao longo do tempo, ou por ticks, nossa unidade de medida de tempo.

8.2. Fuga do fogo

A análise a seguir teve como motivação o estudo do desempenho dos três tipos de agentes quando o número de focos de incêndio aumenta. Para realizar essa análise os dados foram coletados a partir de simulações onde foram fixados a quantidade de agentes, sendo 30 agentes por cada tipo e obviamente 90 no total.

É importante ressaltar que o posicionamento de cada agente e de cada foco no ambiente ocorreu de forma aleatória, logo, em cada execução a configuração inicial do ambiente era diferente, excetuando-se as variáveis fixadas por nós.

A cada conjunto de simulações o número de focos de incêndio foi incrementado. Inicialmente as simulações foram realizadas sem focos com o objetivo medir o desempenho dos agentes sem a interferência do incêndio. O número de focos foi incrementado até que a quantidade de sobreviventes não apresentasse variações significativas.

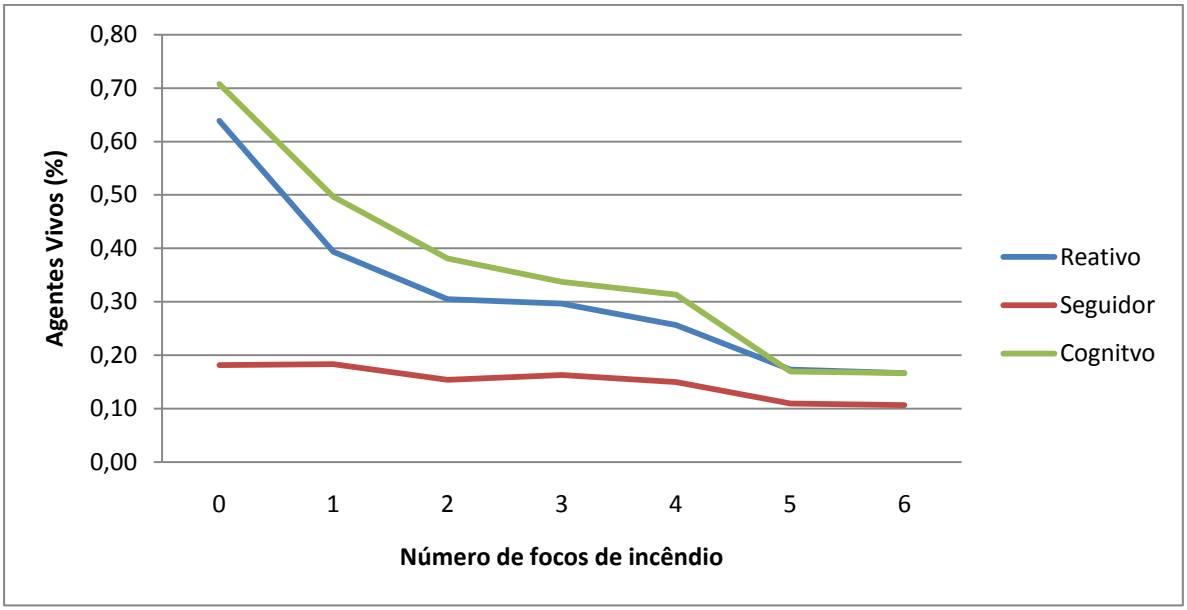


Figura 25 – Fuga do foco

Como podemos observar na o desempenho dos agentes, independentemente do seu tipo, diminuiu com o aumento do número de focos. Este comportamento era esperado, pois o maior número de focos aumenta a probabilidade de contato dos agentes com o fogo, causando um maior número de mortes.

O agente cognitivo apresentou um desempenho melhor em todas as simulações, visto que este tipo de agente armazena o histórico de salas visitadas com a intenção de explorar as regiões ainda não conhecidas por ele. Este fato aumentou a probabilidade do agente cognitvo encontrar a saída global do ambiente. Por outro lado, os agente reativos por não possuírem tal capacidade ficam em média mais tempo no ambiente, e por consequência disso, mais tempo expostos ao avanço do incêndio.

Devido a especificação do agente seguidor como um seguidor “puro”, onde ocorrem sucessivas trocas de agentes alvo, o tempo médio de permanência em cada sala é maior se comparado aos demais tipos de agentes. Assim os seguidores apresentaram uma taxa de sobrevivência menor.

8.3. Comparativo com número de agentes variando

A análise a seguir teve como motivação o estudo do desempenho dos três tipos de agentes quando aumentamos o número de agentes. Para realizar essa análise os dados foram coletados a partir de simulações onde foi fixado o número de focos.

É importante ressaltar que o posicionamento de cada agente e de cada foco no ambiente ocorreu de forma aleatória, logo, em cada execução a configuração inicial do ambiente era diferente, excetuando-se as variáveis fixadas por nós.

A cada conjunto de simulações o número de cada tipo de agente foi incrementado de dez. Inicialmente, as simulações foram realizadas com dez agentes de cada tipo objetivando medir seus desempenhos com uma população pequena. O número de agentes foi incrementado até que a quantidade máxima suportada pelo ambiente.

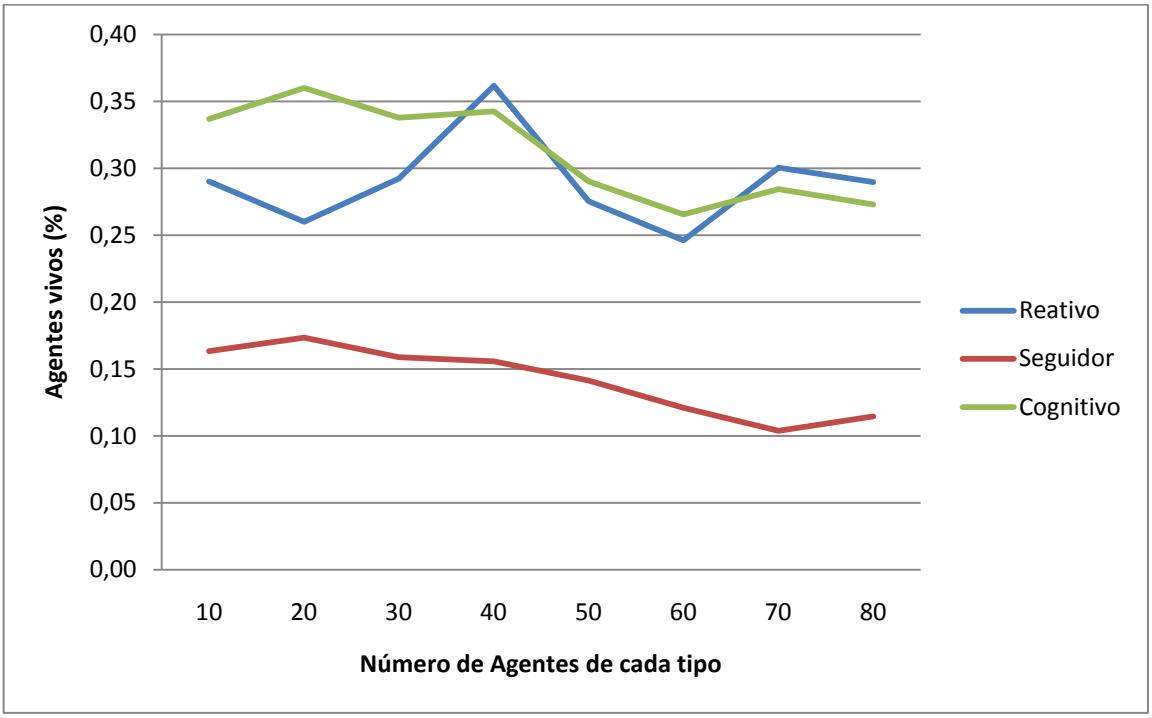


Figura 26 – Comparativo número de agentes

Como ilustra a Figura 26 não há uma relação clara de desempenho entre os três tipos agentes. Neste quadro esperávamos um bom desempenho do agente cognitivo visto que quanto maior o número de cognitivos maior seria a quantidade de informação trocada entre eles e, por conseguinte mais rápido eles encontrariam a saída global do ambiente. Acreditamos que o número elevado de agentes acabou por não evidenciar isto. Grandes quantidades de agente talvez tenham refletido em maiores colisões ou até mesmo mais perda de tempo na comunicação no caso cognitivo. Para o agente seguidor vale a mesma premissa, das trocas sucessivas de agentes alvo.

8.4. Comparativo reativos x seguidores

Nesta análise esperávamos obter informações sobre o comportamento de certos agentes quando estes interagiam somente com agentes cognitivos durante uma simulação. Para um melhor entendimento chamaremos estes agentes de grupo de controle. Um grupo de controle é composto por 15 agentes de um mesmo tipo interagindo com diversos agentes cognitivos. Para obter os dados necessários à análise, realizamos diversos conjuntos de simulações com o grupo de controle sempre do mesmo tamanho, enquanto o grupo de agentes cognitivos aumentava em quantidade.

Primeiramente realizamos as simulações referentes ao primeiro grupo de controle, somente com agentes reativos. Depois realizamos a bateria de simulações correspondente ao segundo grupo de controle, somente com agentes seguidores. Após cada conjunto de simulações de uma mesma bateria, a quantidade de agentes cognitivos foi aumentada em 10 agentes. Assim, colocamos em um mesmo gráfico, os resultados da bateria de simulações dos agentes reativos com os da bateria dos agentes seguidores.

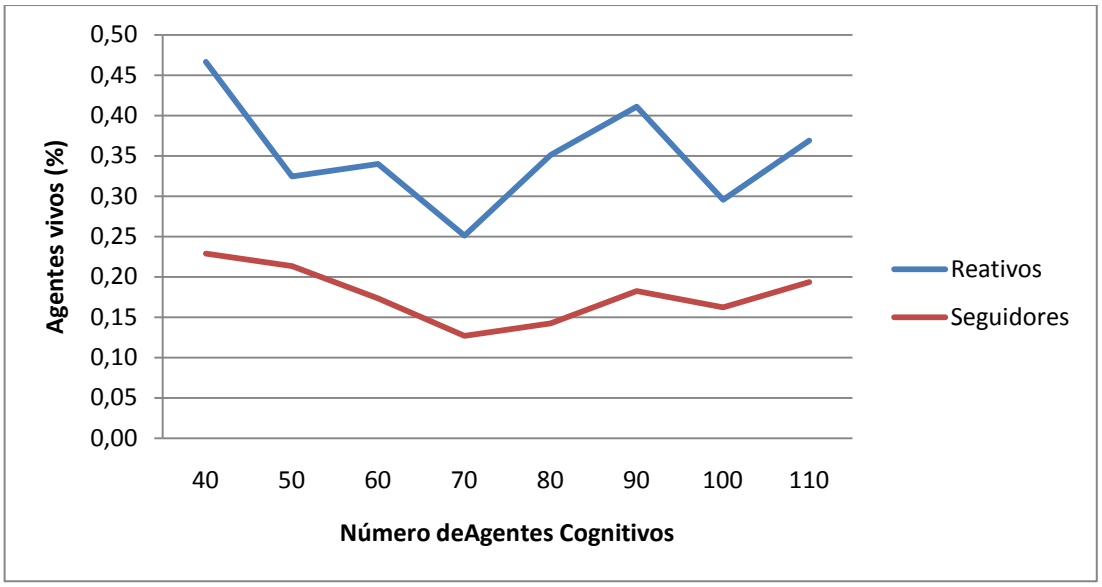


Figura 27 – Comparativo reativo e seguidor

Como podemos perceber na Figura 27 não há uma relação clara de desempenho entre os agentes dos grupos de controle e a quantidade de agentes cognitivos nas simulações. Tanto nas simulações com reativos quanto nas com seguidores. Acreditamos que isto se deve ao fato da característica purista na implementação do agente seguidor. Caso o seguidor fosse mais fiel

aos seus “alvos” ele poderia se beneficiar da maior inteligência dos agentes cognitivos e melhorar seu desempenho individual encontrando a saída de forma mais rápida. Com isso, acreditamos que seu desempenho poderia superar o desempenho dos agentes reativos a partir de um certo número de agentes cognitivos na simulação.

Considerações finais

Neste trabalho expandimos uma implementação do problema de fuga utilizando a plataforma NetLogo como sistema multiagente. Implementamos também alguns tipos de agentes com características específicas a fim de estudar seu comportamento e desempenho em simulações com diversas configurações diferentes.

No decorrer do projeto percebemos a necessidade de refatorar partes do sistema já implementadas, para que pudéssemos progredir com nossos objetivos de forma satisfatória. Consideramos que a decisão de refatorar partes já prontas do código-fonte, e muitas vezes refazê-las, foi uma decisão acertada. Apesar disso, optamos por não refazer a implementação do ambiente (salas e portas) por ser uma parte onde havia muita dependência com outras funcionalidades do código, assim não haveria tempo hábil para tal atividade.

A análise dos resultados nos mostrou informações importantes sobre o desempenho e relacionamento entre os três tipos de agentes, assim como nos deu diversos dados de retorno sobre a implementação da simulação e seu peso na atuação de cada tipo de agente.

Como opção para trabalhos futuros, achamos importante a criação de um novo tipo de agente seguidor, com uma cognição maior que o seguidor “puro”, podendo ter uma implementação que demonstre uma maior “fidelidade” com o alvo escolhido no decorrer da simulação. Esta opção traria o benefício de uma melhor comparação de seu desempenho quando existem mais agentes cognitivos no ambiente. Outra opção seria a melhoria do simulador, com uma refatoração do código na criação do ambiente, para que mais agentes pudessem ser adicionados na simulação sem que isso causasse lentidão no sistema. Esta melhoria do simulador poderia também ser feita no que diz respeito a um ajuste fino das penalidades impostas aos agentes ao andar, por exemplo, ou quando é atingido pelo fogo ou mesmo quando se comunica com outros. Consideramos também que uma melhor análise dos efeitos da comunicação entre agentes poderia ser feita em futuros trabalhos.

Este trabalho foi realizado no decorrer do último ano na graduação, e consideramos que o mesmo foi de grande valia para o nosso aprendizado na área de inteligência artificial,

assim como em sistemas multiagentes. Também acreditamos que ele foi de grande importância para que pudéssemos obter uma maior experiência em metodologia de pesquisa científica.

Bibliografia

[Russel, S., 2003] Russel, Stuart; Norvig, Peter. Artificial Intelligence, 2003.

[Fowler, M., 1999] Fowler, Martin. Refactoring: Improving the design of existing code, 1999.

[Martin, R., 2008] Martin, Robert C. Clean Code: A handbook of agile software craftmanship, 2008.

[Ferreira, Y., 2009] Ferreira, Yuri. Um estudo investigativo sobre a comunicação em simulações de emergência e evacuação, 2009.

[NetLogo, 2009] NetLogo user manual, v.4.1, 20 de dezembro, 2009.

[Uri Wilensky, 2010] NetLogo home page, http://ccl.northwestern.edu/netlogo/. Acessado em: 03/2010.

[Wikipedia, 2010] http://en.wikipedia.org/wiki/NetLogo. Acessado em: 08/2010.

[NetLogo, 2010] http://ccl.northwestern.edu/netlogo/docs/dictionary.html. Acessado em: 09/2010.

9. Apêndice A

Os códigos fonte apresentados aqui correspondem às principais funções dos processos de decisão dos agentes desenvolvidos e estão descritos na linguagem NetLogo [NetLogo, 2009]. Os termos exibidos na cor azul são palavras reservadas e funções nativas da linguagem.

9.1. Código fonte do agente Reativo

```
to processoDecisaoReativo
 set salaAtual (encontraSala xcor ycor salaAtual)
 if (chegouNaSaida)
 [
  matarOAgente 1    ; Se o agente chegou na saída, ele morre e entra na
                    contabilidade de agentes salvos.
 ]

 if (lifetime = 0) [matarOAgente 3]
 let IdPortaSaida verificaSaida

 ; Se a sala tem saída, anda em direção à saída.
 ifelse (IdPortaSaida != -1)
 [
  let portaDeSaida (one-of doors with [idPorta = idPortaSaida])
  face portaDeSaida
 ][
   escolheProximaPortaReativamente
 ]
end

to escolheProximaPortaReativamente
   let minhaSala salaAtual
   let temFogo   ([temFogo?] of minhaSala)
   let portaProxima (min-one-of doors with [salaPorta = minhaSala][distance
                                                          myself])
   ifelse ((portaEscolhida != nobody) and (distance portaEscolhida) < 0.5) [
      let portaTemp nobody
      let pontoMaisAFrente (patch-ahead (tamanhoParede * 2 / 3))
      if (pontoMaisAFrente != nobody) [
         let idPortaAtual ([idPorta] of portaProxima)
```

```
            let portaGeminada (min-one-of doors with [idPorta != idPortaAtual]
                                                    [distance portaProxima])
            let x [pxcor] of pontoMaisAFrente
            let y [pycor] of pontoMaisAFrente
            let sala (encontraSala x y nobody)
            if (sala != nobody) [
              let listaPortas ([listDoors] of sala)
              set listaPortas (remove portaGeminada listaPortas)
              let nPos random (length listaPortas)
              set portaEscolhida (item nPos listaPortas)
            ]
          ]
        ][
          if (portaEscolhida = nobody) [
            set portaEscolhida (min-one-of (doors in-cone (tamanhoParede * 2) 140)
                                  with [(salaPorta != minhaSala)][distance myself])
          ]
        ]
        ifelse (temFogo) [ ; Se a sala tem fogo, foge do foco
          processoFugirFogo
        ][
           ifelse (portaEscolhida != nobody)  ; Se a sala tem pelo menos uma porta, vai
                                                para a porta mais próxima. [
            face portaEscolhida
          ][
            ; Se a sala não tem porta, anda aleatoriamente.
            let direcao random 360
            right direcao
          ]
        ]
      end
```

9.2. Código fonte do agente Seguidor

```
      to processoDecisaoSeguidor
        set salaAtual (encontraSala xcor ycor salaAtual)
        if(chegouNaSaida)
        [
          matarOAgente 1    ; Se o agente chegou na saída, ele morre e entra na contabilidade
                                                              de agentes salvos.
        ]

        if (lifetime = 0) [matarOAgente 3]
        let IdPortaSaida verificaSaida

        ifelse (IdPortaSaida != -1) [       ; Se a sala tem saída, anda em direção à saída.
          let portaDeSaida (one-of doors with [idPorta = idPortaSaida])
          face portaDeSaida
        ][
```

```
    let seguidorAtual (escolheSeguidorMaisPerto(salaAtual))
    let bMesmaSala (verificaSalaSeguidorAntigo)
    ifelse (bMesmaSala = true)
    [
      linkarComSeguidor(seguidorAtual)
    ][
      ask my-links[die]
      if(seguidorAtual != nobody)
      [
        let distanciaSeguidorAntigo (obterDistanciaDoAgente(seguidor))
        let distanciaSeguidorAtual (obterDistanciaDoAgente(seguidorAtual))

        ifelse(distanciaSeguidorAntigo >= distanciaSeguidorAtual)
        [
          linkarComSeguidor(seguidorAtual)
        ][
          escolherPortaDoSeguidor
        ]
      ]
    ]

    if(seguidor = nobody)
    [
      escolheProximaPortaReativamente
    ]
  ]
end
```

9.3. Código fonte do agente Cognitivo

```
to processoDecisaoCognitivo
  if(chegouNaSaida)[
    matarOAgente 1    ; Se o agente chegou na saída, ele morre e entra na contabilidade
                                                          de agentes salvos.
  ]

  let sala (encontraSala xcor ycor salaAtual)

  if (sala != salaAtual) [ ; Atualizando a sala atual e guardando um histórico de salas
                                                          conhecidas.
    set salaAtual sala
    set salasConhecidas (lput sala salasConhecidas)
    set salasConhecidas (remove-duplicates salasConhecidas)
    set tempoMesmaSala ticks
  ]

  if(lifetime = 0) [matarOAgente 3]

  ifelse (not empty? plano) [
```

```
    executaAcaoPlano
  ][
    planejadorCognitivo
  ]
end

to executaAcaoPlano
  if (salaAtual != nobody)
  [
    let temFogo   ([temFogo?] of salaAtual)
    if (temFogo) [
      set plano []
      processoFugirFogo
      stop
    ]
  ]

  let acao (first plano)
  let tipoAcao (first acao)

  ifelse (tipoAcao = "p")[
    let alvo (last acao)

    ifelse (alvo != nobody)[
      ifelse (distance alvo < 0.5) [   ;Chegou no alvo, concluiu a ação.
        set plano (remove-item 0 plano)

        let acaoAndar []
        set acaoAndar (lput "andar" acaoAndar)
        set acaoAndar (lput (patch-ahead 2) acaoAndar)
        set plano (fput acaoAndar plano)
      ][
        face alvo
      ]
    ][
      set plano (remove-item 0 plano)
    ]
  ][
    ifelse (tipoAcao = "s")
    [
      let alvo (last acao)
      if (alvo != nobody) [
        let bPossoSeguir (possoSeguirAlvo alvo)
        let bConhecoPoucasSalas  ((length salasConhecidas) < 3)

        ifelse (bPossoSeguir and bConhecoPoucasSalas)
        [
          let salaAlvo ([salaAtual] of alvo)
          ifelse (salaAtual = salaAlvo)
          [
```

```
          face alvo  ; Vou seguir o alvo.
         ][
          ; O alvo mudou de sala
           let portaAlvo (min-one-of doors with [salaPorta = salaAlvo][distance alvo])
          set acao []
          set acao (lput "p" acao)
          set acao (lput portaAlvo acao)
           set plano (fput acao plano)  ; Vou passar pela mesma porta que o alvo
                                          passou, e depois vou segui-lo.
         ]
        ][
         if ((length plano) > 0) [
          set plano (remove-item 0 plano)
          ask my-links [die]
         ]
        ]
      ]
     ][
      if (tipoAcao = "andar")
      [
       let alvo (last acao)
       ifelse (distance alvo < 0.5) [     ;Chegou no alvo, concluiu a ação.
        set plano (remove-item 0 plano)
       ][
        face alvo
       ]
      ]
     ]
    ]
end

to planejadorCognitivo
  let idPortaSaida -1
  ifelse (saidaEncontrada = -1) [
   set idPortaSaida verificaSaida
  ][
    set idPortaSaida saidaEncontrada
  ]

  ifelse ((idPortaSaida != -1) ) [ ; Se conheço a saída...
    let portaDeSaida (one-of doors with [idPorta = idPortaSaida])
    let acao []
    set acao (lput "p" acao)
    set acao (lput portaDeSaida acao)
    set plano (lput acao plano)
  ][
    let numSalasConhecidas (length salasConhecidas)

    ifelse (numSalasConhecidas < 3)
    [
```

```
    let nEscolha (random 2)

    if (nEscolha = 0) [ ;Segue o agente mais próximo
      let minhaSala salaAtual
      set seguidor (min-one-of peoples with [salaAtual = minhaSala and
                                             self != myself][distance myself])

      if (seguidor = nobody) [ ; Se não há mais ninguém na minha sala, muda a
                                                                  escolha.
        set nEscolha 1
      ]
    ]

    ifelse (nEscolha = 0) ;Segue o agente mais próximo.
    [
      processoDecisaoSeguir
      let acao []
      set acao (lput "s" acao)
      set acao (lput seguidor acao)
      set plano (lput acao plano)
    ][
       ;Escolhe uma porta aleatoriamente.
      set portaEscolhida nobody
      let listaPortas ([listDoors] of salaAtual)
      let salaTemp salaAtual
      let portaMaisProxima (min-one-of doors with [self != salaTemp]
                                                  [distance myself])
      ;Excluindo a porta de onde vim...
      if (portaMaisProxima != nobody) [
        if (distance portaMaisProxima < 0.5) [
          let idPortaMaisProxima ([idPorta] of portaMaisProxima)
          set listaPortas (remove idPortaMaisProxima listaPortas)
        ]
      ]
      ;;;;;;;;;;;;;;

      let nPos random (length listaPortas)
      set portaEscolhida (item nPos listaPortas)
      let portaTemp portaEscolhida
      if (portaTemp != nobody) [
        let portaAdjacente (min-one-of doors with [self != portaTemp]
                                                  [distance portaTemp])
        let acao []
        set acao (lput "p" acao)
        set acao (lput portaAdjacente acao)
        set plano (lput acao plano)
      ]
    ]
  ][
    ifelse (numSalasConhecidas < 8)
```

```
  [
    gerarRota 1 ; Escolhe uma porta que leve a uma sala desconhecida. Se todas são
                                          conhecidas escolhe uma aleatoriamente.
  ][
    ifelse (numSalasConhecidas < 15)
    [
      let t ticks
      let minhaSala salaAtual
      let agenteComm (one-of peoples with [(salaAtual = minhaSala) and (self !=
                                                                    myself)])
      let ultimaComm (t - ultimaComunicacao) ; última vez que alguém me pediu
                                                                    comunicação.
      ifelse ( ultimaComm > 10 and (t - tasked) > 3 and (agenteComm != nobody))
      [
        set commCL commCL + 1
         let salasOutroAgente ([salasConhecidas] of agenteComm)

        ; Verificando se uma das salas conhecidas é a saída.
        foreach salasOutroAgente
        [
          let portaDeSaida one-of doors with [(salaPorta = ?) and saidaPorta? = true]
          if (portaDeSaida != nobody) [
            set saidaEncontrada ([idPorta] of portaDeSaida)
          ]
        ]

        ; Adquirindo conhecimento...
        foreach (salasOutroAgente) [
          set salasConhecidas (lput ? salasConhecidas)
        ]
        set salasConhecidas (remove-duplicates salasConhecidas)
        decrementaHP 2
        ;;;;;;;;;;

        set tasked t   ;t is the current iteration
      ][
        ; Não tem agente pra perguntar, busca regiões de borda.
        gerarRota 2
      ]
    ][
      if (numSalasConhecidas >= 15)
      [
         ; Escolhe uma porta que leve a uma sala desconhecida. Se todas são
                                          conhecidas escolhe uma aleatoriamente.
        gerarRota 1
      ]
    ]
  ]
 ]
]
```

```
end
```

10. Apêndice B

As simulações criadas para NetLogo têm normalmente códigos entre 100 e 300 linhas, um valor muito pequeno, mesmo para os sistemas mais simples [Uri Wilensky, 2010]. Nestas dimensões o código não necessita de muito esforço para se manter "limpo", e a manutenção é barata e fácil. Provavelmente por esse motivo o ambiente de programação do NetLogo não tenha muitos recursos de busca, depuração e modificação de código, porém, quando simulações maiores são desenvolvidas fica muito mais difícil realizar operações de manutenção no código.

Ao final deste trabalho, o código fonte do simulador já continha mais de 4.000 linhas de código. Na seção seguinte descrevemos os principais problemas encontrados durante o desenvolvimento do sistema e as medidas tomadas para facilitar modificações e simplificar o entendimento do código.

10.1. Refatoração

Refatoração pode ser definido como o processo de modificar código existente, melhorando sua modelagem, sem que o comportamento do sistema seja alterado. Uma definição mais precisa pode ser encontrada em [Fowler, 1999], como segue: "Uma mudança feita na estrutura interna do software a fim de torná-lo mais fácil de entender e mais barato de modificar sem que o seu comportamento observável mude."

Diversos trabalhos têm sido feitos a respeito de refatoração e seu relacionamento com a produtividade de uma equipe no desenvolvimento de sistemas de software. Neste tópico falaremos sobre refatoração em NetLogo, os problemas enfrentados e as soluções escolhidas durante este trabalho.

A linguagem NetLogo é uma linguagem basicamente procedural, e todo o código é manipulado em um mesmo espaço, não podendo ser modularizado em vários arquivos. Assim, não existe uma forma rígida para agrupar funções semelhantes em espaços próprios, esta característica é um dos pontos que dificulta o desenvolvimento de sistemas maiores em

NetLogo, assim como a aplicação de técnicas de refatoração. Algumas linguagens oferecem ferramentas para a devida separação de funções semelhantes, como packages, em Java, e namespaces, em C++ e XML, o que não acontece em NetLogo, por isso, quanto mais funções o sistema tiver, mais complicado se torna de buscar um determinado algoritmo. Ao mesmo tempo, quanto menos funções existir mais difícil fica de modificar e reutilizar código. A solução escolhida por nós foi encontrar um meio termo entre essas duas estratégias.

Como não existe uma forma eficiente de depurar o código em NetLogo nós decidimos por manter as funções no menor tamanho possível, para que fosse mais fácil encontrar e corrigir bugs e realizar modificações no código, e para que a busca de algoritmos não ficasse prejudicada decidimos agrupar funções semelhantes o máximo possível no decorrer do código-fonte, como descrito em [Martin, R., 2008]. Muitas vezes a estratégia de agrupar funções semelhantes não funciona por se tratar de um arquivo só, porém nós tentamos sempre manter esta idéia em mente e aplicá-la o máximo possível.

Como parte do código da simulação já estava feito quando começamos o projeto, a primeira atitude tomada por nós foi de tentar entendê-lo o melhor possível para poder começar a fazer modificações. Uma tarefa inicial e produtiva que poderíamos assumir durante esta fase de compreensão do código-fonte seria a construção de testes automatizados. Com esta atitude não só entenderíamos melhor o código conforme fôssemos criando os testes como também construiríamos uma base de testes aumentando assim a confiabilidade e também a nossa segurança ao realizar futuras modificações no que já estava pronto. Porém, o NetLogo não oferece suporte para a criação e execução de testes, então o que fizemos foi entender e comentar o máximo possível o código pra que as informações aprendidas não caíssem no esquecimento.

Durante o desenvolvimento do projeto tivemos que tomar algumas decisões sobre o uso de código já existente. Muitas vezes optamos por inutilizar funções grandes e complexas para recriar seu comportamento na forma de diversas funções menores e mais simples, como foi o caso dos algoritmos de raciocínio dos agentes. A computação do raciocínio dos agentes estava concentrada em uma função principal, que tratava do comportamento comum a todos os agentes e do comportamento específico de cada tipo de agente. Durante o processo de desenvolvimento nós criamos funções específicas para cada tipo de agente, extraindo funções comuns sempre que possível, para que elas pudessem ser reutilizadas. Outras vezes optamos por reutilizar funções já existentes modificando-as sempre que necessário para adequá-las a um diferente comportamento ou para simplificá-las.

Como o NetLogo compõe um ambiente mais restrito que os de linguagens comerciais, percebemos que o conjunto de refatorações que poderíamos aplicar no sistema também era bastante limitado. Durante o desenvolvimento nos concentramos em aplicar as seguintes técnicas de refatoração, em ordem de uso:

- **Extrair método:** Criar um novo método a partir de parte de outro método. Esta refatoração ajudou a diminuir e simplificar métodos.
- **Substituir algoritmo:** Alguns algoritmos usados são computacionalmente custosos. Mudar algoritmos muitas vezes foi uma necessidade para otimizar a execução das simulações.
- **Introduzir variável explicativa:** Muitas vezes etapas de um processamento são feitas sem uma devida separação de cada parte e sem comentários para ajudar no entendimento do código. Nesses casos é necessário criar variáveis intermediárias com nomes explicativos.
- **Adicionar parâmetro:** Na linguagem NetLogo muitas vezes são usadas variáveis globais no processamento de uma função. Para facilitar a programação e a depuração preferimos evitar esta estratégia assando tais variáveis como parâmetros das funções.
- **Renomear método:** Dar um novo nome a um método é muitas vezes necessário para aumentar a legibilidade do código e evitar o uso indevido de métodos por simples confusão.

No processo, outras técnicas de refatoração foram utilizadas em menor quantidade. Os nomes das técnicas de refatoração aqui apresentados foram extraídos e traduzidos da referência [Fowler, 1999].

O processo de refatoração do código aconteceu de forma simultânea com a implementação de novas funcionalidades. Sempre que era necessário modificar o código existente para possibilitar a implementação de novas funcionalidades ou a correção de erros no programa aplicávamos alguma técnica de refatoração. Assim, durante o projeto, não definimos períodos em que deveríamos somente refatorar o sistema parando a implementação de novos comportamentos. Esta decisão foi tomada alinhada nos trabalhos de diversos pesquisadores desta área, como Kent Beck, William Opdyke, Steve Freeman, e descrito nas seguintes referências: [Fowler, 1999] [Martin, 2008].

Consideramos que, apesar das limitações existentes na linguagem e no ambiente de desenvolvimento, o uso destas técnicas nos possibilitou realizar modificações com maior segurança e maior rapidez.