Universidade Federal de Pernambuco
Centro de Informática

Patrulha Multiagente: Uma Análise Empírica e Sistemática

por

Aydano Pamponet Machado

Dissertação de Mestrado

Recife, Setembro/2002.

Aydano Pamponet Machado

Patrulha Multiagente:
Uma Análise Empírica e Sistemática

Este trabalho foi submetido à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Geber Lisboa Ramalho

Recife, Setembro/2002.

“On ne voit bien qu'avec le coeur. L'essentiel est invisible pour les yeux.”

Antoine de Saint-Exupéry

Dedicação

Aos meus pais Robério Dias Machado e Ana Lúcia Pamponet Machado.

Agradecimentos

Gostaria de começar agradecendo a todos que me ajudaram ou contribuíram de forma direta ou indireta para a realização deste trabalho, bem como para a minha formação pessoal ou acadêmica.

Em especial a minha família, meu pai Robério Dias Machado, minha mãe Ana Lúcia Pamponet Machado e meu irmão Alison Pamponet Machado pelo amor, dedicação, incentivo, e por me propiciarem tudo para chegar onde estou.

A meu orientador Geber Lisboa Ramalho pela amizade, incentivo e dedicação, além da orientação que me mostrou uma nova maneira de encarar os problemas, criticar e defender as devidas soluções.

A minha namorada Janiffer Miranda Lacet Vieira pelos momentos felizes que me ajudaram nesta jornada.

Aos amigos e a nova família que aqui fiz, pessoas estas que me acolheram, me deram abrigo e me ajudaram de todas as maneiras. Sempre os guardarei no coração.

A todos que contribuíram para o cumprimento do meu curso, desenvolvimento deste trabalho, confecção desta dissertação e apresentação da mesma.

Ao Cin/UFPE e a CAPES pela estrutura, suporte e financiamento, fundamentais para a concretização desse trabalho.

A quem infelizmente eu esqueci de citar e agradecer, o meu

MUITO OBRIGADO.

Resumo

Um grupo de agentes pode ser usado para realizar tarefas de patrulhamento em uma variedade de domínios, desde administração de redes de computadores a simulações de jogos de computadores. Apesar de seu alto grau de aplicabilidade, arquiteturas multiagentes para patrulhamento ainda não foram profundamente estudadas. As abordagens do estado da arte usadas para lidar com problemas correlatos não podem ser adaptadas facilmente à especificidade da tarefa de patrulhamento. Além disso, as abordagens específicas para patrulhamento ainda estão em estágio preliminar. O trabalho aqui apresentado descreve uma discussão original sobre a tarefa de patrulhamento multiagente, assim como uma avaliação empírica de possíveis soluções. A fim de realizar esse estudo, propusemos diversas arquiteturas de sistemas multiagentes, alguns critérios de avaliação, dois cenários de experimentação e implementamos um simulador de patrulhamento. Os resultados mostram que tipo de arquitetura pode patrulhar uma área mais adequadamente de acordo com as circunstâncias.

Palavras-chaves: sistemas multiagentes, coordenação de agentes, patrulha e exploração de terrenos, jogos de computador.

Abstract

A group of agents can be used to perform patrolling tasks in a variety of domains ranging from computer network administration to computer wargame simulations. Despite its wide range of potential applications, multiagent architectures for patrolling have not been studied in depth yet. First state of the art approaches used to deal with related problems cannot be easily adapted to the patrolling task specificity. Second, the existing patrolling-specific approaches are still in preliminary stages. In this work, we present an original in-depth discussion of multiagent patrolling task issues, as well as an empirical evaluation of possible solutions. In order to accomplish this study we have proposed different architectures of multiagent systems, various evaluation criteria, two experimental scenarios, and we have implemented a patrolling simulator. The results show which kind of architecture can patrol an area more adequately according to the circumstances.

Keywords: Multiagent systems, coordination, patrolling, exploration, and computer games.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Equações

1
Introdução

"... Entre as ações de combate à dengue na cidade, segundo a Prefeitura do Recife, estão a intensificação dos agentes de saúde na inspeção do mosquito Aedes Aegypti cobrindo 100% da cidade..." ***(Diário de Pernambuco, 28 de Março de 2002)***.

"... Três crianças foram resgatadas em tempo recorde do Parque Estadual Mata Atlântica. A equipe de resgate, composta de cinco robôs REGX300-8, levou cinqüenta minutos para encontrar as crianças numa imensa área de mata fechada e de difícil acesso..." ***(Jornal Fictício, 23 de maio de 2020)***.

Estas duas notícias, sendo a segunda com um pouco de quimera, mostram que existem várias situações onde se precisa proteger, resgatar, procurar, detectar, supervisionar e/ou rastrear algo ou alguém. Geralmente, tais situações envolvem algum tipo de patrulha e são de importância e/ou de periculosidade para aqueles que estão participando e realizando tal tarefa.

Podemos também citar como alguns exemplos de situações com as características descritas anteriormente:

- A supervisão feita por um administrador de uma rede de computadores procurando falhas ou uma situação específica nesta rede [1];
- A detecção de modificações ou novas páginas na WEB a serem indexadas por engenhos de busca [8];
- A identificação de objetos ou pessoas em situações perigosas que devem ser resgatados [46], entre outras.

Para exemplificar uma das aplicações possíveis da patrulha, mostraremos duas situações retiradas de um jogo de computador (*Swat2*), que também poderiam ser circunstâncias reais sem perda ou acréscimo de qualquer característica essencial da conjuntura. Um armazém foi tomado por bandidos durante a fuga de um assalto a banco, mas ainda existem pessoas inocentes dentro do armazém. Na primeira figura (Figura 1.1) temos um grupo de policiais fazendo uma patrulha para encontrar os bandidos ou retirar as pessoas inocentes.

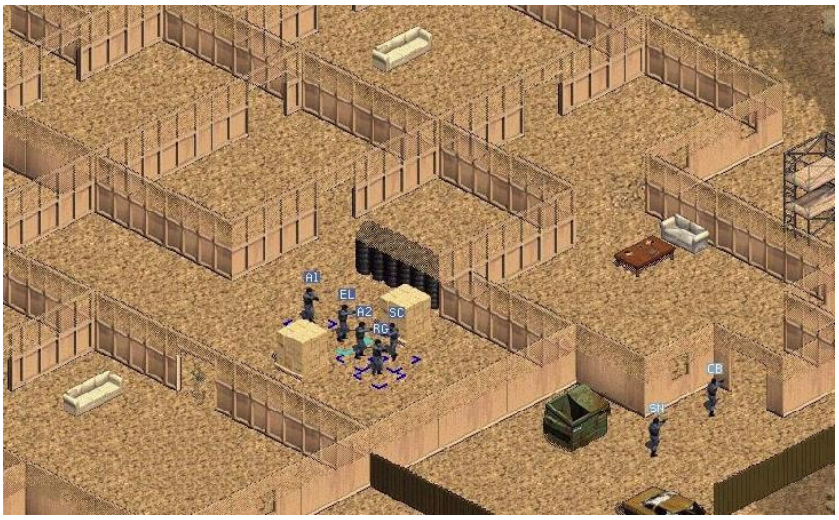


Figura 1.1 – Patrulha sendo realizada por um grupo de policiais. Só são mostrados os itens dos locais onde estão os policiais, assim podem existir bandidos e vitimas em outras salas. A1, A2, EL, RG, SG, CB e SH são os policiais.

Na segunda figura (Figura 1.2) é mostrado o resgate de uma mulher que foi encontrada durante a patrulha da região.

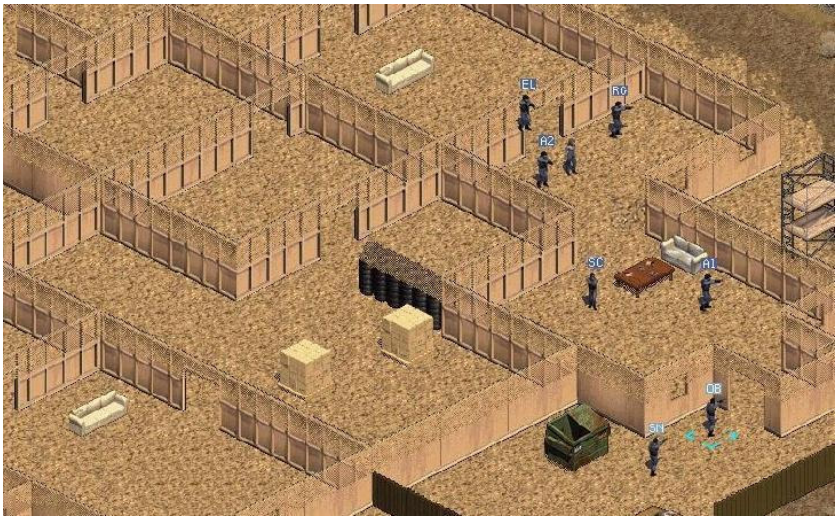


Figura 1.2 – Resgate de uma vítima sendo feito pelo grupo de policiais.

Nos dois casos, o objetivo é do grupo, embora as atitudes e comportamentos de cada um sejam executados de forma individual. Tudo que é planejado e realizado visa ao sucesso do grupo na operação. Isto é um ponto forte na caracterização de uma ação realizada por um grupo de indivíduos.

As idéias iniciais do presente trabalho surgiram durante o desenvolvimento de um jogo de combate entre naves chamado Canyon [32]. Durante esse desenvolvimento, apareceram vários problemas interessantes a serem estudados. Dentre eles, o patrulhamento de uma área com obstáculos, feito por um grupo de naves controladas pelo computador, serviu como motivação para a realização desta dissertação de mestrado. Tal problema se mostrou intrigante e, do que se tem notícia, inexplorado. De fato, na realização desta pesquisa, não foi encontrado estudo prévio que contemplasse tal assunto de uma maneira profunda e sistemática.

Patrulhar significa: “percorrer sistematicamente (uma área ou zona de passagem provável do inimigo), para detectá-lo e obter informes acerca da composição, atividades, etc., da área ou zona” [16]. Tal tarefa, quando executada por um grupo de patrulheiros, pode ser resolvida seguindo uma abordagem multiagente.

Além de muito útil para os simuladores e jogos, o esquema de solução aqui trabalhado pode ser reutilizado em várias outras aplicações de domínios diferentes onde seja necessário algum tipo de supervisão, inspeção ou controle distribuído. Por exemplo, agentes patrulheiros podem ser utilizados nas seguintes situações: por administradores de rede na supervisão de falhas ou de uma situação específica numa Intranet [1]; para detectar modificações ou novas páginas na WEB à serem indexadas por engenhos de busca [8]; para identificar objetos ou pessoas em situações perigosas que devem ser resgatados [46]; ou ainda por agentes de saúde para tratar os focos de doenças como a dengue, etc.

Apesar de sua grande aplicabilidade, a patrulha multiagente ainda não foi estudada adequadamente, conforme se observa na literatura da área de Sistemas Multiagentes (SMA) e outras áreas relacionadas, tais como: jogos de computador, redes de computadores e economia.

1.1. Objetivos

O objetivo principal deste trabalho de mestrado é fazer um estudo sistemático e metodológico do problema da patrulha multiagente, com o intuito de dar algumas diretrizes para implementação de soluções. Em outras palavras, não pretendemos propor a “melhor” arquitetura multiagente para esta tarefa, mas estudar uma variedade de arquiteturas multiagentes para melhor compreender suas propriedades em relação à patrulha.

1.2. Metodologia e Contribuição

Para atingir o objetivo de realizar um estudo sistemático e pioneiro do problema proposto, foi necessária uma definição mais formal e geral do problema, assim como e a utilização de uma metodologia de trabalho.

Tamanha é a pluralidade de aplicações da patrulha e as diferenças para cada caso que se faz necessária uma definição de maneira a representar essas classes de problemas. Para isso foi utilizada uma abstração, para o ambiente, criada através de um grafo. Essa abordagem generalizou o problema de forma adequada. Com isso não é difícil apontar quais tipos de problemas podem ser encarados como uma patrulha.

Após a definição do problema foi determinado como seria medido o desempenho de cada arquitetura de SMA durante a execução da patrulha, pois não existia nenhum trabalho anterior que sugerisse uma métrica para a situação aqui estudada.

Depois, foram propostas várias arquiteturas de SMA com diferentes números e características dos agentes. Essa combinação e variação dos parâmetros das arquiteturas propostas foram feitas de maneira a testar uma parte relevante dos SMA mais simples. Essas arquiteturas seguem o princípio de começar com as soluções (arquiteturas) mais simples para as mais complexas. Tal abordagem fortalece ainda mais a metodologia de trabalho adotada, além de mostrar os resultados com a evolução das arquiteturas de agentes.A variação da população (quantidade) dos agentes colocados em cada arquitetura mostrou como varia o comportamento de acordo com a relação entre o número de agentes e o tamanho da área a ser patrulhada.

Logo após, foram definidos alguns cenários que seriam patrulhados pelos agentes. Esses cenários foram determinados de maneira a proporcionar

diferentes tipos de situações para os agentes, e.g. diferentes níveis de conectividade entre os pontos do ambiente (nós do grafo).

Para realizar todos esses experimentos foi construído um simulador de maneira a representar todas as características e restrições encontradas num ambiente real que são relevantes para nossa pesquisa, além de coletar todas as informações necessárias para a análise realizada.

Com tudo isso definido, foram realizados experimentos, cujos resultados servem como reflexão e diretriz para o projeto de arquiteturas de SMA para a patrulha.

Como tal estudo ainda não tinha sido realizado, este trabalho é uma contribuição original e útil para o desenvolvimento de sistemas multiagentes para tarefas relacionadas ao escopo desse trabalho.

1.3. Estrutura da Dissertação

O restante da dissertação está estruturado da seguinte maneira. No capítulo 2 define-se o que é a patrulha. Esse capítulo também mostra a relevância deste problema para a sociedade, bem como algumas de suas possível aplicações.

A metodologia adotada durante todo o processo de estudo e análise do problema da patrulha é descrita no capítulo 3. Esse capítulo mostra como o problema foi tratado, descrevendo sucintamente cada item da metodologia.

O simulador é detalhado no capítulo 4, descrevendo seu funcionamento, além particularidades de implementação tais como: linguagem e bibliotecas utilizadas, descrição das classes, criação de novas arquiteturas de agentes e configuração do ambiente. Por fim, é mostrado como foi realizado o processo de coleta de dados dos experimentos.

O capítulo 5 traz os resultados e as discussões dos experimentos realizados, bem como, detalhes de como esses experimentos foram executados e como os dados foram preparados para a análise. Ainda são sugeridas algumas diretrizes preliminares que podem ajudar o projetista na construção de uma solução para um problema de patrulha multiagente.

No capítulo 6 descreve-se que trabalhos poderiam ser relacionados, de alguma forma, com o assunto em questão.

As considerações finais são colocadas no capítulo 7, onde também são apresentadas as contribuições deste trabalho. Assim como os possíveis trabalhos futuros.

2
Patrulhando

Várias tarefas envolvem algum tipo de patrulha e estas possuem características levemente diferentes de acordo com cada domínio e cada circunstância particular. Isso implica na necessidade de uma definição mais precisa e abrangente do que é uma patrulha para o estudo realizado neste trabalho.

2.1. Definindo o Problema

Para definir nosso problema, tomamos como base as técnicas de *skeletonization* utilizadas para alterar a representação de um ambiente [47] e [49]. Tais técnicas consistem em substituir um ambiente real por um grafo (esqueleto, daí vem o nome *skeletonization*) que reflete as características espaciais do ambiente, mostrando possíveis caminhos dentro deste ambiente. Um exemplo disso é mostrado na Figura 2.1. Essas técnicas, largamente utilizadas em robótica, onde são empregadas para facilitar a movimentação dos robôs.

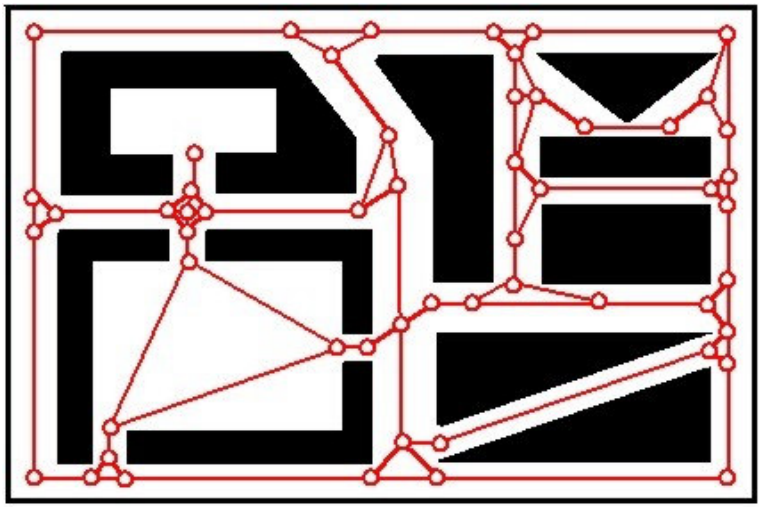


Figura 2.1 – Exemplo de uma abstração de um ambiente utilizando as técnicas de *skeletonization*. O grafo é mostrado com seus nós e arestas, sobre a região que ele esta representando. As partes escuras representam os obstáculos ou região não-navegável.

Existem várias maneiras de gerar o grafo que representa o terreno. São algumas delas: grafo de visibilidade, diagrama de Voronoi, C-cells, métodos de contorno [47]. Uma outra forma é a manual. Cada uma destas técnicas vai dar uma forma diferente para o grafo que é gerado, como por exemplo: deixar os nós do grafo próximos aos obstáculos, colocar os nós numa região central entre os obstáculos ou posicionar os nós de forma uniforme no terreno. A escolha de uma dessas técnicas, utilizada para gerar o grafo, é feita de acordo com o terreno e com a preferência da pessoa que esta querendo resolver o problema. Algumas delas serão detalhadas a seguir e exemplificadas graficamente na Figura 2.2.

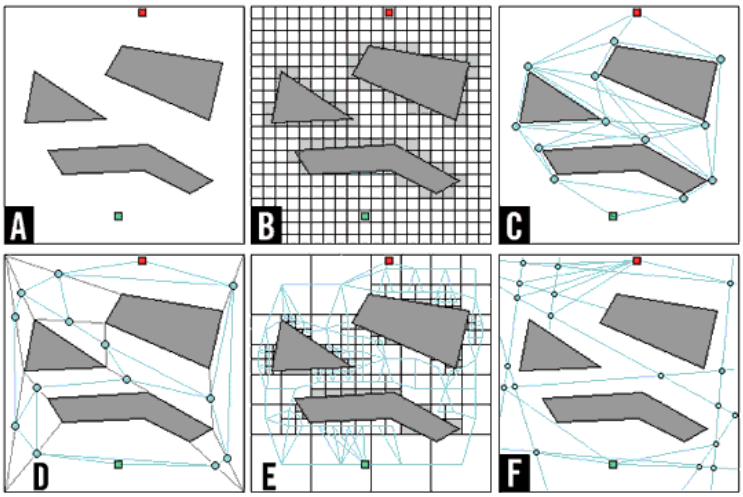


Figura 2.2 – Exemplos de abstrações para um ambiente.

A abordagem mais simples, chamada de *Tiles* (ladrilhos), consiste em utilizar uma grade para representar o ambiente. Os ladrilhos que estão parcialmente ou totalmente ocupados por obstáculos são marcados como não navegável (Figura 2.2-B).

Quando se quer construir um grafo de visibilidade são focados os chamados pontos críticos, situados nos vértices dos obstáculos, mas com espaço suficiente para evitar colisões. O grafo é criado colocando arestas entre os pontos que são visíveis entre si, isto é, se não existe nenhum obstáculo entre eles (Figura 2.2-C).

Existem outras técnicas que têm como princípio a divisão do espaço não ocupado por obstáculos em polígonos convexos. Os nós do grafo podem ser colocados dentro desses polígonos ou nas suas bordas. Os esquemas de divisão do espaço incluem: C-Cells (cada vértice é conectado ao vértice visível mais próximo) e Maximum-Area (cada vértice convexo de um obstáculo projeta arestas formando um vértice com os muros ou obstáculos mais próximos). Um exemplo dessas técnicas é mostrado na Figura 2.2-D.

O espaço não ocupado por obstáculos também pode ser dividido com a utilização de outras figuras, como é o caso de quadrados (Figura 2.2-E) e cilindros (Figura 2.2-F) . No primeiro exemplo os quadrados vão sendo divididos em quatro quadrados menores até que eles sejam homogêneos, ou seja, ocupados somente por obstáculos ou estejam totalmente navegáveis, nesse momento o pondo central de cada quadrado é utilizado como nó do grafo (Figura 2.2-E). Para o segundo, o espaço entre os obstáculos adjacentes é ocupado por um cilindro, quando todos os espaços estiverem ocupados pelos cilindros os eixos dos mesmos podem ser conectados, criando assim o grafo (Figura 2.2-F).

Com tal abstração para o ambiente disponível, os diferentes problemas de patrulha se tornam semelhantes, pois estes terão a mesma representação. De fato, uma grande vantagem da utilização de tal representação abstrata é que as diferentes soluções propostas podem ser aplicadas a vários tipos de problemas, indo desde a navegação num terreno até a navegação na *Web*.

Definição 1: Um grafo G = (V,E) é um conjunto não-vazio V, cujos elementos são chamados *vértices* ou *nós*, e um conjunto E de *arestas*. Uma aresta é um par não-ordenado (vi,vj), onde vi e vj são elementos de V.

Definição 2: Dado o grafo *G*, composto de *m* nós, representando o ambiente, a tarefa de patrulha consiste em visitar continuamente todos os nós n_i de *G* de maneira a minimizar o intervalo de tempo t_i entre duas visitas consecutivas a um mesmo nó n_i, onde $1 \leq i \leq m$.

Muitas variações podem ocorrer em um grafo a ser patrulhado. Por exemplo, em algumas situações onde o terreno possua obstáculos móveis, as arestas do grafo podem ser alteradas. Além disso, os pesos colocados nas arestas podem representar alguma característica que precise ser incorporada ao ambiente ou ao problema. Por exemplo, por se tratar de um grafo que

representa um terreno, esses valores podem corresponder à distância entre os nós ou ao custo de ir de um nó para o outro. Em outras situações, prioridades podem ser dadas para algumas regiões do grafo. Com isso, essas regiões serão mais intensamente patrulhadas por terem mais importância para determinada situação. Outras características também poderão ser incorporadas dependendo da necessidade.

Neste trabalho, nós estudamos a tarefa de patrulha para situações onde os grafos em questão tinham as seguintes características:

- Arestas estáticas: os ambientes em que foram realizados os experimentos não possuíam obstáculos móveis;
- Arestas de tamanho unitário: por questão de simplicidade nós colocamos primeiramente todas as arestas com o mesmo peso, implicando que a distância ou custo entre nós adjacentes é sempre a mesma;
- A patrulha é uniforme: não foi dada nenhuma prioridade a qualquer região do terreno.

2.2. Conclusões

O desenvolvimento de SMA capazes de realizar patrulha é relevante, dadas todas as atividades em que esta tarefa se insere. Tudo que envolva automação de supervisão, inspeção e/ou controle realizado por grupos cooperativos de entidades concorrentes está relacionado com o que foi estudado neste trabalho.

Mas como tal tarefa pode ter inúmeras definições e várias características diferentes dependendo do ponto de vista que está se colocando, houve a necessidade de definir o que seria a patrulha no escopo desse trabalho.

3
Metodologia

Como descrito na introdução, estávamos diante da construção de um Sistema Multiagente (SMA), para a tarefa de patrulha a ser colocada no jogo Canyon, quando apareceu uma dificuldade em definir qual seria a melhor maneira de fazer uma patrulha no ambiente, utilizando um grupo de naves. Como não foi encontrado, na literatura, nenhum estudo sistemático concentrado nesse assunto, várias questões ficaram abertas. A maioria delas estavam relacionadas às decisões que deveriam ser tomadas ainda na fase inicial do projeto. São algumas delas:

- Qual a melhor arquitetura de SMA para uma determinada tarefa de patrulha e suas restrições?
- Como avaliar uma arquitetura de SMA numa patrulha?
- Como determinados fatores influenciam o comportamento geral do grupo de agentes? Fatores tais como: tamanho e conectividade do grafo, tipo de comunicação, estratégia de raciocínio.

Para responder essas questões nós adotamos uma metodologia que consiste dos seguintes passos:

- Definição de medidas de desempenho;

- Proposição de diferentes arquiteturas de SMA;
- Definição dos estudos de caso (cenários para a patrulha);
- Implementação de um simulador para realização dos experimentos;
- Experimentação;
- Análise dos resultados.

Esses passos serão descritos no restante desse capítulo. Com exceção dos três últimos que serão detalhados nos capítulos 4 e 5.

3.1. Critérios de Avaliação

Um dos pontos cruciais encontrados no desenvolvimento desse trabalho foi a decisão de como seria feita a avaliação das diferentes arquiteturas de agentes patrulheiros.

Deste modo, uma das contribuições desse trabalho foi à definição de alguns critérios de avaliação que foram utilizados para comparar as diferentes arquiteturas de SMA no problema da patrulha. Os critérios escolhidos foram: *ociosidade média*, *pior ociosidade* e *tempo de exploração*.

Consideramos um *ciclo* o tempo necessário para que um agente vá de um nó para outro nó adjacente. Tal ciclo tem sempre o mesmo valor, pois consideramos, por questão de simplicidade, que todos os nós adjacentes têm distância constante entre si ou que a transição entre os nós vizinhos é sempre feita em um mesmo intervalo de tempo.

Chamamos de *ociosidade instantânea do nó* a quantidade de ciclos que um nó permanece sem ser visitado. Essa ociosidade é medida a cada ciclo durante todo o experimento. Utilizamos a *ociosidade instantânea do grafo*, definida como a média das ociosidades instantâneas de todos os nós do grafo

num dado ciclo, para descrever o estado global do grafo. Finalmente, a *ociosidade média do grafo*, ou simplesmente *ociosidade* é a média das ociosidades instantâneas do grafo encontradas nos *n* ciclos da simulação.

Sendo o_i a ociosidade do nó *i*, OI a ociosidade instantânea do grafo, *m* a quantidade de nós, *O* a ociosidade e *n* a quantidade de ciclos, temos:

$$OI = \frac{\sum_{i=1}^{m} o_i}{m} \qquad \textbf{3.1}$$

$$O = \frac{\sum_{i=1}^{n} OI_i}{n} \qquad \textbf{3.2}$$

Em todos os casos que trabalhamos com valores médios, sempre calculamos os desvios padrões respectivos, tendo assim uma informação mais completa que inclui o tamanho da variação desses valores.

No mesmo contexto, uma outra medida interessante é a *pior ociosidade*, i.e., o maior valor encontrado para a ociosidade instantânea do nó durante toda a simulação. Relacionamos a essa medida com o ciclo em que esse valor ocorre, determinando assim quando aconteceu esse pior caso.

O último critério de avaliação é o que chamamos de *tempo de exploração*. O mesmo consiste no número de ciclos necessários para que os agentes visitem, ao menos uma vez, todos os nós do grafo. Isso corresponde intuitivamente à noção de exploração de uma área para construção do mapa dessa região.

Todas essas medidas de desempenho, já descritas, tendem a exibir melhores resultados quando o número de agentes que estão patrulhando o grafo cresce. Isso pode esconder a contribuição do tipo de coordenação realizada entre os agentes, sendo a melhora conseguida apenas pelo aumento

da população dos agentes. Portanto, para avaliar a qualidade da coordenação com o aumento do número de agentes, foi decidido medir a contribuição individual dos agentes normalizando os três critérios (ociosidade, pior ociosidade e tempo de exploração) como mostrado na equação a seguir:

$$valor\_normalizado = valor\_absoluto \times \frac{número\_de\_agentes}{número\_de\_nós} \qquad \textbf{3.3}$$

Por exemplo, cinco agentes patrulham um grafo de cem nós e conseguem um tempo de exploração absoluto de cem ciclos, o valor normalizado para o mesmo é cinco (5 = 100 * 5/100). Para o dobro de agentes (dez agentes), a mesma arquitetura obteve um valor absoluto de sete ciclos para a mesma medida, o que dá um valor normalizado de sete (7 = 70 * 10/100). Esses dados mostram claramente como a coordenação não obteve um resultado tão bom com o aumento da população.

3.2. Arquiteturas de Sistemas Multiagentes Propostas

Para definir as arquiteturas de SMA que seriam interessantes para serem estudas e avaliadas na tarefa de patrulha, foram explorados principalmente quatro parâmetros básicos (mostrados na Tabela 3.1).

A idéia por trás da escolha dessas arquiteturas era de começar das mais simples para as mais complexas. Isso evitaria a necessidade de experimentações prematuras e ainda mostraria a variação do comportamento das arquiteturas à medida que sua complexidade aumentasse, ou novas características fossem sendo incorporadas ou alteradas. Por essa mesma razão, foram apenas consideradas arquiteturas homogêneas, onde todos os agentes são iguais. As exceções são as arquiteturas que possuem a figura de um coordenador, sendo esse o único diferente em relação aos demais.

Nome	Tipo básico	Comunicação	Estratégia de coordenação	Estratégia de raciocínio
Random Reactive	reativo	nenhuma	emergente	aleatória
Conscientious Reactive				heurística baseada na ociosidade
Reactive with Flags		flags		
Conscientious Cognitive	cognitivo	nenhuma		
Blackboard Cognitive		blackboard		
Blackboard Cognitive Monitored				
Random Coordinator		mensagens	central	aleatória
Idleness Coordinator				heurística baseada na ociosidade
Random Coordinator Monitored				aleatória
Idleness Coordinator Monitored				heurística baseada na ociosidade

Tabela 3.1 – Resumo das principais características das arquiteturas de SMA propostas e estudadas.

3.2.1. Tipo básico de Agente

O primeiro parâmetro considerado para a definição das arquiteturas de SMA foi a clássica diferença entre os agentes reativos e os agentes cognitivos [47]. Os agentes reativos simplesmente operam baseados em sua percepção atual, enquanto que os agentes cognitivos podem perseguir um objetivo. Os agentes cognitivos não possuem uma visão imediatista, como a dos reativos. Eles podem, por exemplo, considerar acontecimentos eventuais futuros ou um objetivo que está mais à frente.

Uma restrição, até natural, que colocamos foi a limitação do campo de visão dos agentes. Onde o mesmo seria de um nó de profundidade no grafo, em relação a sua posição atual, para os agentes reativos. Isto significa que os agentes reativos apenas podem perceber os nós adjacentes a sua posição atual, o que faz sentido já que eles não planejam uma rota para os nós mais

distantes. Já os agentes cognitivos têm conhecimento de todo o grafo e utilizam técnicas de *path-finding*[1] para alcançar o seu nó objetivo.

A técnica de *path-finding* utilizada neste trabalho foi o algoritmo Floyd-Warshall que consiste numa computação, realizada a priori, onde todos os menores caminhos entre todos os nós do grafo são calculados e ficam guardados numa estrutura para consulta posterior [34]. A opção dessa técnica foi tomada por questão de desempenho, pois não existe alteração no grafo durante o experimento, caso isso acontecesse uma outra técnica que fizesse esse cálculo *on-line* seria utilizada, tal como A*. Entretanto ambas resultam no mesmo caminho, que é o menor (*shortest path*).

3.2.2. Comunicação

Uma outra característica importante é a comunicação entre os agentes. De uma maneira geral, existem três formas dos agentes se comunicarem: através de *flags*, através de um *blackboard* ou através de *mensagens* [15][47].

No primeiro caso, os agentes deixam *flags* ou marcas no ambiente [11][15]. Essas marcas são reconhecidas por todos agentes e as informações são trocadas através das mesmas.

No segundo caso, a informação é armazenada numa estrutura de memória compartilhada (chamada de *blackboard*) que pode ser acessada e modificada por todos os agentes [15][47].

Para o último caso, existe a possibilidade de um canal direto entre os agentes e esses podem se comunicar através da troca de mensagens [15][47].

[1] São técnicas que servem para encontrar ou planejar um caminho para ir de um lugar para outro. Tais técnicas já estão num nível bem avançado e são largamente utilizadas em diversas situações práticas [47][49].

Para os testes realizados neste trabalho, os agentes podem trocar mensagens apenas com o coordenador, quando esse existe. Se a comunicação entre todos os agentes fosse permitida existiriam arquiteturas bem mais complexas. Por exemplo, arquiteturas com mecanismos de negociação e resolução de conflitos. Isso por si só já seria um estudo muito grande e complexo, assim deixamos essas arquiteturas para serem estudadas após o término destes trabalhos iniciais.

3.2.3. Estratégia de Coordenação

Por fim, definimos os tipos de coordenação que seriam aplicados as arquiteturas de SMA. Este é um dos pontos mais importantes quando estamos trabalhamos com um grupo de agentes [9][15]. É através da coordenação que se consegue tirar proveito da interação entre os integrantes de um grupo. Quando esta não existe, o grupo de agentes toma um comportamento caótico de uma forma bem rápida. A coordenação é essencialmente o comprometimento individual dos agentes em contribuir para um comportamento coerente e harmonioso do grupo. Para realizar tal coordenação utilizamos dois métodos bem diferentes. No primeiro colocamos um coordenador central que determinava qual seria o próximo objetivo de cada agente, que por sua vez executava a tarefa da maneira que melhor o conviesse, planejando como a tarefa será executada. No outro método de coordenação, a mesma era feita de forma descentralizada emergindo da interação entre os agentes.

3.2.4. Estratégia de raciocínio

A tomada da decisão também foi um ponto importante na geração das possíveis soluções para a patrulha. Em outras palavras, quais foram as estratégias utilizadas para decidir qual será o próximo nó a ser visitado. Dois aspectos foram considerados: o campo de visão, o qual pode ser *local* ou

global, e o critério de escolha, o qual pode ser *aleatório* ou com uma *heurística baseada na ociosidade* dos nós.

O campo de visão está fortemente relacionado com o que cada agente pode perceber do ambiente num determinado instante. Quando dizemos que tal característica é local os agentes somente percebem o que está próximo deles naquele momento, ou seja, os nós vizinhos de profundidade um. Para o caso global existe a percepção de todo o grafo. Esses limites colocados para a visão local e global poderiam ser diferentes, mas resolvemos trabalhar primeiramente com esses valores.

A primeira estratégia de raciocínio (critério de escolha) foi simplesmente escolher o próximo nó de forma aleatória, isto é, o próximo nó era decidido através de um sorteio. Como a ociosidade é o critério de avaliação mais importante para medir o desempenho das arquiteturas de SMA na patrulha, a outra heurística utilizada para o raciocínio foi escolher como próximo nó a ser visitado aquele que tivesse a maior ociosidade.

Existem duas variações para as estratégias de raciocínio que dependem do que cada agente sabe sobre os outros. A decisão pôde ser tomada levando em conta a *ociosidade individual*, quando foram consideradas apenas as visitas realizadas pelo próprio agente, não se tem conhecimento do que os outros agentes estão fazendo. A segunda variação se deu quando foi levada em conta a *ociosidade coletiva*, onde foram consideradas as visitas feitas por todos os agentes, assim as ações dos agentes eram conhecidas. Ainda existem mais duas vertentes que detalham ainda mais o processo de decisão: a *global* e a *local*. Quando a estratégia é *global* significa que todo o grafo é utilizado para a tomada da decisão, diferentemente da estratégia *local* onde apenas os nós adjacentes são considerados. Tudo isso é descrito na Tabela 3.1 e na Tabela 3.2.

Nome	Estratégia de raciocínio detalhada
Random Reactive	aleatória local
Conscientious Reactive	ociosidade individual local
Reactive with Flags	ociosidade coletiva local
Conscientious Cognitive	ociosidade individual global
Blackboard Cognitive	ociosidade coletiva global
Blackboard Cognitive Monitored	ociosidade coletiva global
Random Coordinator	aleatória global
Idleness Coordinator	ociosidade coletiva global
Random Coordinator Monitored	aleatória global
Idleness Coordinator Monitored	ociosidade coletiva global

Tabela 3.2 – Estratégia de raciocínio detalhada, considerando o campo de visão e a comunicação entre os agentes.

A inserção do monitoramento foi mais uma característica importante para a construção das arquiteturas, mais só é possível adicionar tal característica quando existe a possibilidade do agente saber se seu nó objetivo foi visitado por outro agente durante sua caminhada. Assim, quando existe o monitoramento nas arquiteturas de SMA o mesmo é realizado da seguinte maneira: cada agente verifica a cada passo se seu objetivo foi alcançado por outro. Caso isso tenha acontecido, um outro nó é escolhido, evitando um trabalho hipoteticamente inútil.

3.2.5. Colocando Tudo Junto

Combinando essas características principais e outras que também foram consideradas (e.g. visibilidade, técnicas de *path-finding*, etc.) obtivemos um número muito grande de possíveis arquiteturas de SMA para serem testadas. Muitas dessas arquiteturas não faziam sentido e outras não tinham uma relevância muito grande para este momento de estudo, como por exemplo: seria muito estranho colocar monitoramento nas arquiteturas compostas de agentes reativos, seria como monitorar os acontecimentos imediatos. Dentre as arquiteturas conseguidas com essa combinação de características, foram escolhidas dez (mostradas na Tabela 3.1).

Maiores detalhes sobre cada uma das arquiteturas são dados a seguir.

3.2.5.1. Random Reactive

Sendo esta arquitetura composta dos agentes de comportamento mais simples, eles escolhem qualquer um dos nós de sua vizinhança de forma aleatória para ser o próximo nó a ser visitado. Foi determinado que o tamanho da vizinhança para os agentes reativos seria um nó de profundidade no grafo, como já mencionado anteriormente.

3.2.5.2. Conscientious Reactive

Esses agentes possuem memória e têm consciência das suas ações antecedentes. Cada agente escolhe como próximo passo o nó de sua vizinhança que faz mais tempo que foi visitado por ele. Sempre que existe um empate é feito um sorteio entre esses nós.

3.2.5.3. Reactive with Flags

Esta arquitetura compartilha a informação das visitas de todos agentes através de marcas deixadas no ambiente (*flags*). Assim, os agentes escolhem o nó de sua vizinhança que faz mais tempo que foi visitado por qualquer agente, ou seja, o nó de maior ociosidade dentro da vizinhança (nós adjacentes). Em caso de empate é feito um sorteio entre estes nós.

3.2.5.4. Conscientious Cognitive

Estes agentes trabalham de forma análoga aos da seção 3.2.5.2 (*Conscientious Reactive*), pois possuem memória e têm consciência das suas ações antecedentes. No entanto, os integrantes dessa arquitetura têm um objetivo e a capacidade de planejar um caminho para ele, encontrando o menor caminho (*shortest path*) como já descrito. Isso implica na possibilidade de escolher qualquer nó do grafo como o próximo objetivo, assim foi adotada a estratégia de escolher o nó de maior ociosidade individual.

3.2.5.5. Blackboard Cognitive

No caso da arquitetura *Blackboard Cognitive*, de forma semelhante a arquitetura *Reactive with Flags* (seção 3.2.5.3) existe uma comunicação entre os agentes que é feita de forma indireta. Porém, ao invés de marcas deixadas no ambiente, existe uma estrutura comum (*blackboard*) onde todos os agentes indicam sua passagem por cada nó. Essa memória compartilhada pode ser alterada e consultada por todos os agentes, sendo assim, o nó de maior ociosidade, daquele momento, é escolhido como próximo objetivo, se houver empate a opção de sorteio é utilizada.

3.2.5.6. Blackboard Cognitive Monitored

Quando os agentes têm conhecimento da atuação dos outros, existe a possibilidade de realizar um monitoramento desses atos. Como o próprio nome sugere, esta arquitetura de SMA é idêntica a anterior. Diferenciando-se apenas pela adição da característica de monitoramento. O monitoramento é feito da seguinte forma: a cada passo que um agente dá em direção ao seu objetivo, ele verifica se seu objetivo foi alcançado por outro agente. Caso isso tenha acontecido ele já escolhe outro. Portanto, não gasta tempo para completar uma tarefa que acabou de ser realizada.

3.2.5.7. Random Coordinator

A partir dessa seção as arquiteturas deixam de ser homogêneas, pois é adotada a estratégia de coordenação central explícita, e para tal é incluído um outro tipo de agente chamado de coordenador. Esse coordenador tem conhecimento de todas as visitas que acontecem no ambiente e determina o objetivo de cada agente. O protocolo de decisão acontece da seguinte forma: cada agente pergunta ao coordenador qual será seu objetivo, com seu objetivo determinado a responsabilidade de como cumprir a tarefa fica por conta do

agente patrulheiro. Os agentes que executam a patrulha são cognitivos, ou seja, dado o objetivo eles planejam o caminho que deve ser percorrido.

Nesta arquitetura de SMA, o coordenador escolhe os objetivos de forma aleatória. Assim, quando um agente pergunta para onde ele deve ir, o coordenador faz um sorteio entre os nós que não são objetivos para os outros agentes naquele momento. O nó sorteado será o próximo objetivo para o agente que fez o pedido. Com isto, não existe a possibilidade de dois agentes terem o mesmo objetivo ao mesmo tempo.

3.2.5.8. Idleness Coordinator

Para este caso, a decisão do coordenador é baseada na ociosidade, onde o nó atribuído ao agente da vez é aquele com mais tempo sem ser visitado. O critério de escolher somente dentre aqueles nós que não são objetivos para os outros agentes continua para todas as arquiteturas com coordenador central.

3.2.5.9. Random Coordinator Monitored

Nas duas arquiteturas anteriores, assim que os agentes recebem seus objetivos seguem seus caminhos até alcançarem os mesmos. Tal como o nome já sugere, os agentes desta arquitetura têm capacidade de realizar o monitoramento. Portanto, a cada passo do plano os agentes verificam se seu objetivo foi alcançado por outro agente, caso isso aconteça ele solicita um outro ao coordenador. O coordenador continua agindo da mesma forma que o coordenador da arquitetura *Random Coordinator* (seção 3.2.5.7).

3.2.5.10. Idleness Coordinator Monitored

Os agentes desta arquitetura trabalham de forma análoga aos agentes da arquitetura *Idleness Coordinator* (seção 3.2.5.8) com exceção do monitoramento que é adicionado para este caso.

Com as arquiteturas definidas a próxima preocupação metodológica foi a de definir as situações em que esses agentes (arquiteturas) iriam atuar. Assim, passamos a definir os cenários apresentados a seguir.

3.3. Cenários de Experimentação

Após uma reflexão sobre que parâmetros do ambiente poderiam influenciar o desempenho de cada arquitetura de SMA durante a execução da patrulha, concluímos que a quantidade nós por agente e a conectividade desses nós seriam fatores importantes. Nesta perspectiva foram criados dois mapas com características bem diferentes, apesar de algumas semelhanças. Estes são mostrados na Figura 3.1.

O mapa A tem poucos obstáculos e possui um grafo altamente conectado, facilitando o deslocamento de uma região para outra do mapa, já que há vários caminhos possíveis. No mapa B existem alguns gargalos que criam regiões com poucas vias de acesso, que diminuindo a mobilidade de uma região para outra do mapa. Vale a pena notar que o grafo apresentado para o mapa B (69 arestas e 50 nós) tem bem menos arestas e é bem menos conectado do que o mapa A (106 arestas e 50 nós), mas utiliza os mesmos cinqüenta nós para a construção do grafo.

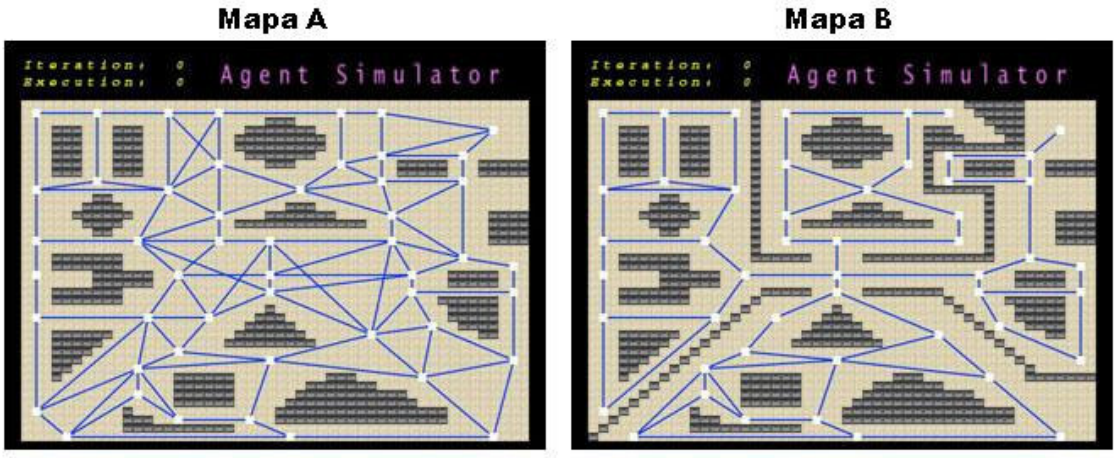


Figura 3.1 – Mapas A e B em nosso simulador. Os blocos pretos representam os obstáculos. Os grafos (*skeletons*) com os caminhos possíveis são mostrados. O grafo do mapa A possui 106 arestas e 50 nós, e o grafo do mapa B, com mais gargalos, tem 69 arestas e os mesmos 50 nós. Estas figuras também são instantâneos de tela retirados do simulador.

A próxima preocupação foi variar a relação entre a quantidade de nós e a quantidade de agentes. Assim, dados os mapas com cinqüenta nós, já mostrados, escolhemos as populações de 1, 2, 5, 10, 15 ,25 agentes para cada arquitetura de SMA proposta. Essas escolhas resultam na proporção de 50, 25, 10, 10/3 e 2 nós para cada agente. As relações entre nós e agentes, mostradas anteriormente, dão uma boa variação para a análise dos resultados dos experimentos.

4
Simulador

Como já discutido no capítulo 3, a construção de um simulador faz parte da metodologia adotada neste trabalho. Tal simulador foi arquitetado para prover um ambiente com todas as características relevantes para o problema em questão, onde pudéssemos realizar os experimentos necessários para a conclusão de nossos estudos no problema da patrulha.

Além de prover tal ambiente, o simulador dá um maior controle da situação, facilitando a coleta das informações necessárias para o estudo do desempenho das arquiteturas de SMA propostas na seção 3.2. Outra vantagem da utilização de um simulador é a redução dos custos, pois não precisamos comprar robôs para realização de um experimento num ambiente real. O simulador ainda tem a opção de animar graficamente a desenvoltura dos agentes durante o experimento.

Neste capítulo será apresentada a arquitetura do simulador, as classes desenvolvidas, a modelagem do sistema, o funcionamento, além de detalhes de implementação.

4.1. Desenvolvimento do Simulador

A primeira versão desse simulador foi concebida juntamente com Alessandro de Luna Almeida, Eric Bruno Perazzo Mariz e Mozart de Siqueira

Campos Araújo Filho, alunos do CIn/UFPE, que foram responsáveis por boa parte do projeto e da implementação do mesmo. A forte relação do nosso trabalho com o desenvolvimento de jogos influenciou muito a concepção do simulador, além de utilizar C++ e OpenGL[1], ferramentas comumente utilizadas para esse fim, o seu funcionamento segue um processo muito semelhante ao dos jogos.

Na seção seguinte será explicado o funcionamento básico de um jogo [27], isso servirá como base para mostrar o funcionamento do simulador.

4.1.1. Componentes de um Jogo

Um jogo de computador (*video game*) é basicamente um laço (*loop*) contínuo que excuta uma determinada lógica e desenha uma imagem na tela, normalmente a uma taxa de 30 quadros por segundo (*fps - frame per seconds*) ou mais. Isso é similar a execução de um filme, exceto que o jogo vai sendo criado à medida que é mostrado.

Este laço, descrito na Figura 4.1, é responsável pelo gerenciamento e execução dos principais componentes da aplicação [27].

[1] OpenGL (Open Graphics Library) é uma biblioteca de rotinas gráficas para modelagem 2D e 3D (http://www.opengl.org).

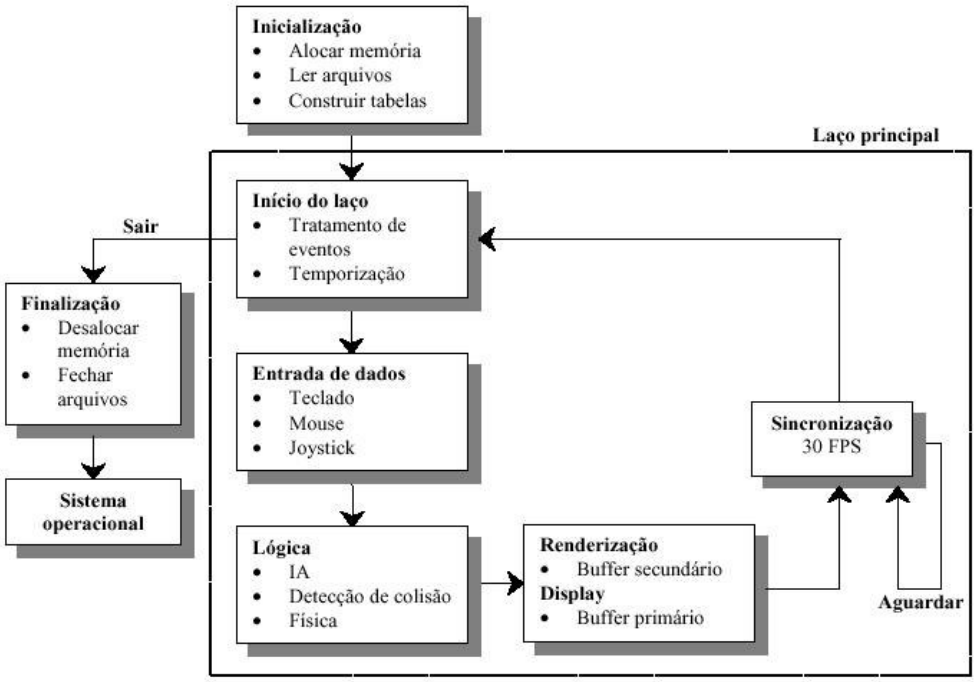


Figura 4.1 – Laço principal de um jogo.

Cada parte desse laço será descrita a seguir:

- **Inicialização** – responsável pelas operações padrões de inicialização que estão presentes em qualquer programa. Como alocação de memória, aquisição de recursos, leitura de dados do disco rígido e assim por diante;
- **Início do laço** – realiza a execução do código de entrada no laço principal;
- **Entrada de dados do usuário** – recebe as entradas do jogador que são processadas e/ou armazenadas para uso posterior na seção de IA e lógica;
- **Lógica do jogo** – esta seção contém a maior parte do código do jogo. A inteligência artificial, modelagem física e a lógica geral do

jogo são executados, e os resultados serão utilizados para desenhar a próxima imagem na tela;

- **Renderização** – os resultados da entrada do jogador, a execução da IA e da lógica do jogo são utilizados para gerar o próximo quadro para a animação do jogo. Esta imagem geralmente é desenhada em uma área chamada de *buffer* e logo após é desenhada na tela, através de uma operação de cópia, que é bem rápida;
- **Sincronização** – devido a variação de desempenho dos computadores que também é influenciada pela complexidade de cada jogo, uma sincronização é colocada para conseguir uma taxa de apresentação satisfatória. Essa sincronização é feita através de temporizadores;
- **Saída** – esta seção é o fim da aplicação, quando o usuário sai do jogo e retorna para o sistema operacional. No entanto, antes da saída do usuário, todos os recursos devem ser liberados. Como deve acontecer com qualquer outro tipo de software.

Com o funcionamento do laço principal de um jogo explicado fica fácil entender o processo de execução do simulador, que será mostrado na próxima seção, pois este segue o mesmo princípio.

4.1.2. Funcionamento do Simulador

Tal como mencionado anteriormente o simulador segue todos os passos da Figura 4.1, onde primeiramente acontece a inicialização. É nesta fase que ocorre a construção do ambiente virtual para experimentação. A criação deste ambiente é baseada num arquivo de configuração. Arquivo este que descreve todas as características necessárias para a construção do

ambiente, tais como: o tamanho, os obstáculos, o grafo, o número de agentes, as posições iniciais, o tipo de arquitetura de SMA, o número de ciclos, etc.

Assim, os primeiros passos do simulador são: fazer as primeiras alocações necessárias, ler o arquivo de configuração e construir o ambiente (instanciar os objetos nesse ambiente). Logo depois, acontece a primeira interação com o usuário, onde é perguntado se a simulação deve acontecer em tela cheia. Após esta resposta, começa o laço principal.

No início do laço são tratados os eventos gerados durante a execução do mesmo. Nesse ponto também acontece o fim do laço, que pode ocorrer devido ao término da experimentação ou através de um pedido do usuário.

No próximo passo são coletadas as entradas do usuário. Em nosso caso, diferentemente de um jogo, as entradas do usuário não influenciam em nada o comportamento dos agentes, apenas possibilitam a saída do simulador.

Na seção de lógica é o coração do simulador. Nesse ponto os agentes são atualizados, isto é, cada agente executa suas ações, um novo estado do mundo é gerado e as estatísticas são coletadas.

Logo após a execução da parte lógica uma imagem é criada e armazenada no *buffer*, que é copiado para a tela. Em seguida acontece a sincronização.

A sincronização só acontece se for determinada a visualização da simulação no arquivo de configuração. Essa possibilidade de escolha permite uma execução mais rápida quando uma visualização não se faz necessária, o quê geralmente acontece quando existe uma quantidade muito grande de experimentos.

O laço é continuamente executado até que ocorra um evento de saída ou se chegue ao final da simulação. Quando é executada a saída do simulador, os recursos são liberados e ocorre a “limpeza” necessária.

4.2. A Estrutura do Simulador

Nosso simulador foi concebido de maneira a deixá-lo reutilizável e extensível. Para tanto, criamos um total desacoplamento entre: as estruturas do grafo, o simulador, o mapa, o gerador/editor de mapa, as estatísticas e os agentes.

Um exemplo de uma execução do simulador é mostrado na Figura 4.2.

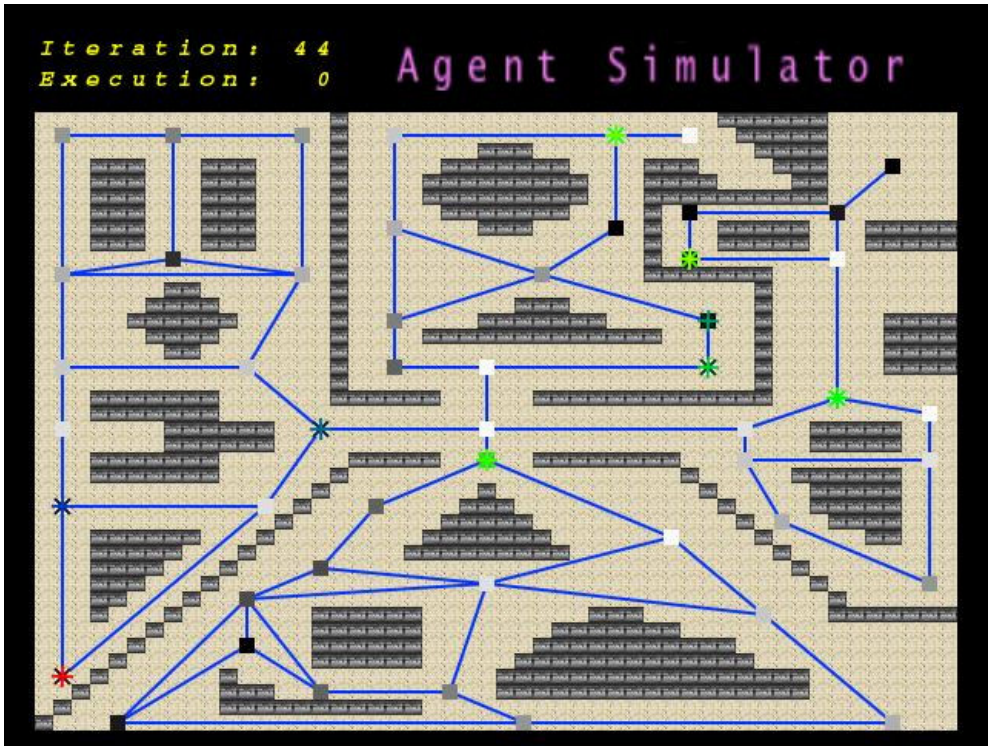


Figura 4.2 – Tela da execução de uma simulação no mapa B com dez agentes.

O sistema foi modelado com UML (*Unified Modeling Language*) [43], que representa muito bem o paradigma orientado a objetos, utilizado para a construção do nosso simulador.

Partindo para maiores detalhes do projeto, mostraremos as classes principais, classes do domínio do problema, bem como sua funcionalidade e/ou atuação no sistema. As classes de infra-estrutura técnica, criadas apenas para prover as condições de funcionamento do simulador, só serão detalhadas quando essas forem necessárias para um melhor entendimento do funcionamento do simulador ou para o desenvolvimento de novas arquiteturas.

4.2.1. Descrição das Classes

O diagrama de classe, com as classe do domínio do problema, é mostrado na Figura 4.3.

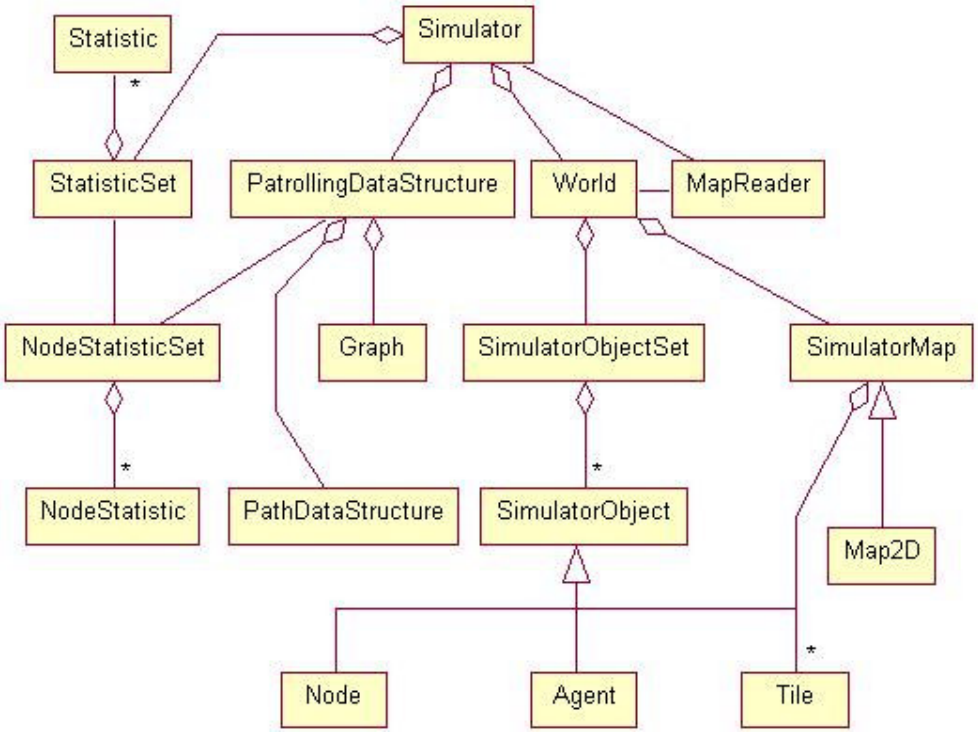


Figura 4.3 – Diagrama com as classe do domínio do problema.

`Simulator` é a principal classe do sistema. Ela é responsável pelo laço descrito nas seção 4.1.1. A classe `Simulator` é composta pelas classes `World`, `PatrollingDataStructure`, `StatisticSet` e utiliza a classe `MapReader` para ler o arquivo de configuração, retirar as informações importantes e construir o ambiente (classe `World`).

A classe `PatrollingDataStructure` cria uma estrutura que foi utilizada para trabalhar com o grafo. Com esta estrutura, podemos fazer várias consultas relacionadas, tais como: menor caminho, nó menos visitado, trabalhar com os nós, etc. Essa classe é utilizada até na hora de gerar as estatísticas, pois através da mesma conseguimos gerar um conjunto de dados estatísticos (`NodeStatisticSet`) utilizados pela classe `StatisticSet` para criar os resultados da simulação. O algoritmo utilizado para busca do menor caminho (*shortest path*) está implementado na classe `PathDataStructure`. Como mencionado anteriormente, utilizamos o algoritmo de Floyd-Washal para tal procedimento.

O ambiente é representado pela classe `World` que é composta pelo mapa (`SimulatorMap`) e por um conjunto de objetos (`SimulatorObjectSet`) embutidos nesse ambiente. Os tipos de mapas vão sendo criados a partir de extensões da classe `SimulatorMap`. Com o intuito de realizar os experimentos, criamos uma classe chamada `Map2D` para representar um mapa em duas dimensões. Essa extensibilidade mostra como é fácil criar novos tipos de mapa. Por exemplo, um mapa isométrico poderia ser criado estendendo a classe `SimulatorMap`, e tal classe seria utilizada, pelo simulador, com apenas pequenas modificações na classe `MapReader`.

Da mesma forma, os objetos do mundo vão sendo criados. Todos objetos presentes no ambiente de simulação e visíveis para o usuário são subclasses de `SimulatorObject`. Onde são determinadas algumas características padrões, tais como: posição, método de renderização, etc.

A classe `StatisticSet` trabalha com a parte estatística do experimento. Para cara simulação são gerados três arquivos, que serão descritos a seguir, contendo informações sobre cada experimento realizado.

4.2.2. Criando Novas Arquiteturas

A criação de novas arquiteturas de agentes também tira proveito da facilidade de extensão das concebidas no projeto da nossa aplicação. Para criar novos agentes basta estender a classe `Agent`; implementar o método `reason`, que é chamado a cada interação com o ambiente; e modificar a classe responsável pela leitura do mapa para compor cada arquitetura com os agentes respectivos. Os agentes criados para nossas arquiteturas estão descritos na Figura 4.4.

Além das classes que descrevem os agentes, existem mais duas classes, na Figura 4.4, que não são derivadas da classe `Agent`, mas são importantes para a construção de alguns agentes. Uma delas é a classe `Graph` utilizada pelos agentes que tem um conhecimento próprio ou individual do estado do grafo, ou seja, guardam o estado do ambiente de acordo com suas ações. A outra classe é a `Path` que é utilizada pelos agentes cognitivos para construir um caminho de um nó para outro do grafo.

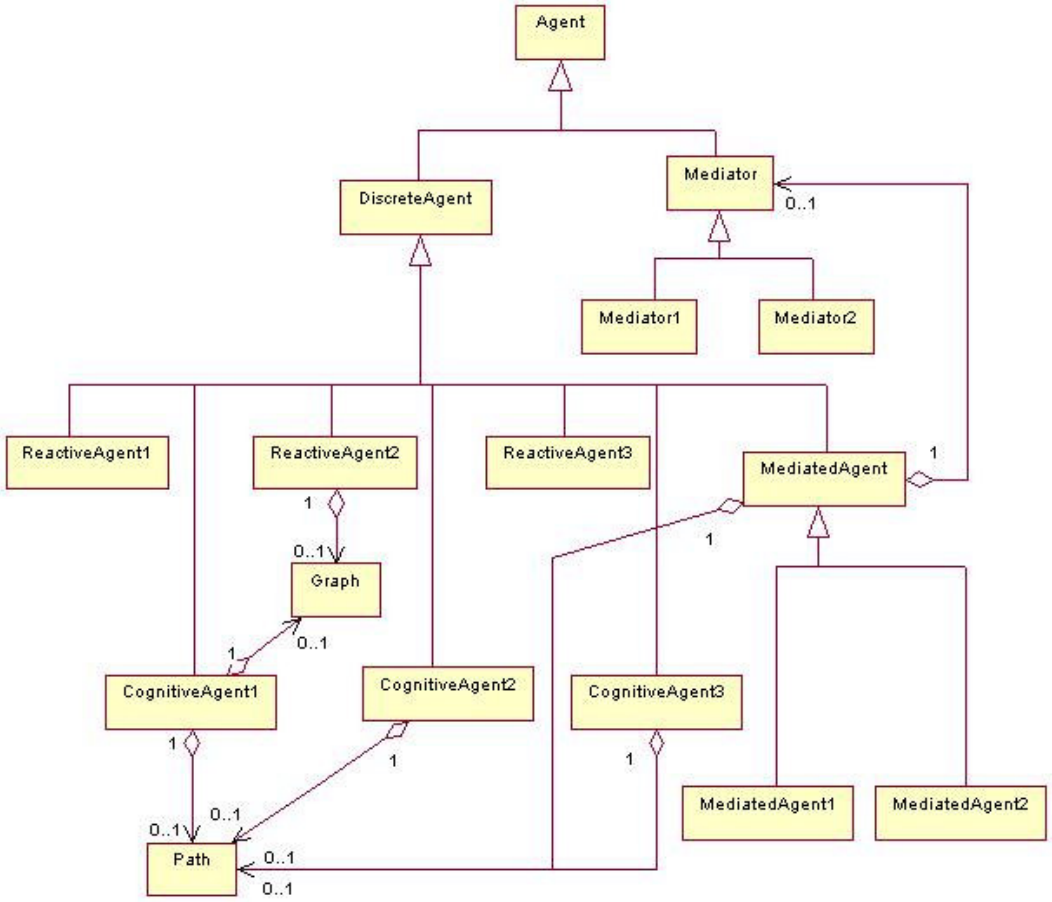


Figura 4.4 – Agentes criados para a construção das arquiteturas testadas

Para definir melhor nossos agentes, criamos uma classe chamada `DiscreteAgent`, todos os agentes utilizados nas simulações foram criados a partir dessa classe. Tal classe reflete a decisão de tratar a transição entre os nós vizinhos de forma discreta, onde a mesma é feita sempre durante o intervalo de um ciclo.

Existem três classes de agentes reativos, são elas: `ReactiveAgent1`, `ReactiveAgent2` e `ReactiveAgent3`. A classe `ReactiveAgent1` é utilizada na arquitetura *Random Reactive* e tem um comportamento aleatório. A classe `ReactiveAgent2` é utilizada pela arquitetura *Conscientious Reactive*, essa classe possui uma representação individual do grafo que é conseguida

através da classe `Graph`. Já a classe `ReactiveAgent3` é utilizada na arquitetura *Reactive with Flags* que compartilha informações através de marcas (flags) deixadas no ambiente, tais marcas podem ser reconhecidas pelos outros agentes e assim a informação é compartilhada.

Mais três classes implementam os agentes cognitivos e como tais estão relacionadas com a classe `Path`, estas classes foram chamadas: `CognitiveAgent1`, `CognitiveAgent2` e `CognitiveAgent3`. Os agentes da classe `CognitiveAgent1` compõem a arquitetura *Conscientious Cognitive* que também estão relacionados com a classe `Graph` para criar uma visão individual do grafo. As classes `CognitiveAgent2` e `CognitiveAgent3` fazem parte das arquiteturas *Blackboard Cognitive* e *Blackboard Cognitive Monitored* respectivamente, esses agentes compartilham um grafo comum (*blackboard*) onde são escritas/lidas todas as informações sobre as ações de cada um. A única diferença entre as duas arquiteturas é o monitoramento presente no segundo tipo. Nesse caso, os agentes verificam a cada passo se seu objetivo foi alcançado por outro, caso isso tenha acontecido, o objetivo é abandonado e outro é escolhido.

As classes restantes descrevem as arquiteturas com coordenador explícito representado pela classe `Mediator`, os agentes coordenados são implementados pela classe `MediatedAgent`. A classe `Mediator` possui duas extensões que são utilizadas para coordenar as arquiteturas de dois modos: aleatório (`Mediator1`) e baseado na maior ociosidade (`Mediator2`). Os agentes coordenados também possuem duas extensões, `MediatedAgent1` e `MediatedAgent2`. Esses dois tipos de agente possuem o comportamento de ir para o seu objetivo seguindo o menor caminho, mas são diferentes quanto à questão do monitoramento, que esta presente na segunda classe. A combinação dos dois tipos de coordenadores com os dois tipos de agentes coordenados formam as quatro arquiteturas restantes: *Random Coordinator*,

Idleness Coordinator, *Random Coordinator Monitored* e *Idleness Coordinator Monitored*.

4.2.3. Descrevendo o Ambiente

O ambiente é descrito para nosso simulador através de um arquivo de configuração chamado `world.ini` que é lido durante sua inicialização. Nesse arquivo são especificados:

- O tamanho do mapa;
- As áreas que são navegáveis e onde estão os obstáculos;
- O grafo;
- A arquitetura de SMA e a quantidade de agentes;
- A posição inicial de cada agente;
- O número de simulações e a quantidade de ciclos em cada simulação;
- A visualização do experimento.

O formato deste arquivo é mostrado e explicado com um exemplo na Figura 4.5.

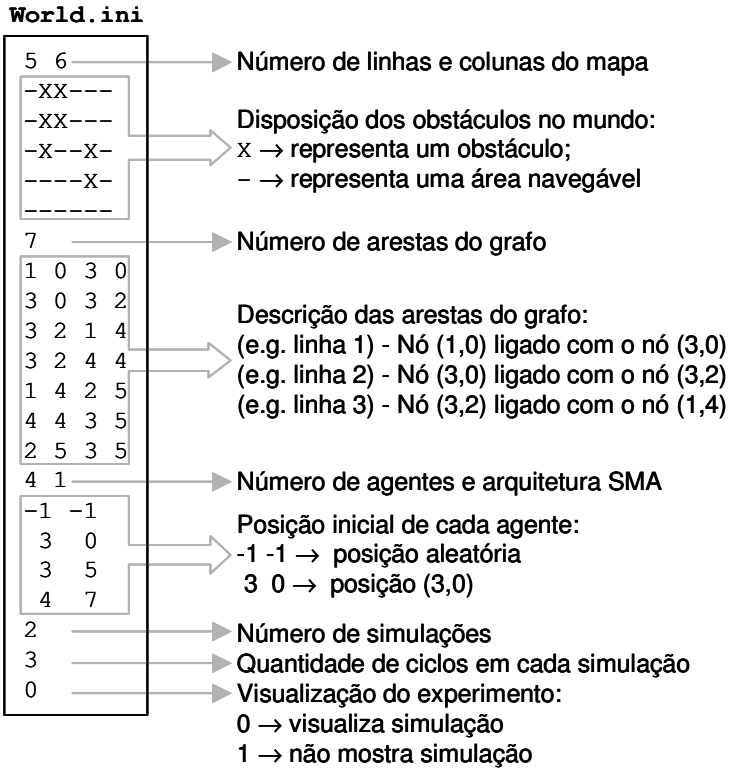


Figura 4.5 – Descrição do arquivo de configuração. A esquerda está o arquivo (`world.ini`) e a direita uma explicação para cada parte do mesmo.

Na Figura 4.6, a seguir, é mostrada a visualização gerada pelo simulador utilizando o arquivo de configuração da Figura 4.5.

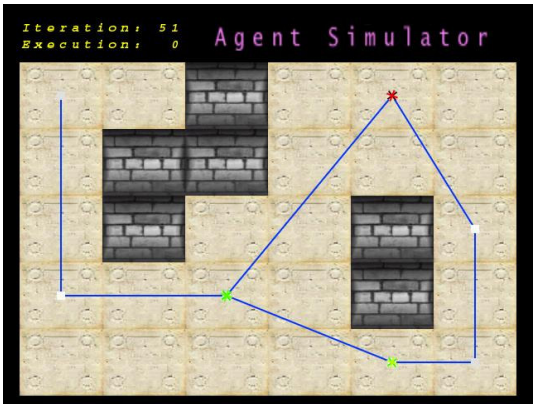


Figura 4.6 – Visualização gerada pelo simulador utilizando o arquivo da Figura 4.5.

4.2.4. Resultados das Simulações

Além de mostrar o comportamento de cada agente o simulador gera três arquivos que descrevem como foi a simulação, numa espécie de *log* do experimento.

O primeiro arquivo gerado contém uma descrição do experimento, um exemplo desse arquivo é mostrado na Tabela 4.1.

```
*************************************************************************
*                          Simulation Description
*************************************************************************
*
* Simulation ID:         10243462330
* Number of nodes:       50
* Number of executions: 1
* Actual executions:     0
* Number of steps:       50
* Number of agents:      10
* Type of agent:         11
*
*************************************************************************
```

Tabela 4.1 – Conteúdo do arquivo com a descrição do experimento.

Os outros dois arquivos mostram o que aconteceu durante a simulação com duas visões diferentes: viés temporal e viés nodal. No viés temporal é mostrado o que se passou a cada ciclo. No segundo caso o enfoque é o nó, assim este arquivo apresenta os resultados sob o ponto de vista de cada nó.

Todos os campos do arquivo temporal são mostrados no cabeçalho da Tabela 4.2, nas outras linhas da tabela foi colocado, como exemplo, parte de um arquivo de uma simulação realizada.

Ciclo	Ociosidade		Pior	Ociosidades dos nós				
	Média	Variância						
0	0.0	0.0	0	0	0	0	0	...
1	0.8	0.1632653	1	1	1	1	0	...
2	1.54	0.58	2	2	2	0	1	...
3	2.08	1.54449	3	3	0	1	2	...
4	2.64	2.888163	4	4	0	1	0	...
5	2.98	4.509796	5	0	1	2	1	...
M	M	M	M	M				

Tabela 4.2 – Descrição e parte do conteúdo de um arquivo de viés temporal.

O arquivo que apresenta dados com uma visão nodal é mostrado da mesma maneira na Tabela 4.3.

Nó	Visitas			Pior	Ociosidades das visitas					
	Quantidade	Média	Variância							
0	3	3.0	3.	4	4	1	4			
1	4	2.25	2.25	4	3	1	4	1		
2	2	4.0	8.	6	2	6				
3	7	0.8571	0.4761	2	0	0 1	1	1	1	2
4	4	2.	3.3333	4	0	3	1	4		
5	5	0.8	3.2	4	0	0	0	4	0	
M	M	M	M	M	M					

Tabela 4.3 – Descrição e parte do conteúdo de um arquivo de viés nodal.

Todas essas informações contidas nesses arquivos servem de subsídio para análises, estudos e discussões bem completas e variadas sobre as arquiteturas testadas.

4.3. Conclusões

Esse capítulo descreveu o funcionamento do simulador e mostrou alguns detalhes de implementação e organização do mesmo. Conforme descrito, o sistema é modular e extensível, sendo assim, flexível para testar as mais variadas arquiteturas de SMA.

O próximo capítulo apresenta os resultados e discussões, bem interessantes, sobre os experimentos realizados.

5
Experimentos e Resultados

Este capítulo delineia os resultados das experimentações realizadas com o simulador que foi apresentado, em detalhes, no capítulo anterior. Primeiramente, será descrito como foi realizado cada experimento (ou simulação), logo em seguida serão mostrados os resultados e por último uma discussão sobre os mesmos.

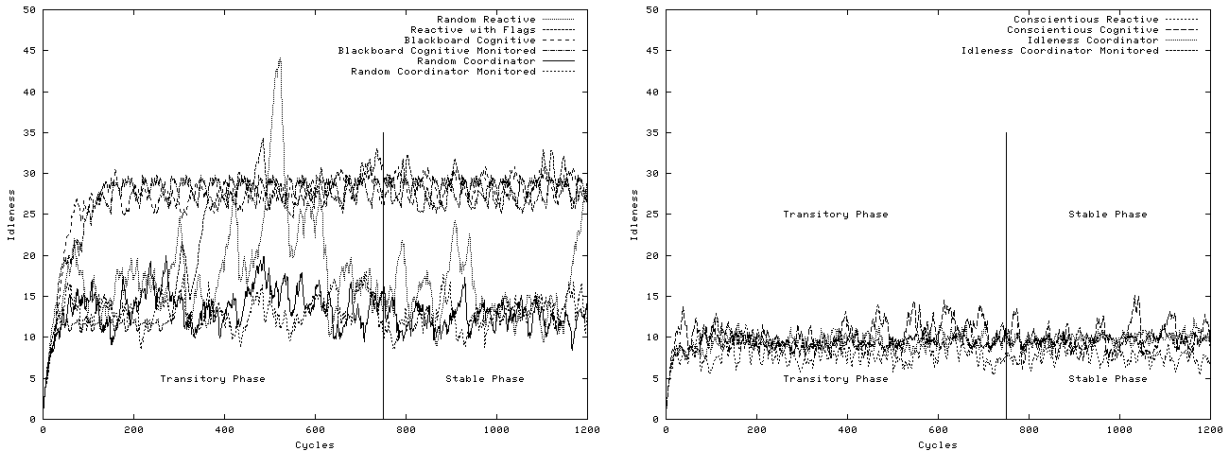
5.1. Configuração das simulações

Após discussões com estatísticos decidimos realizar trinta simulações com cada cenário descrito na seção 3.3. Em cada uma das trinta simulações, os agentes começam em pontos (nós) diferentes (i.e. posição inicial dos agentes). Estas posições iniciais foram escolhidas de maneira aleatória, mas uma vez escolhidas elas foram novamente utilizadas na inicialização das diferentes arquiteturas. Com isso, todas arquiteturas com a mesma quantidade de agentes tiveram suas simulações começando dos mesmos nós, escolhidos de forma aleatória. Isso implica numa melhor comparação entre as arquiteturas de SMA, pois todos os experimentos, com cada arquitetura, começaram da mesma forma.

Nós rodamos 360 simulações para cada uma das dez arquiteturas da Tabela 3.1 (coluna 1). Este número corresponde à combinação dos dois tipos de mapas, das seis populações diferentes e das trinta simulações (2 x 6 x 30 = 360). Cada simulação é composta de 3000 ciclos, isto é, cada agente muda de um nó para o outro 3000 vezes.

No início da simulação, consideramos que todas as ociosidades instantâneas dos nós começavam com valor igual a zero, como se todos os nós fossem visitados naquele momento. Conseqüentemente, esta decisão faz o resultado final da ociosidade tender para um valor menor que o real, pois estamos determinando que o seu valor inicial é zero. Mas é obvio que esta alteração no resultado fica irrelevante se a quantidade se ciclos for muito grande, entretanto esta não é a solução ideal. Para não dar essa falsa impressão seria necessário um grande número de ciclos, o que custa muito caro e inviabiliza os experimentos.

Os resultados dos experimentos podem ser divididos em duas fases: *fase transitória* e *fase estável*. A primeira, chamada de *fase transitória*, o valor da ociosidade sai do zero para um intervalo de valores não influenciados pelo estado inicial imposto. Na segunda, chamada de *fase estável*, não ocorre mais à influência do estado inicial, os valores são decorrentes do comportamento das arquiteturas. Isso é facilmente verificado na Figura 5.1, onde a transição de zero para um valor mais estável é bem visível no início de cada gráfico.



A B

Figura 5.1 – Os gráficos mostram a evolução, ociosidade(eixo y) sobre o tempo(eixo x), de cinco agentes durante uma simulação. Os resultados das arquiteturas estão separados (figuras A e B) para uma melhor visualização.

Portanto, para evitar a influência do estado inicial do experimento e ter a avaliação do verdadeiro resultado bastava suprimir a fase transitória da computação final da simulação, mas surgiu a dúvida de quando terminaria a fase transitória.

A primeira idéia foi a de terminar a fase transitória no momento em que todos os nós do grafo fossem visitados ao menos uma vez, ou seja, quando o grafo fosse explorado. Apesar de intuitiva, tal abordagem não foi utilizada, pois existiam situações onde a exploração de todo o grafo não acontecia numa quantidade de ciclos que fosse viável para realização de todos os experimentos, como era o caso dos agentes que tinham um comportamento aleatório. Além disso, essa abordagem atrapalharia a comparação entres os experimentos, pois não teríamos as fases estáveis com o mesmo número de ciclos, visto que a exploração acontece em tempos diferentes para cada caso. Por exemplo, teríamos que comparar uma arquitetura que teve uma fase estável de 2250 ciclos com outra com uma fase estável de 1732 ciclos.

Sendo assim, a abordagem utilizada para determinar as duas fases foi a de escolher um valor que garantisse, para a maioria dos casos, que a transição já teria ocorrido. Tal decisão foi bem aceita, pois estaríamos trabalhando dentro da fase estável da simulação. Além disso, utilização de um valor fixo para a fase transitória deixa todos os experimentos com a mesma quantidade de ciclos computados e soluciona o caso de quando não acontece a exploração do terreno. Tais características são de vital importância para toda a análise feita sobre resultados.

De acordo com alguns experimentos prévios, determinamos que a fase transitória terminaria no ciclo 750 e a ociosidade do grafo seria medida nos 2250 ciclos restantes. Tal valor foi determinado dando uma faixa de segurança em relação ao tempo de exploração da maioria dos agentes. Tal fato se confirmou, pois todas as explorações aconteceram bem antes do ciclo 750, exceto para alguns casos com os agentes aleatórios onde isso só acontece numa quantidade de ciclos bem elevada, como já explicado anteriormente.

5.2. Resultados

Nos gráficos seguintes cada tipo de linha representa uma arquitetura de SMA diferente, conforme é descrito na legenda. Um padrão quanto ao tipo de linha para cada arquitetura foi adotado para todos os gráficos, o que significa que as arquiteturas estão sempre representadas através das mesmas linhas. As barras verticais mostram o desvio padrão calculado nas trinta variantes das posições iniciais dos agentes. Os gráficos serão apresentados em pares, mostrando respectivamente os valores absolutos e os valores normalizados, segundo a equação 3.3, para cada critério de avaliação utilizado.

A Figura 5.2 e a Figura 5.3 mostram a ociosidade do grafo (eixo y) para cada grupo de agente (eixo x) para as simulações realizadas no mapa A.

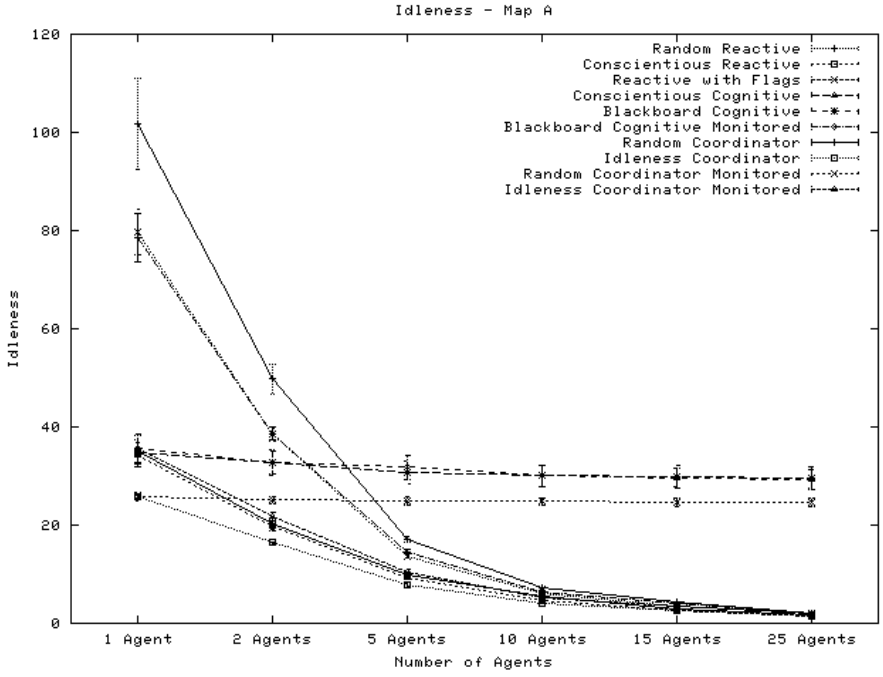


Figura 5.2 – Figura apresenta a **ociosidade** e os seus desvios para os experimentos realizados como **mapa A**.

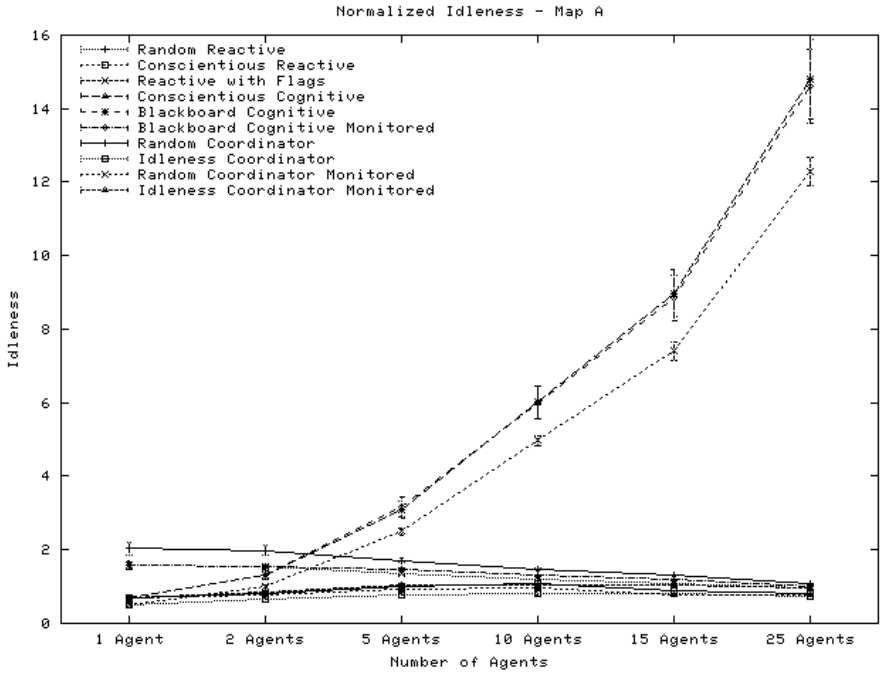


Figura 5.3 – Resultados da **ociosidade normalizada** conseguida nos experimento com os agentes no **mapa A**.

Com apenas estes gráficos, já se nota claramente como o aumento da população não influencia nos resultados das arquiteturas onde existe um compartilhamento da informação e não existe um mecanismo de coordenação. Isto é ainda mais explícito no gráfico normalizado. As arquiteturas que se enquadram neste fato são: *reactive with flags, cognitive blackboard e cognitive blackboard monitored.*

Para as demais arquiteturas o aumento da população sempre melhora o desempenho final. Mas descobrimos um fato muito interessante com os resultados normalizados: as arquiteturas que escolhem o próximo objetivo baseado no critério da maior ociosidade (conscientious reactive, idleness coordinator monitored, idleness coordinator e conscientious cognitive) apresentam seus melhores resultados, do ponto de vista da coordenação, a partir de uma população com mais de dez agentes. Ou seja, a divisão da tarefa ou participação individual de cada agente é aproveitada melhor a partir da razão de cinco nós por agente (10 agentes). Com populações menores a melhora se deve principalmente ao fato de ter mais agentes na execução da tarefa.

Para verificar isto graficamente colocamos ampliações dos mesmos gráficos apresentados neste capítulo no Apêndice A, dando uma melhor visualização para os detalhes que não podemos ver na escala apresentada neste capítulo.

Na Figura 5.4 e na Figura 5.5 são mostrados os resultados, do mesmo critério (ociosidade), conseguidos pelos agentes no mapa B.

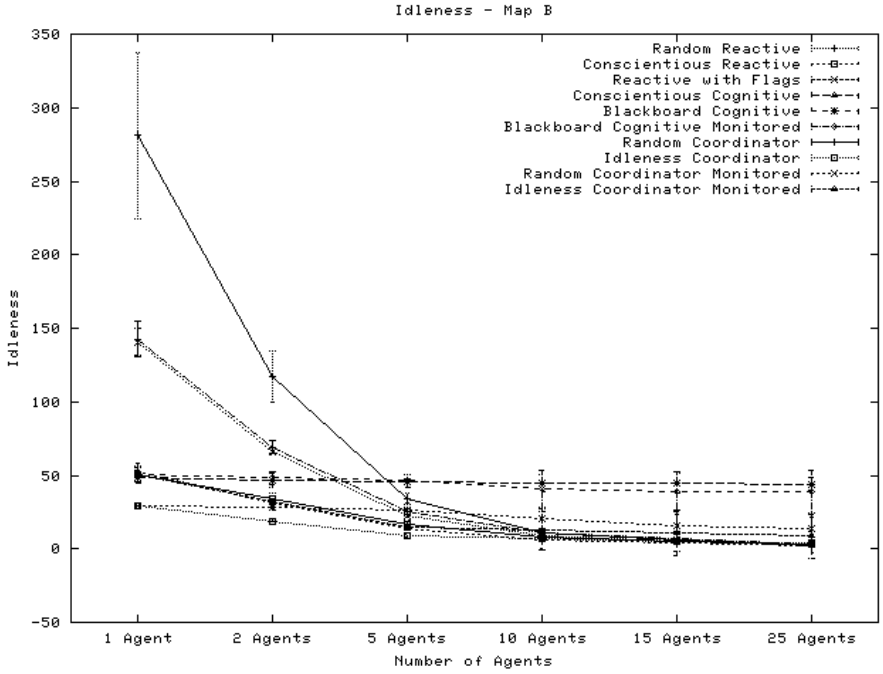


Figura 5.4 – Figura apresenta a **ociosidade** e os seus desvios para os experimentos realizados como **mapa B**.

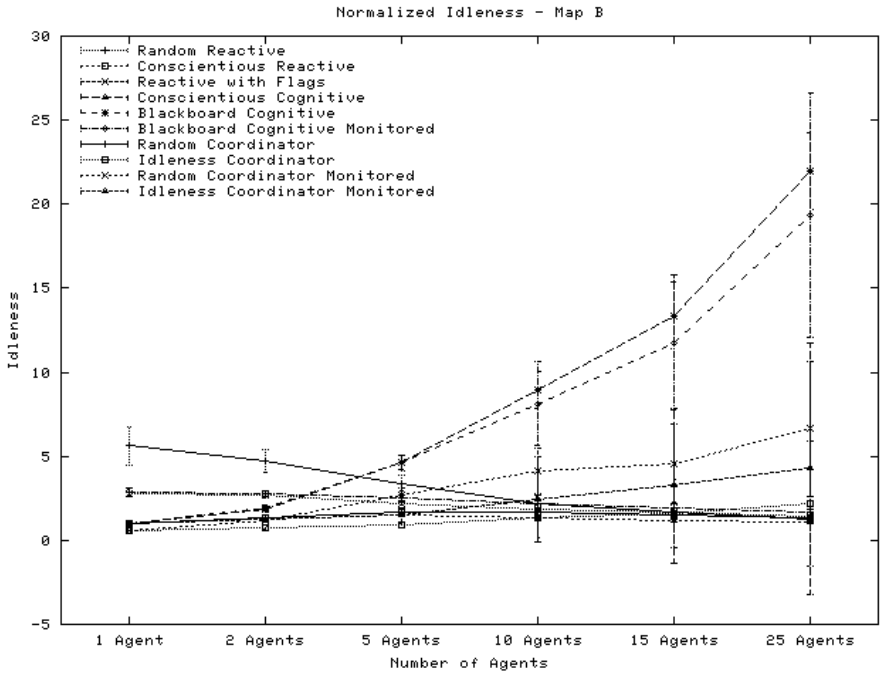


Figura 5.5 – Figura apresenta a **ociosidade normalizada** e os seus desvios para os experimentos realizados como **mapa B**.

Os gargalos colocados no mapa B influenciaram nos resultados de todas as arquiteturas, onde o desempenho foi sempre pior. As arquiteturas mais afetadas pelas características do mapa B foram àquelas baseadas na escolha aleatória dos nós.

Quanto à questão da participação individual de cada agente, existiu também uma diferença quanto aos resultados conseguidos no mapa A, onde os efeitos da coordenação foram percebidos a partir da população com mais de cinco agentes.

Tanto no mapa A quanto no mapa B constatamos que as arquiteturas aleatórias obtiveram os maiores desvios. Portanto, estas arquiteturas não apresentam um resultado uniforme para todo o grafo, sendo então encontradas grandes disparidades entre as ociosidades dos nós.

Na Figura 5.6 e na Figura 5.7 são mostrados os resultados da pior ociosidade para o mapa A.

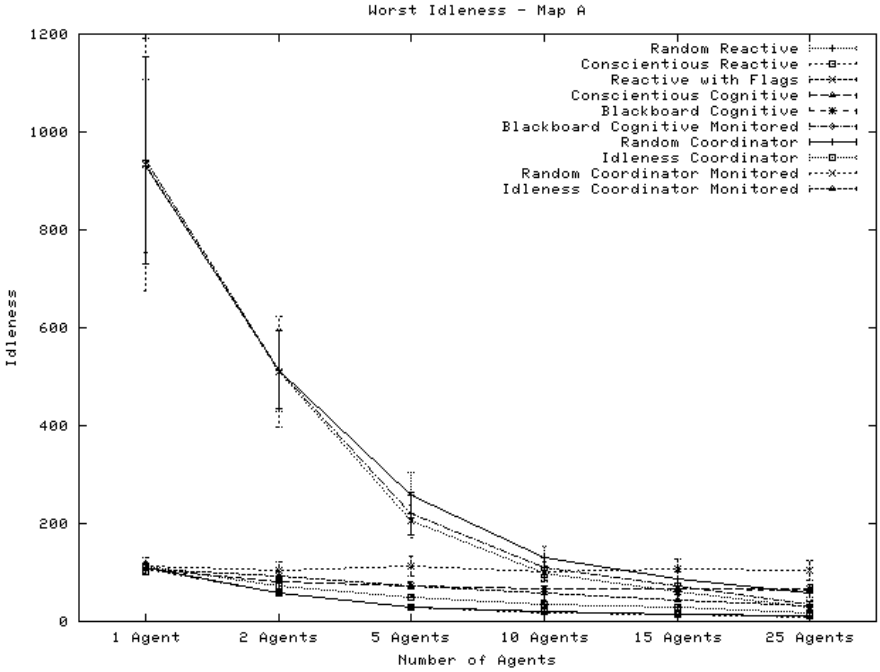


Figura 5.6 – Média e desvio padrão das **piores ociosidades** encontradas nos experimentos no **mapa A**.

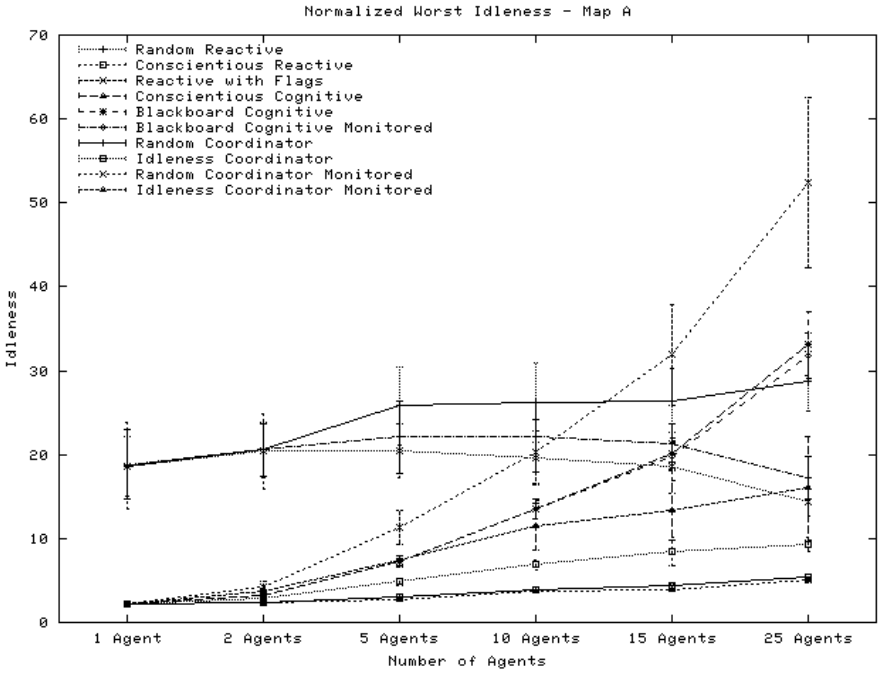


Figura 5.7 - Resultados **normalizados** para o critério de **pior ociosidade** no **mapa A**.

Os resultados para o critério do pior caso têm um formato bem parecido com o critério anterior, a ociosidade média. Assim, os comentários também são bem semelhantes. O aumento da população não altera os resultados das arquiteturas onde existe o compartilhamento da informação e não existe um mecanismo de coordenação.Para as demais arquiteturas o aumento da população sempre melhora o desempenho final.

Nos resultados normalizados notamos que os agentes baseados na escolha aleatória são os que apresentam o efeito de uma melhora mais acentuada devido ao mecanismo de coordenação dito emergente, mas tal melhora é pequena e se dá com populações maiores que cinco agentes. Para os agentes que tomam suas decisões baseadas na ociosidade podemos ver que o mecanismo de coordenação não influencia muito na melhora dos resultados, sendo isto principalmente conseguido por causa do aumento da quantidade de agentes.

Na Figura 5.8 e na Figura 5.9 são mostrados os resultados da pior ociosidade para o mapa B.

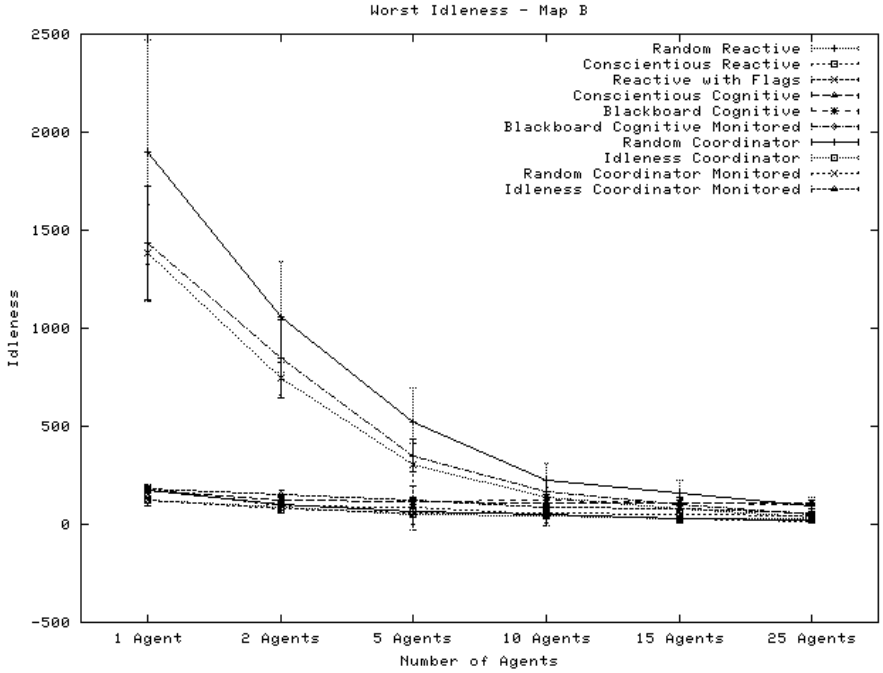


Figura 5.8 – Média e desvio padrão das **piores ociosidades** encontradas nos experimentos no **mapa B**.

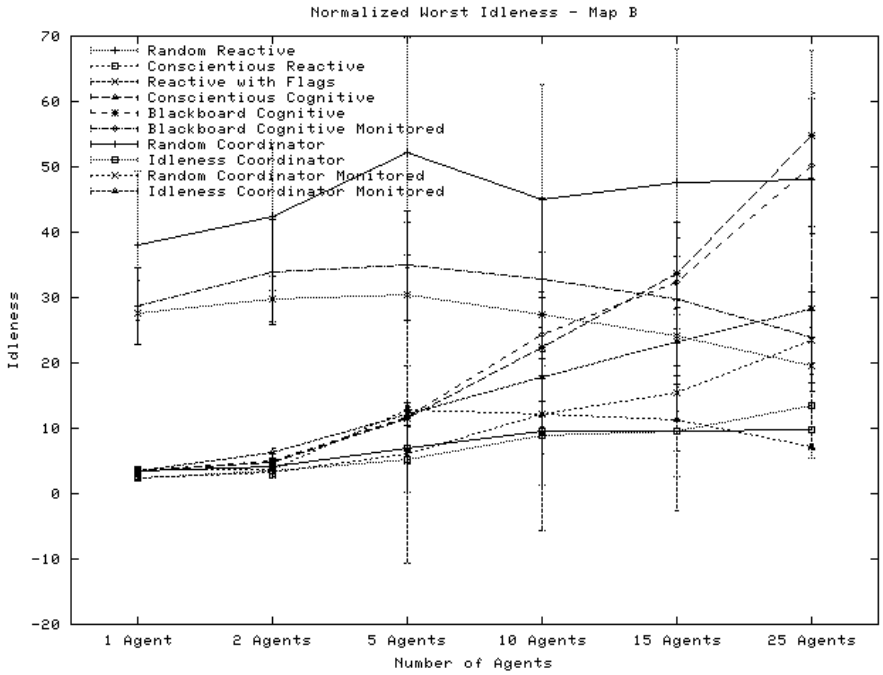


Figura 5.9 – Resultados **normalizados** para o critério de **pior ociosidade** no **mapa B**.

Além da já conhecida piora nos resultados e o aumento nos desvios que são provocados pelos gargalos do mapa B, constatamos que as arquiteturas que tinham uma abordagem local, como é o caso da arquitetura *conscientious reactive,* tiveram melhores resultados nesse tipo de mapa, superando as outras arquiteturas e tendo até resultados melhores que no mapa A. Principalmente para os casos com populações menores. Outro fato interessante foi o grande aumento dos desvios em relação ao mapa A para a arquitetura *idleness coordinator monitored*. Não houve outras mudanças quanto às observações feitas com os resultados do mapa A.

Estas observações mostram que estratégias locais dão melhores resultados para mapas onde o acesso a determinadas regiões seja difícil, ou seja, onde existam longos caminhos a serem percorridos. De fato, na estratégia local, os nós mais próximos são visitados antes dos nós que estão mais distantes, o que é uma vantagem para este tipo de mapa.

Na Figura 5.10 e na Figura 5.11 são mostrados os resultados, para o mapa A, do último critério de avaliação, o tempo de exploração.

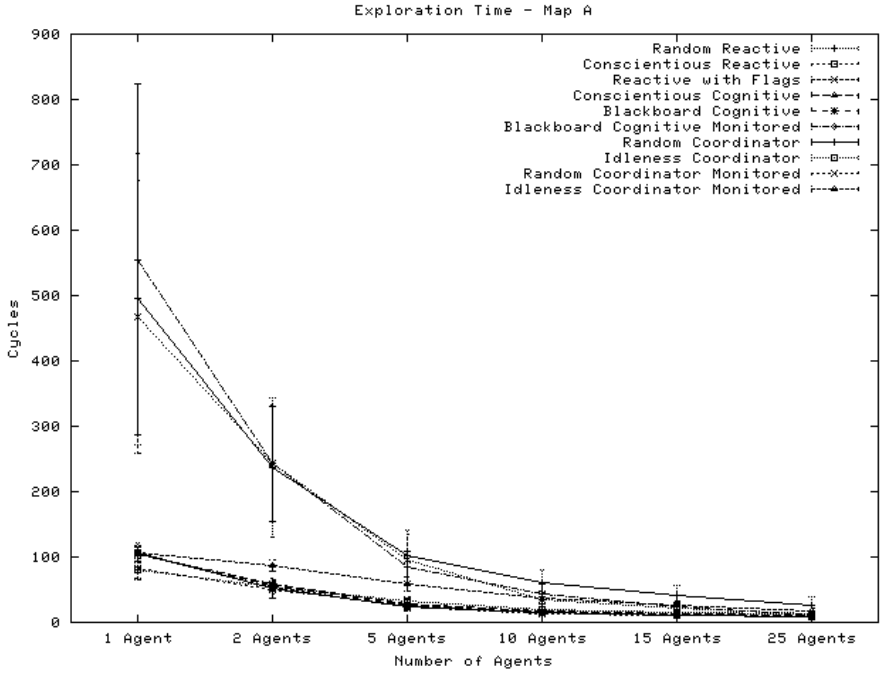


Figura 5.10 – Tempo de exploração para o **mapa A**.

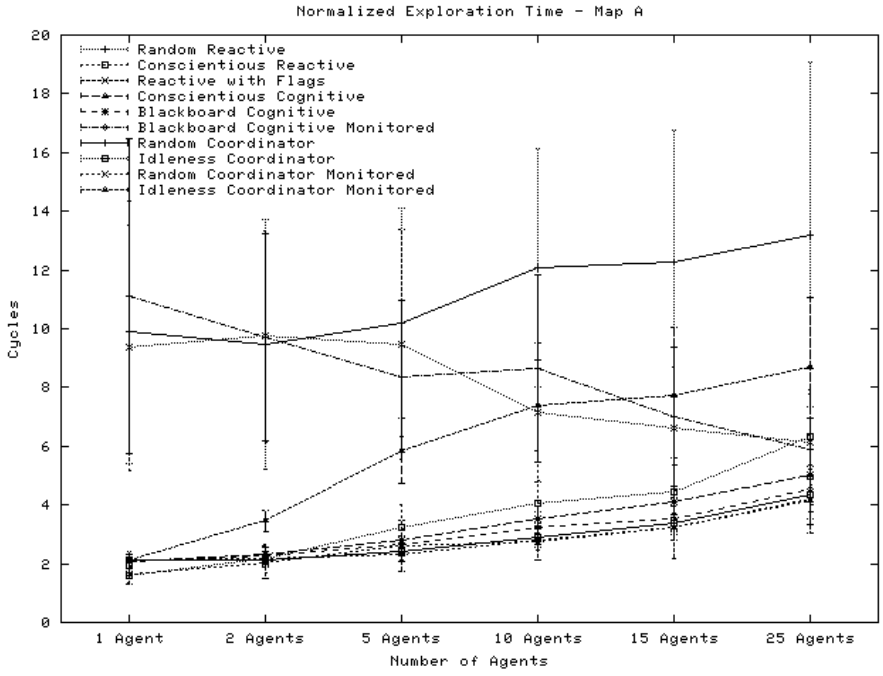


Figura 5.11 – Tempo de exploração normalizado para o **mapa A**.

Neste critério de avaliação as arquiteturas ficaram divididas em dois grupos. Com os piores resultados ficaram as abordagens aleatórias e os melhores resultados foram conseguidos pelas outras arquiteturas. Todas com formas bem semelhantes. Sendo diferente apenas a arquitetura *conscientious cognitive* que ficou numa região intermediária. Isto é facilmente notado no gráfico.

Os resultados normalizados mostram que não há muita alteração com o aumento da população. Tendo uma melhora muito pequena para os casos aleatórios e uma piora também muito pequena para as outras arquiteturas.

Na Figura 5.12 e na Figura 5.13 são mostrados os resultados do tempo de exploração para o mapa B.

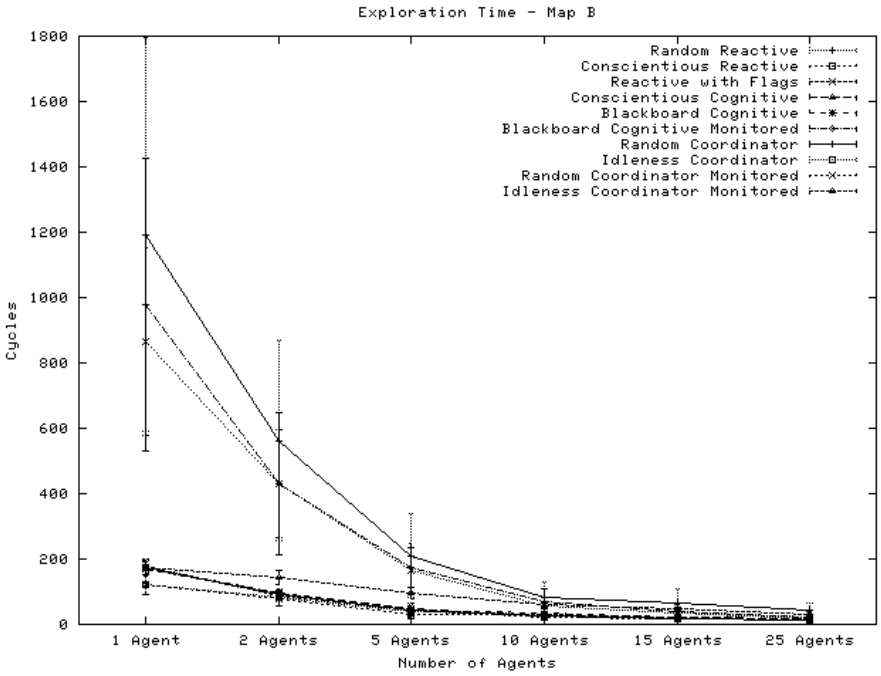


Figura 5.12 – Tempo de exploração para o **mapa B**.

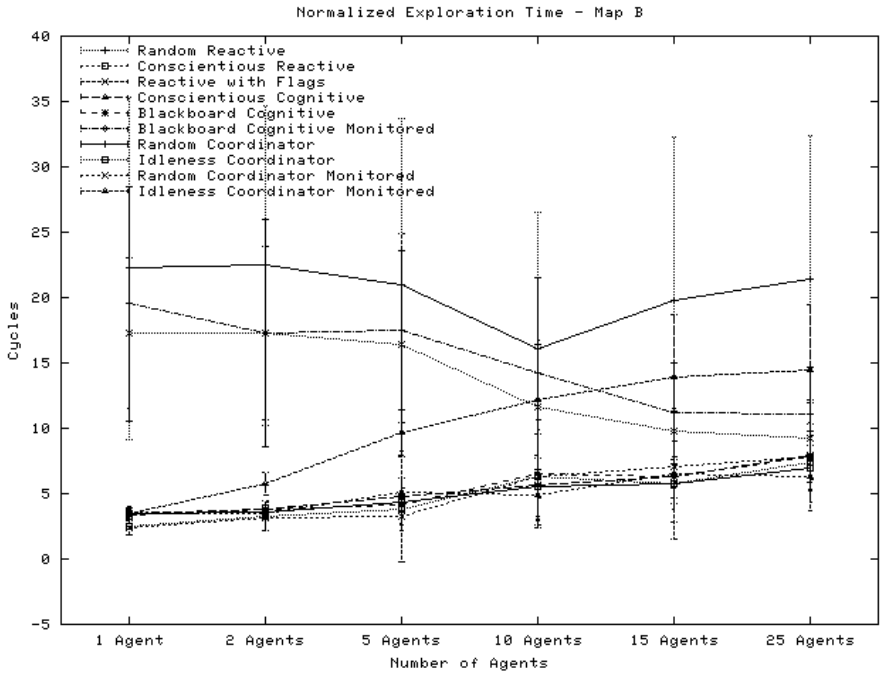


Figura 5.13 – Tempo de exploração normalizado para o **mapa B**.

As únicas alterações que aconteceram com tempo de exploração, em virtude da mudança de mapa foram um aumento nos valores médios e nos desvios, mas todas as demais características dos gráficos foram mantidas. Outras discussões serão apresentadas na seção seguinte.

5.3. Discussão

De uma perspectiva geral, nota-se nos gráficos apresentados na seção 5.2 que as arquiteturas estão divididas em três grupos distintos, os quais nós chamamos de: grupo I, grupo II e grupo III. Tal agrupamento é facilmente notado por causa do comportamento semelhante de determinados agentes, que logicamente foram colocados num mesmo grupo. A Tabela 5.1 mostra a composição de cada grupo.

Grupos	Agentes
Grupo I	*Conscientious Reactive*
	Conscientious Cognitive
	Idleness Coordinator
	Idleness Coordinator Monitored
Grupo II	*Random Reactive*
	Random Coordinator
	Random Coordinator Monitored
Grupo III	*Reactive with Flags*
	Blackboard Cognitive
	Blackboard Cognitive Monitored

Tabela 5.1 – Descrição dos grupos. Esta tabela mostra a que grupo pertence cada agente.

O grupo I obteve os melhores resultados em todas as métricas. O agente *conscientious reactive* teve desempenho um pouco melhor que os outros agentes deste grupo na maioria dos experimentos, mas esta pequena diferença tende a zero com o aumento da população.

O grupo II apresentou os piores resultados com populações pequenas. Entretanto seus resultados tiveram uma melhora, sendo até equivalentes aos do grupo tope, com populações mais numerosas. Mas, tais resultados apresentaram um desvio padrão maior que as arquiteturas não aleatórias. Uma forte característica deste grupo é seu comportamento imprevisível, onde não existe uma fase estável bem comportada como nas outras arquiteturas. Tal propriedade é mostrada nos altos valores dos desvios padrões.

O grupo III teve um comportamento que, de certa forma, já era esperado. Os agentes tenderam a ir para o mesmo nó ao mesmo momento. Isto aconteceu porque existiu o compartilhamento da informação e não existiu nenhum mecanismo de coordenação que ordenasse as ações que os agentes deveriam executar. Conseqüentemente, esse grupo obteve praticamente os mesmos resultados para qualquer população. Seria como se existisse apenas um agente no ambiente.

Com uma comparação entre os resultados das arquiteturas de SMA nos dois mapas (mapa A *versus* mapa B), pode-se ver a influência da conectividade do grafo nos resultados. O desempenho sempre foi pior no mapa B em todos os experimentos, utilizando todas as métricas. O grupo tope foi menos afetado pelos gargalos do mapa B e obtiveram os resultados mais semelhantes nos dois mapas. O grupo aleatório foi o mais afetado, onde o mesmo obteve péssimos resultados para o mapa B.

Os resultados normalizados são bem interessantes, pois mostram a participação de cada agente, ou seja, a contribuição de cada agente para a arquitetura. Outrossim, indicam o significado da coordenação para cada arquitetura dada, mostrando se a melhora se deve ao mecanismo de coordenação adotado ou ao simples fato do aumento da população. Por exemplo, a normalização mostra como o desempenho do grupo não-coordenado sempre priora com o aumento da quantidade de agentes. Um fato curioso acontece com o tempo de exploração. O aumento da população não dá um aumento significante no desempenho individual dos agentes, não importando qual era o tipo da arquitetura de SMA que foi utilizada.

5.4. Conclusões

Mais que identificar o grupo tope, estes experimentos mostram algumas diretrizes preliminares para o projeto das arquiteturas de SMA para o problema da patrulha, bem como tarefas similares. Estas diretrizes serão apresentadas a seguir:

O primeiro e principal passo é entender o domínio de aplicação, suas restrições e características. Isto é essencial para determinar:

- O máximo de ociosidade que pode ser aceita;
- A ociosidade média desejada;

- A variação ideal para a ociosidade média;
- A possibilidade de comunicação entre os agentes;
- Capacidade de realizar monitoramento sem perdas no desempenho;
- O melhor tipo de comunicação, caso isso seja possível;
- As características do grafo (conectividade, número de nós, gargalos, etc);
- Necessidade de exploração;
- O tempo máximo para realizar a exploração.

Conhecendo estas características e restrições do domínio de aplicação e os resultados deste trabalho fica mais fácil determinar qual é a melhor arquitetura de SMA para a situação em questão. Para tal basta ver quais as arquiteturas que satisfazem as restrições do problema em questão e escolher aquela que melhor convier.

Por exemplo, as arquiteturas do grupo aleatório não são recomendadas para os grafos que contenham gargalos e variações significantes de ociosidade não sejam desejadas. Além disso, o número de agentes pode ser determinado de acordo com a ociosidade desejada (a taxa de um agente para cada dez nós já possibilita bons resultados).

6
Trabalhos Relacionados

Como já mencionado anteriormente, apesar da sua importância, não foi encontrado um estudo que tratasse a patrulha de maneira adequada, apresentando uma análise profunda e sistemática. Mas, apesar disso esse capítulo mostra alguns trabalhos que poder ser relacionados, de certa forma, com o problema em questão.

6.1. Patrulha em Jogos de Computador

Um dos poucos lugares onde encontramos trabalhos relacionados com a patrulha, tratada com seu real significado (da palavra), foi na indústria de jogos de computador. Isso aconteceu devido ao fato de existirem situações, nesse tipo de aplicação, com características similares às encontradas na patrulha tal como a conhecemos.

Para a literatura desta área, patrulhar consiste em mover-se entre uma lista de pontos especificados a priori (Figura 6.1), simulando assim uma patrulha [22]. Existem algumas variações que procuram deixar o comportamento um pouco mais “inteligente”, tal como sortear o ponto inicial depois de percorrer todos os pontos da lista, sortear novamente o primeiro ponto e percorrer, assim por diante. Outra solução encontrada, para o mesmo

problema, foi simplesmente deixar as unidades paradas enquanto não acontecia nada de relevante para elas [22].

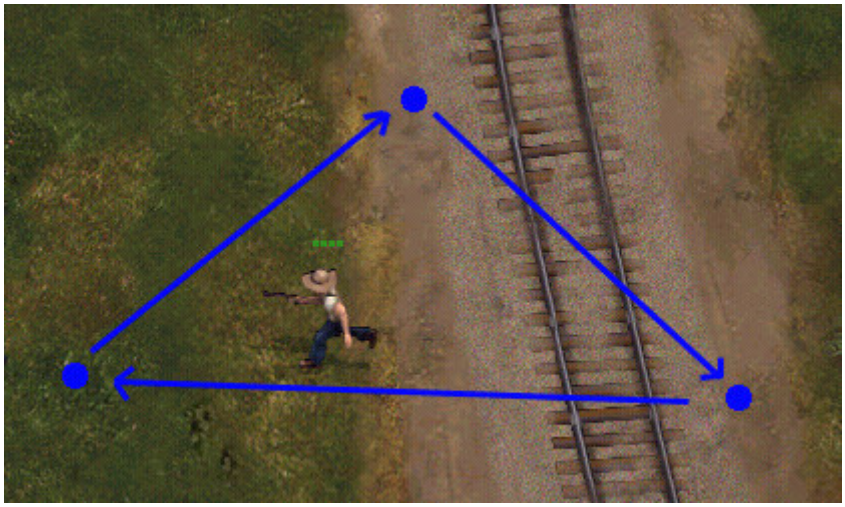


Figura 6.1 – Exemplo de uma patrulha realizada num jogo

Em jogos que envolvem a busca de algo ou mesmo da saída num labirinto, encontramos algoritmos simples, tal como: a regra da mão direita, segundo o qual é possível sair de um labirinto ou percorrê-lo totalmente seguindo sempre a parede direita, desde que não existam ilhas nesse labirinto [12]. Isso geralmente é aplicado quando a máquina comanda os inimigos num labirinto e o jogador é o que esses inimigos estão querendo encontrar.

Outros trabalhos se preocupam com a movimentação em grupo e utilizam técnicas de formação que são baseadas em posições relativas entre as unidades do grupo [41] e [42]. Tais abordagens têm como destaque técnicas de: *path-finding*, detecção de colisões, previsão de posição e formação de grupo.

A base funcional da coordenação de movimentação, definida no parágrafo anterior, é a utilização da previsão de posições dos membros do grupo para manter uma formação e, ao mesmo tempo, evitar colisões com obstáculos e entre os membros do grupo. Isso cria uma movimentação bem

mais inteligente, para um grupo de unidades, sendo esse o principal objetivo da coordenação de movimentação de Pottinger.

Todos esses trabalhos, encontrados na área de jogos, que abordam a patrulha e movimentação têm o objetivo principal de simular um comportamento bem determinado, assim não tratam o problema definido neste trabalho.

6.2. Comportamentos de Navegação

Uma outra abordagem, baseada em comportamentos, foi encontrada para solucionar os problemas de movimentação ou navegação, tal abordagem foi batizada pelos pesquisadores dessa área como: sistemas robóticos baseados em comportamento [2][3] e comportamentos de navegação [44].

A capacidade de navegação é conseguida, nestes sistemas, através da composição de comportamentos simples. Esses comportamentos são baseados na informação sensorial instantânea e os mesmos são canalizados de forma imediata para o sistema motor, responsável pela locomoção. Isso mostra a natureza reativa desses comportamentos.

Durante a fase de levantamento bibliográfico chegamos até a implementar alguns dos comportamentos descritos por Reynolds, um exemplo da aplicação criada é mostrado na Figura 6.2.

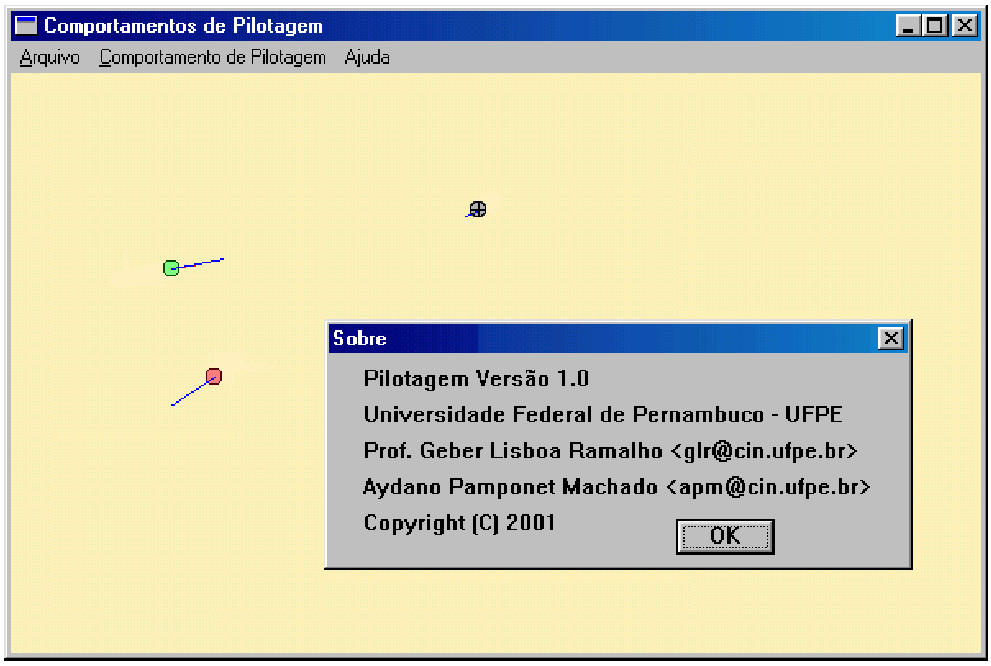


Figura 6.2 – Implementação de alguns comportamentos de navegação

As unidades ou veículos foram modelados de uma maneira bem simples com as seguintes características: ponto de massa definido pela posição e a massa, velocidade, força máxima, velocidade máxima e uma orientação. As atualizações nessas propriedades de cada unidade são feitas de acordo com as leis básicas da física, tais como: *aceleração = força / massa, nova_posição = posição_atual + velocidade* e etc.

São alguns dos comportamentos: *buscar/fugir; perseguir/evadir; perseguir com intervalo; chegada; desvio de obstáculos; vagar; seguir caminho; acompanhar parede; seguir campo; evitar colisão; separação; coesão; alinhamento.*

Alguns comportamentos serão detalhados a seguir:

- *Buscar/fugir*, dado o alvo fixo é calculado uma força para ser aplicada ao veículo para direcioná-lo ao alvo. Na implementação

da fuga basta utilizar o oposto da força calculada, isso irá colocá-lo na direção oposta;

- *Chegada*, esse comportamento é idêntico à busca quando o veículo esta longe do alvo, diferenciando-se apenas ao chegar perto do alvo, pois sua velocidade vai diminuindo gradativamente até parar no mesmo;
- *Vagar*, para dar esse tipo de comportamento aleatório é sorteado um ponto de uma esfera (3D) ou circunferência (2D) colocado na frente do veículo, isso serve para evitar uma rotação brusca do veículo (Figura 6.3);

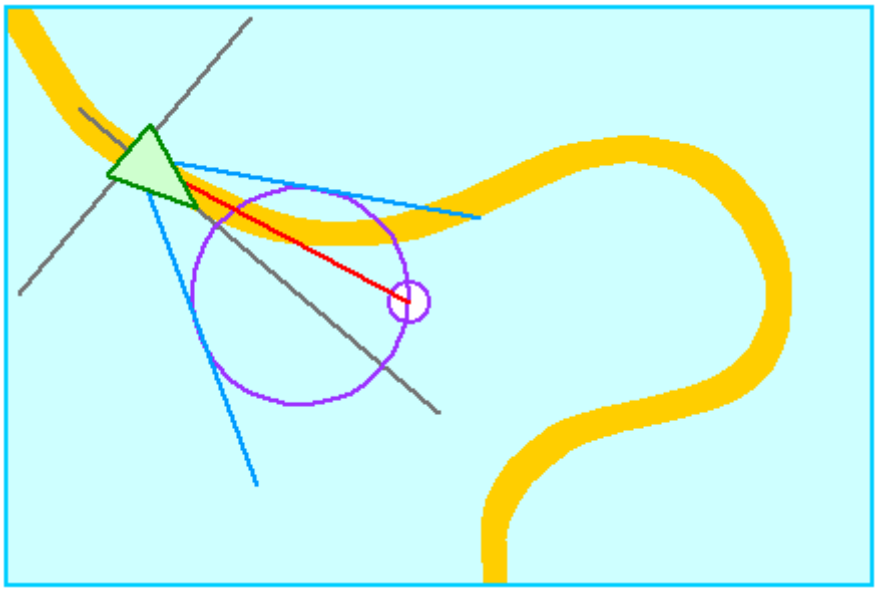


Figura 6.3 – Veículo vagando

Esses comportamentos podem ser combinados para gerar outros tipos de comportamentos, como é o caso de combinar a característica de vagar com a de busca temos um comportamento parecido com uma exploração.

As pesquisas dessa área têm como principal preocupação a criação de novos comportamentos, bem como a adaptação e utilização dos mesmos nos ambientes mais variados.

Esses tipos de agentes, baseados em técnicas de movimentação, mostram ótimos resultados para movimentar agentes em ambientes hostis. Isso se dá graças à reatividade desses agentes, que é muito útil em situações imprevisíveis que exigem uma decisão rápida. Sendo esse o principal objetivo dessa área. Tais características podem ser utilizadas para implementar a movimentação dos agentes para uma patrulha, sendo esta uma solução sem muito recurso que não dá o devido tratamento para o problema proposto neste trabalho.

6.2.1. Multidões Virtuais

Também com uma preocupação comportamental para um grupo de indivíduos, existem trabalhos especificamente voltados para os problemas que aparecem quando a quantidade de indivíduos cresce e atinge as chamadas multidões. Tais trabalhos, batizados de multidões virtuais, têm aplicação nos campos de: entretenimento, simulação da movimentação de multidões, povoamento e simulação comportamental [36].

Essas aplicações exigem alguns requisitos e restrições, intrínsecos a própria natureza do problema, que também já são diferentes dos requisitos e restrições dos sistemas de agentes individuais.

Foram encontrados vários trabalhos e arquiteturas para manipular e interagir com as multidões [13][37], mas nenhum deles trata do problema da patrulha. Até porque esse problema não esta muito relacionado, a não ser que a patrulha seja realizada por uma grande quantidade de agentes, sendo esta quantidade enquadrada na escala das multidões.

6.3. Agentes para Redes de Computadores

As redes de computadores, encontradas hoje, crescem e se modificam bastante em escala, escopo e importância. Tais características aumentaram a complexidade de configurar, administrar e utilizar esse tipo de rede. Devido a este fato, muitas pesquisas têm procurado alternativas para auxiliar nessas tarefas que se tornam cada vez mais complexas. Em particular, algumas pesquisas tomaram como foco a criação de sistemas de cooperação com agentes móveis. Agentes móveis são programas que têm a liberdade de se locomover, autonomamente, de um computador para outro continuando sua linha de execução [52].

Dentro deste enfoque, existem trabalhos que procuram auxiliar em tarefas administrativas da rede [1]. Na abordagem destes trabalhos, o gerenciamento não é mais centralizado no administrador da rede e sim pulverizado e distribuído através dos agentes que auxiliam na detecção e correção de eventuais problemas.

Também encontramos trabalhos interessantes preocupados com a topologia ou disposição física da rede. Em particular, a utilização de agentes móveis cooperativos para o mapeamento de redes de computadores [35] se mostrou interessante para o escopo da patrulha pela metodologia adotada. Esse trabalho tirou proveito da mobilidade, modularidade, flexibilidade e adaptabilidade das arquiteturas multiagentes, que podem ser criadas para realizar o mapeamento. Com este mapeamento realizado, decisões sobre rotas nessa rede podem ser tomadas de uma forma mais eficiente e segura.

O ponto mais interessante encontrado nestes trabalhos foi a metodologia adotada para testar e avaliar os agentes em cada tarefa, onde essa partiu de agentes mais simples para os mais complexos. Isto validou os

resultados e a nova abordagem dada para estes problemas. Entretanto esses trabalhos estão relacionados ao problema da exploração.

6.4. Bar el Farol

Mudando para o campo da economia, encontramos um problema que gera um confronto interessante com o problema da patrulha.

O problema do "*Bar El Farol*" [5] foi definido da seguinte forma: N pessoas decidem de forma independente, a cada semana, se elas devem ir ou não para um bar que oferece entretenimento em certa noite. O espaço é limitado e somente é divertido se o bar não estiver muito cheio. Não existe a possibilidade de se saber, a priori, o número de pessoas que irão para o bar. Não existe a possibilidade de comunicação entre as pessoas. A única informação disponível é o número de pessoas que foram para o bar nas semanas anteriores. Assim, como cada pessoa ou agente deve atuar de maneira a maximizar o divertimento e minimizar as saídas frustradas?

O autor mostra soluções através de previsões baseadas no histórico das semanas anteriores. Estas soluções são enquadradas nos campo de estudo do raciocínio indutivo e da racionalidade limitada, que são bem aplicados para situações onde não existe a possibilidade do raciocínio dedutivo e da lógica. O autor criou um modelo dinâmico onde cada agente começa com uma hipótese particular ou previsão e esta vai sendo ajustada com a experiência, ou seja, com o passar das semanas.

O problema do "*Bar El Farol*" é similar ao da patrulha, na medida que a proposta é deixar o número de visitas abaixo de um certo limiar de aceitação. Poderia-se imaginar um bar em cada nó do grafo. No caso da patrulha o objetivo é deixar a ociosidade no seu valor mínimo. Este trabalho mostra uma solução interessante, mas apenas para esse tipo de aplicação onde apareçam todas as restrições que foram descritas com o problema, como é o caso da

existência de um histórico dos acontecimentos anteriores. Essa abordagem também não leva em consideração os outros parâmetros colocados nas arquiteturas de agentes discutidas neste trabalho de mestrado, são alguns deles: comunicação, monitoramento, planejamento, coordenação.

6.5. Conclusões

De acordo com o que foi apresentado nas seções anteriores deste capítulo, podemos dizer que:

- Os jogos de computador tratam a patrulha como tal, isto é, no real significado da palavra, mas utilizam soluções muito simples para tratar o problema;
- Na área de redes de computadores encontramos soluções mais sofisticadas direcionadas para a administração e gerenciamento, mas não tratam um problema com as características da patrulha sendo os estudos mais relacionados com o problema da exploração;
- Os comportamentos de navegação podem ser uma abordagem interessante, mas não resolvem o problema da patrulha de uma forma direta mantendo suas principais características;
- O problema do *Bar el Farol* é interessante por ter algumas características semelhantes as do problema da patrulha, mas junto com essas características vêem uma série de restrições que o reduz a um caso especial de patrulha.

Em SMA, de acordo com nossas pesquisas do estado da arte o problema da patrulha não foi estudado com suficiente profundidade e método, o que motiva ainda mais este trabalho de mestrado.

Com o problema bem definido e especificado e o estado da arte mostrado pudemos traçar uma metodologia de trabalho que foi seguida. Todos os elementos, dessa metodologia, serão explicados no capítulo seguinte.

7
Conclusões

Concebida de forma sistemática e metodológica, esta pesquisa teve a originalidade como uma forte característica, pois a mesma apresentou definições e resultados que, até onde sabemos, não eram conhecidos para problema da patrulha multiagente, cuja relevância foi justificada anteriormente.

O objetivo principal foi fazer um estudo sistemático e metodológico do problema da patrulha multiagente, com a pretensão de dar algumas diretrizes para a implementação de soluções. Tal objetivo foi alcançado do seguinte modo:

- Realizando um estudo profundo do problema em questão;
- Criando uma abstração para o problema, definindo-o através dessa abstração;
- Definindo métricas para a avaliação do desempenho das arquiteturas de agentes que executaram a patrulha;
- Propondo algumas arquiteturas de agentes para patrulha;
- Estabelecendo os cenários de teste;

- Realizando experimentos e mostrando diretrizes para a utilização das arquiteturas de agentes após uma análise dos resultados conseguidos.

Todas essas atividades investidas sobre os objetivos, tal como mencionadas anteriormente, foram contribuições relevantes para o estado da arte do problema da patrulha e de outros similares.

Um outro resultado interessante e alcançado no desenvolvimento deste trabalho foi uma maior interação de pesquisa entre o nosso grupo no Centro de Informática (CIn) da Universidade Federal de Pernambuco (UFPE) e o do Laboratório de Informática (LIP6) da Universidade Pierre et Marrie Currie (PARIS VI). Tão logo tomaram conhecimento da nossa pesquisa, os pesquisadores do LIP6 passaram a ter interesses no problema da patrulha, passando a engajarem-se no desenvolvimento deste trabalho. Como resultado dessa interação, publicamos dois artigos, tal como pode ser observado nas referências [30], [31].

7.1. Trabalhos Futuros

Pretendemos aprofundar os estudos apresentados neste trabalho com a realização do doutorado, do autor da presente dissertação, no LIP6. Nesta pesquisa de doutorado, a idéia central é utilizar aprendizagem de máquina para inserir agentes, no ambiente construído, com a capacidade de aprender como seria a melhor maneira de efetuar a patrulha. Tal característica deixará esse tipo de agente com a capacidade de se adaptar a uma maior variedade de situações.

Uma outra vertente de estudo, já iniciada pelo aluno de mestrado do CIn, Alessandro Almeida, está focada em estudar mais a fundo as características dos grafos, verificando como cada característica influencia as arquiteturas de agentes. A idéia é criar um *benchmark* onde os projetistas de

soluções multiagentes, para a patrulha, tenham as diretrizes de acordo com as características do grafo do seu problema, sendo assim bem mais específico.

Pretende-se criar arquiteturas mais complexas, continuando assim o princípio adotado. As novas arquiteturas terão, entre outras, as seguintes melhorias:

- Técnicas mais avançadas de *path-finding*;
    - Por exemplo: levando em contra outros critérios (ociosidade dos nós, antecipação, etc.), além da distância entre os nós do grafo.
- Decisões mais astutas para a escolha do próximo objetivo;
    - Como por exemplo, levando em conta a distância e a ociosidade dos nós.
- Coordenação descentralizada;
    - Isso pode ser conseguido, por exemplo, com técnicas de negociação.
- Variações no campo de visão dos agentes.

E por fim, a construção de um sistema de patrulha multiagente que seja aplicado numa situação prática e real, validando ainda mais esta pesquisa aqui apresentada.

Apêndice A. Figuras Numa Outra Escala

Nesse apêndice são mostrados detalhes dos gráficos do capítulo 5, que não foram vistos por conta da escala.

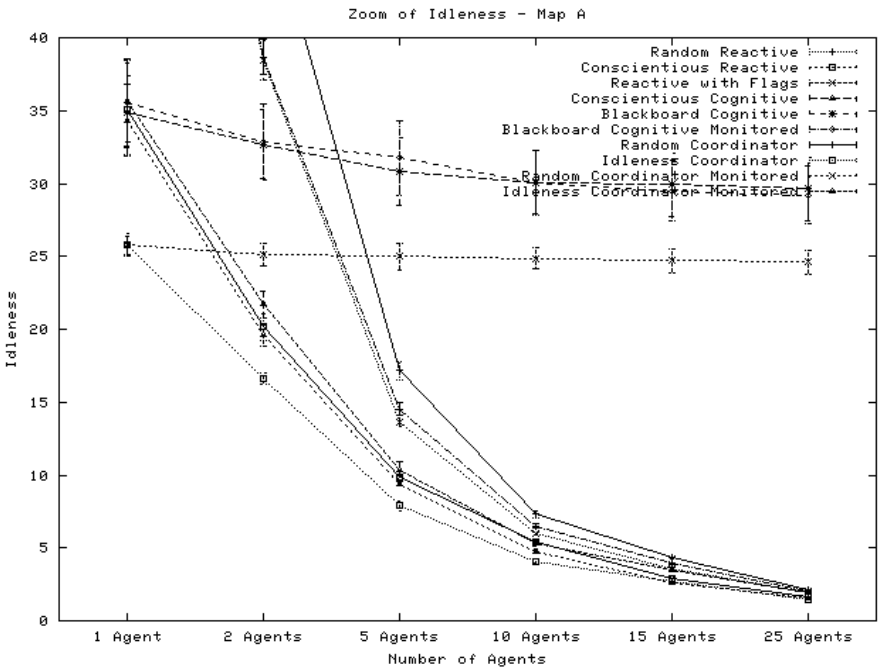


Figura A. 1 – Figura apresenta a ociosidade e os seus desvios para os experimentos realizados como mapa A.

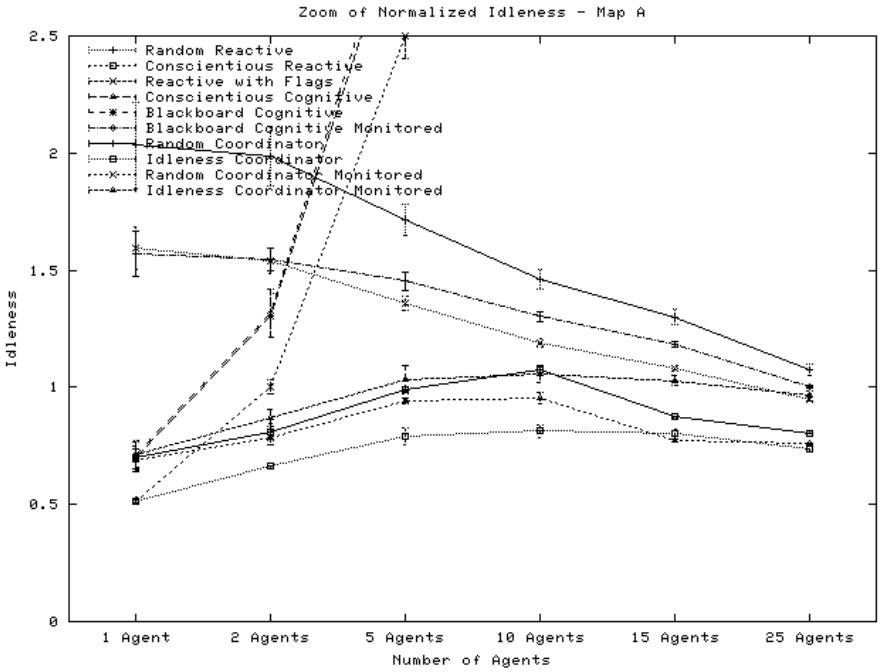


Figura A. 2 – Resultados normalizados da ociosidade conseguida nos experimento com os agentes no mapa A.

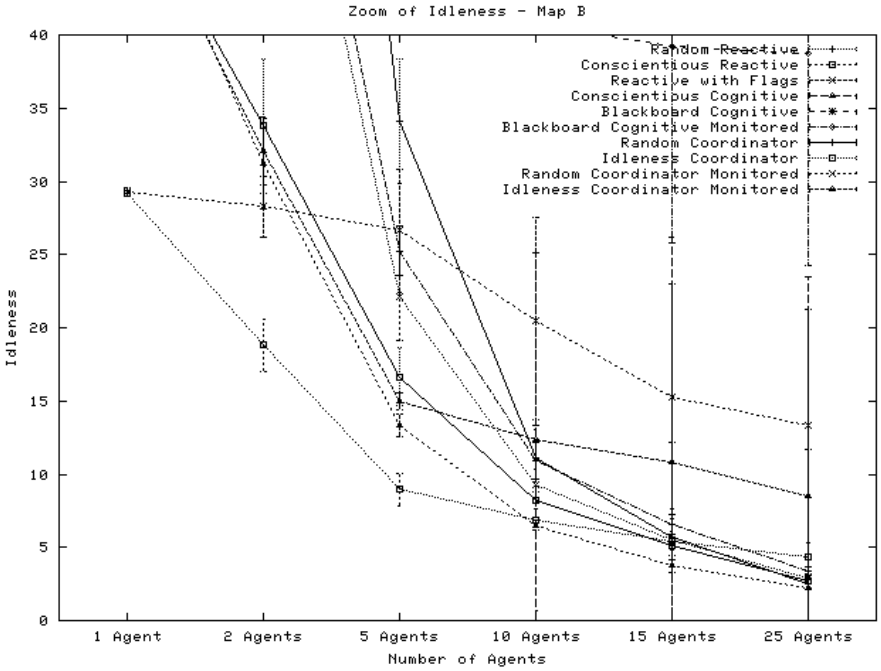


Figura A. 3 – Figura apresenta a ociosidade e os seus desvios para os experimentos realizados como mapa B.

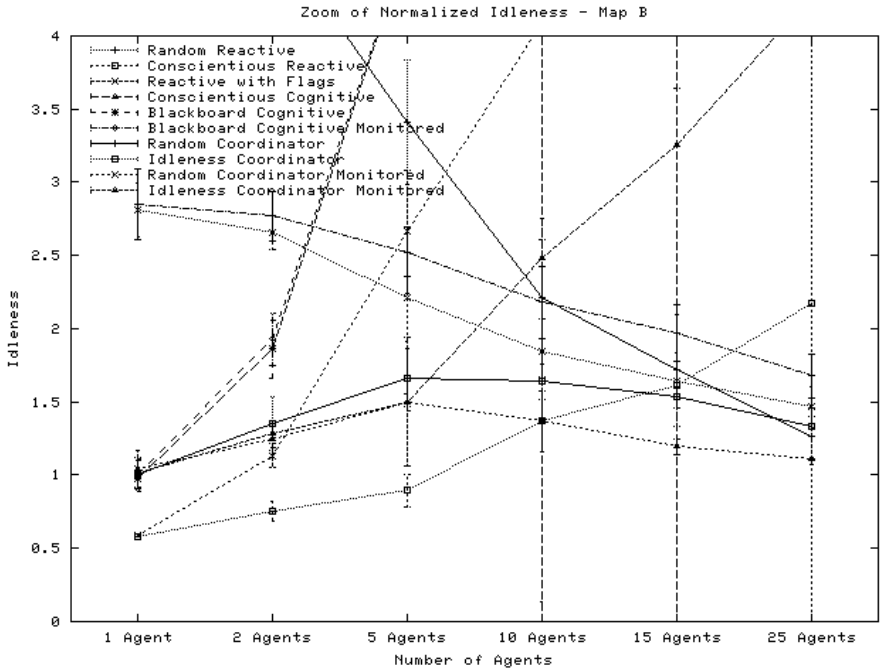


Figura A. 4 – Resultados normalizados da ociosidade conseguida nos experimento com os agentes no mapa A.

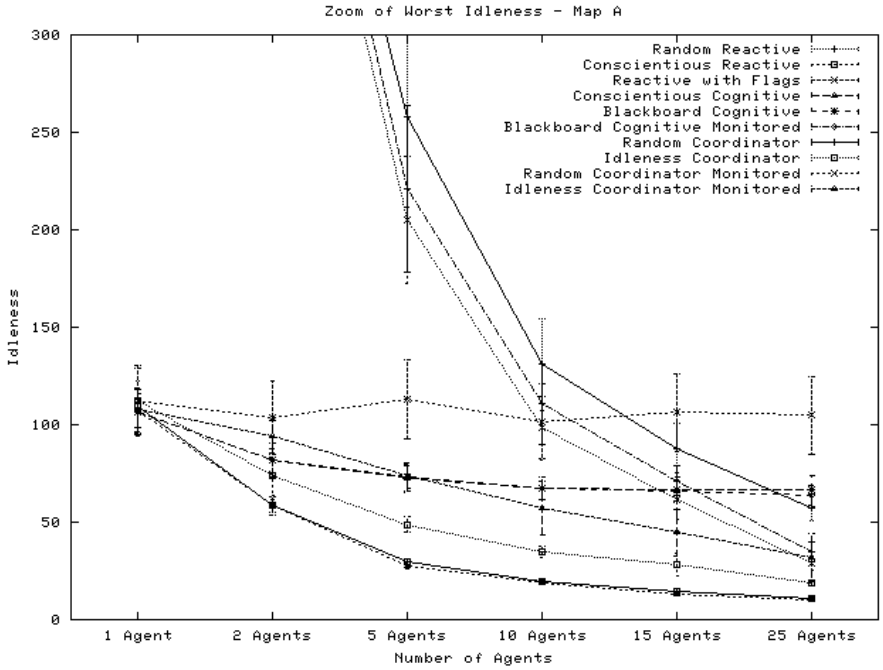


Figura A. 5 – Média e desvio padrão das piores ociosidade encontradas nos experimentos no mapa A.

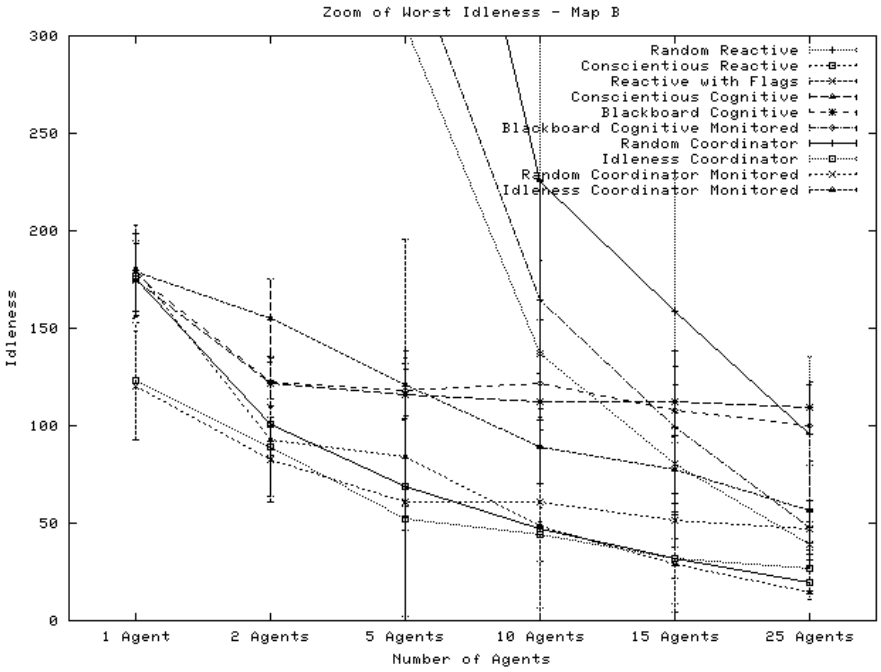


Figura A. 6 – Média e desvio padrão das piores ociosidade encontradas nos experimentos no mapa B.

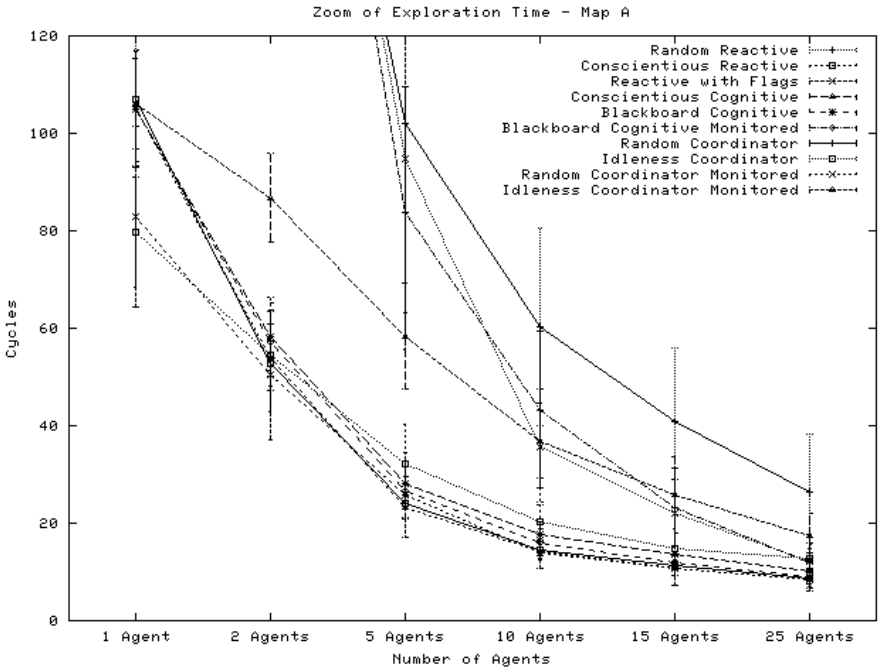


Figura A. 7 – Tempo de exploração para o mapa A.

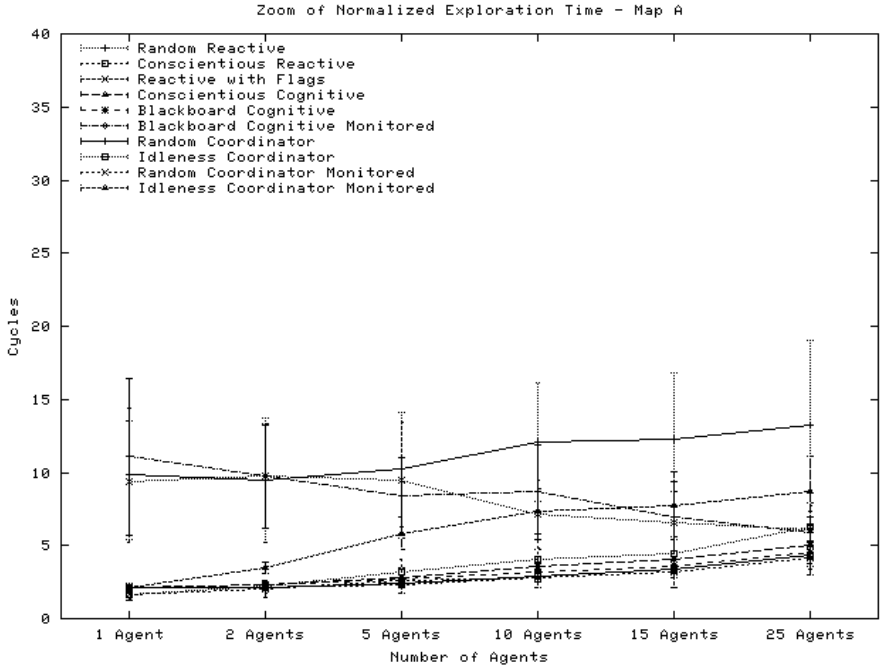


Figura A. 8 – Tempo de exploração normalizado para o mapa A.

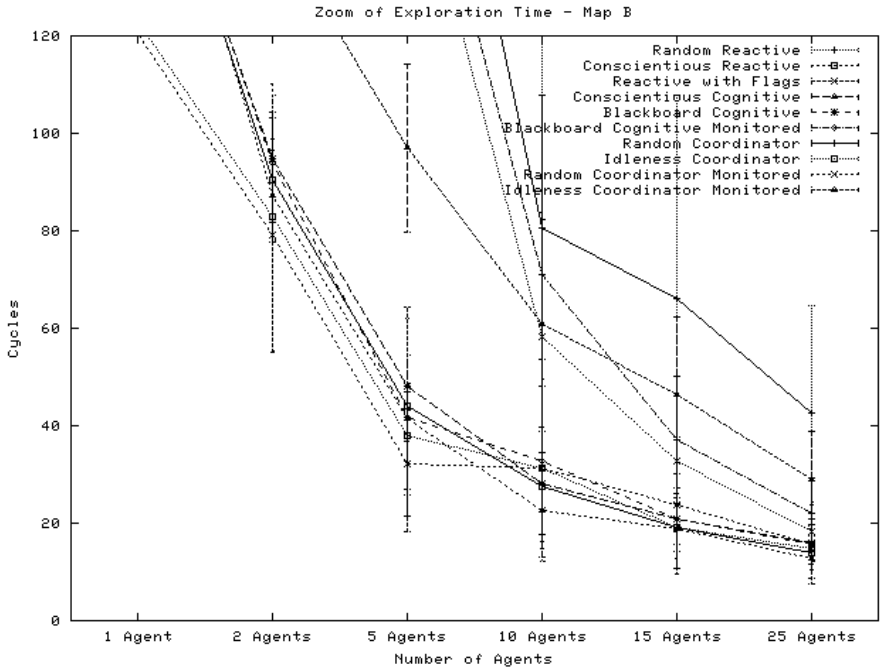


Figura A. 9 – Tempo de exploração para o mapa B.

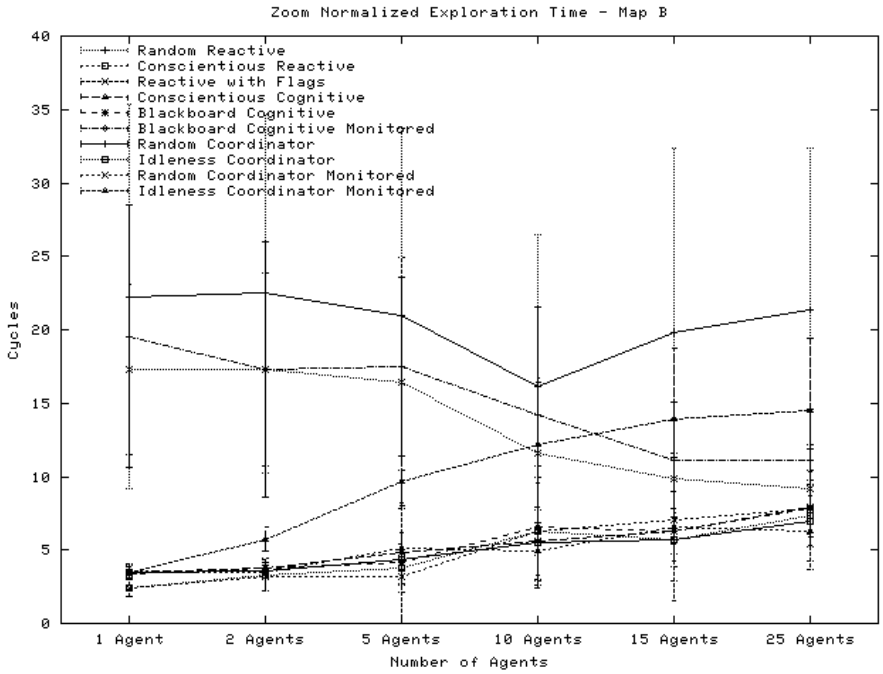


Figura A. 10 – Tempo de exploração normalizado para o mapa B.

Referências Bibliográficas

[1] Andrade, R. de C., Macedo, H. T., Ramalho, G. L., & Ferraz, C. A. G. (2001). Distributed Mobile Autonomous Agents in Network Management. *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*.

[2] Arkin, Ronald C. (1992). Behavior-Based Robot Navigation for Extended Domains. *Adaptive Behaviors, vol. 1(2)*, 201-225.

[3] Balch, Tucker & Arkin, Ronald C. (1999). Behavior-Based Formation Control for Multi-robot Teams. *IEEE Transactions on Robot and Automation vol. XX.*

[4] Bates J. (1994). The role of emotion in believable agents. *Communications of the ACM, vol. 37(7),* 122–125.

[5] Brian, Artur W. (1994). Inductive Reasoning and Bouded Retionality (The El Farol Problem). *American Economic Review (Papers and Proceedings), 84,* 406-411.

[6] Burmeister B., Haddadi A., & Matylis G. (1997) Applications of multi-agent systems in traffic and transportation. *IEEE Transactions on Software Engineering, vol. 144(1)*, 51–60.

[7] Chavez A. & Maes P. (1996) Kasbah: An agent marketplace for buying and selling goods. *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96), London, UK,* 75–90.

[8] Cho J., & Garcia-Molina, H. (2000). Synchronizing a database to Improve Freshness. *Proceedings of 2000 ACM International Conference on Management of Data (SIGMOD).*

[9] Ciancarini, P, Omicini, A. & Zambonelli F. (2000). Multiagent System Engeneering: The Coordination Viewpoint. *Inteilligent Agents VI, LNAI 1757, Edited by N. R. Jennings and Y. Lespérance,* 250-259.

[10] Davis R., & Smith R. (1983). Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence 20,* 63-109.

[11] Dorigo, M. Maniezzo, V. & Coloni, A. (1996). The Ant System: optimization by a colony of cooperating agents. *IEEE Tarns. System, Man and Cybernetics B26(1)*, 29-41.

[12] Editora Nova Cultural (1987). *INPUT – Curso Prático de programação de Computadores – n 8, Torne o Jogo Mais Difícil*, 153-160.

[13] Farenc, N., Raupp Musse, S., Schweiss, E., Kallmann, M., Aune, O., Boulic, R. & Thalmann, D. (2000). A Paradigm for Controlling Virtual Humans in urban Environment Simulations. *Applied Artificial Intelligence Journal - Special Issue on Intelligent Virtual Environments , V. 14, N. 1,* 69-91.

[14] Fenster, M., Kraus S. & Rosenschein, J. S. (1997). Coordination without Communication: Experimental Validation of Focal Point Techniques. *Readings in Agents edited by Michael N. Huhns and Munindar P. Singh,* 380-386.

[15] Ferber, J. (1999). *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Addison-Wesley.

[16] Ferreira, Aurélio B. de H.. (1999). *Dicionário Aurélio Eletrônico Século XXI.* Versão 3.0.

[17] Findler N. V. & Malyankar R. M. (2000). Social Structures and the Problem of Coordination in Intelligent Agent Societies. *16-th IMACS World Congress (© 2000 IMACS)*.

[18] Grand S. & Cliff D. (1998). Creatures: Entertainment software agents with artificial life. *Autonomous Agents and Multi-Agent Systems, vol. 1(1)*.

[19] Green S., Hurst L., Nangle B., Cunningham P., Somers F. & Evans R. (1997). *Software Agents: a review*. [On-line]. Disponível em: http://www.cs.tcd.ie/research_groups/aig/iag/toplevel2.html.

[20] Griffeth N. D. & Velthuijsen H. (1994). The negotiating agents approach to run-time feature interaction resolution. *L. G. Bouma and H. Velthuijsen, editors, Feature Interactions in Telecommunications Systems. IOS Press,* 217–235.

[21] Hayes-Roth B., Hewett M., Washington R., Hewett R., & Seiver A. (1989). Distributing intelligence within Na individual. *L. Gasser and M. Huhns, editors, Distributed Artificial Intelligence Volume II. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA,* 385–412.

[22] Howland, Geoff. *A Practical Guide to Building a Complete Game AI: Vol.II.* Lupine Games. http://www.lupinegames.com/articles/prac_ai_2.html

[23] Jennings N., Sycara K. & Wooldridge M. (1998). A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems, 1,* 7-38.

[24] Jennings, N. R., Corera J., Laresgoiti I., Mamdani E. H., Perriolat F., Skarek P. & Varga L. Z. (1996). Using ARCHON to develop real-world DAI applications for electricity transportation management and particle Acceleration control. *IEEE Expert, vol. 11(6)*, 60–88.

[25] Korf, R. E. (1992). A Simple Solution to Pursuit Games. *Proceedings of the 11th Internetional Workshop on Distributed Artificial Intelligence.*

[26] LaMothe, A. (1995). Building Brains into Your Games. *Game Developer Magazine (August/September)*, 26-34

[27] LaMothe, A. (1999). *Tricks of the Windows Game Programming Gurus.* Sams, Macmillan Computer Publishing.

[28] Liu, J-S & Sycara, K. P. (1997). Multiagent Coordination in Tightly Coupled Task Scheduling. *Readings in Agents edited by Michael N. Huhns and Munindar P. Singh*, 164-171

[29] Ljunberg, M. & Lucas, A. (1992). The OASIS air traffic management system. *Proceedings of the Second Pacific Rim International Conference on AI (PRICAI-92)*, Seoul, Korea.

[30] Machado, A., Almeida, A., Ramalho, G., Zucker, J-D. & Drogoul, A. (2002). Multi-Agent Movement Coordination in Patrolling. In: Workshop on Agents in Computer Games. *Proceedings of The 3rd International Conference on Computers and Games (CG'02)*.

[31] Machado, A., Ramalho, G., Zucker, J-D. & Drogoul, A. (2002). Multi-Agent Patrolling: an Empirical Analysis of Alternative Architectures. *Third International Workshop on Multi-Agent Based Simulation (MABS'02). Proceedings of First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*.

[32] Madeira, Charles A. G. (2000). *Projeto do jogo Canyon*. UFPE.

[33] Madeira, Charles A. G. (2001). *FORGE V8: Um framework para o desenvolvimento de josgo de computador e aplicações multimídia.* Dissertação de mestrado (Mestrado em Ciência da Computação)– Universidade Federal de Pernambuco, 20 de julho de 2001.

[34] Manber, U. Introduction to Algorithms: A Creative Approach, 212-214

[35] Minar, N., Kramer, K. H. & Maes P. (1998). Cooperatin Mobile Agents for Mapping Networks. *Proceedings of the First Hungarian National Conference on Agent Based Computing.*

[36] Musse, S. R. & Thalmann, D. (2000). From One Virtual Actor to Virtual Crowds: Requirements and Constraints. *Autonomous Agents´2000,* Barcelona, Spain.

[37] Musse, S. R., Garat, F. & Thalmann, D (1999). Guiding and Interacting with Virtual Crowds. *Proceedings of Workshop Eurographics Computer Animation and Simlation*, Milan, Italy.

[38] N. R. Jennings & M. J. Wooldridge (1998). Applications of Intelligent Agents. *Agent Technology: Foundations, Applications, and Markets (eds. N. R. Jennings and M. Wooldridge)*, 3-28

[39] N. R. Jennings (2001). An agent-based approach for building complex software systems. *Comms. of the ACM, 44 (4)*, 35-41.

[40] Parunak, H. (1987). Van Dyke: Manufacturing experience with the contract net. *In M. Huhns, editor, Distributed Artificial Intelligence. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA*, 285–310.

[41] Pottinger, D. C. (1999). Coordinated Unit Movement. *Game Developer Magazine (January),* 42-51.

[42] Pottinger, D. C. (1999). Implementing Coordinated Unit Movement. *Game Developer Magazine (February),* 48-58.

[43] *Rational. UML Notation Guide* (1997). 1.1 edition. Disponível em: http://www.rational.com/uml

[44] Reynolds, C.W. (1999). Steering Behaviors for Autonomous Characters. *Presented at Game Developers Conference*. Disponível em: http://www.red3d.com/cwr/steer/

[45] *RoboCup Home Page.* Disponível em: http://www.robocup.org

[46] *RoboCup Rescue Home Page:* http://www.r.cs.kobe-u.ac.jp/robocup-rescue/

[47] Russell, Stuart J. & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

[48] Smith R. & Davis R.(1981) Frameworks for Cooperation in Distributed Problem Solving. *IEEE Transactions on Systems, MAN and Cybernetics*, Bol. SMC-11, No. 1.

[49] Stout, B. W. (1996). Smart Moves: Intelligent Path-Finding. *Game Developer Magazine (October/ November),* 28-35

[50] Stout, B. W. (1998). Adding Planning Capabilities to Your Game AI. *Game Developer Magazine (January)*, 39-45

[51] Sukthankar, G. & Sycara K. (2000). Team-aware Robotic Demining Agents for Military Simulation. *Robotics Institute - Carnegie Mellon University*. Disponível em: http://www-2.cs.cmu.edu/~softagents/iaai00/iaai00.html

[52] White, J. E. (1996). Telescript Tecnology: Mobile agents. *Jeffrey Bradshaw, editor Software Agents*. AAAI Press/MIT Press.

[53] Woodcock, Steven (2000). Game AI: The State of the Industry. *Game Developer Magazine (November)*.

[54] Wooldridge, M. & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, *vol. 10(2),* 115–152.