Resumo

Este trabalho apresenta análises de comportamentos sociais, *i.e.* egoísmo e altruísmo, que foram implementados em um ambiente oferecido pela *Microsoft* para a competição *Imagine Cup* na categoria *Project Hoshimi*. Nessa categoria, é necessário escrever linhas de código que representam o ambiente e a estratégia de uma equipe de indivíduos (*bots*) a fim de completar diferentes tipos de missões. O objetivo maior do *Project Hoshimi* é conseguir o maior número de pontos, realizando ou não todos os objetivos que compõem uma missão, em um determinado número de turnos (aproximadamente 5 min.). Utilizando técnicas de inteligência artificial, este trabalho mostra como o sistema como um todo opera através do comportamento dos seus indivíduos, ou seja, os indivíduos do sistema podem tomar atitudes que sejam em prol do grupo ou em prol de si mesmo e o próprio sistema verificará quais destas atitudes maximizarão a sua função objetivo.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

(Dispostos por ordem de aparição no texto)
IC - *Imagine Cup*
PH – *Project Hoshimi*
DLL – *Dynamic-link Library*
SDK – *Software Development Kit*
VS – *Versus*
SA - *Simulated Annealing*

Agradecimentos

Agradeço à Priscila, minha namorada, que entendeu as vezes que deixei de vê-la para me dedicar a este trabalho; ao amigo do trabalho e da faculdade, Thiago Fragoso, que discutiu diversas vezes comigo e deu sua opinião no que poderia melhorar este trabalho; à minha família que compreendia o momento importante e proporcionava um silêncio maior que o habitual.

Agradeço ao meu professor orientador, Fernando Buarque, que sempre acreditou no meu potencial e esteve sempre disponível para ouvir minhas dificuldades. Suas broncas eram encaradas como estímulos e assim me mantive motivado para a realização deste trabalho.

Agradeço também em especial a um aluno de mestrado do meu professor Fernando Buarque, Marcelo Pita, que me ajudou muito a pensar nas abordagens que foram atacadas para este trabalho e esteve sempre disponível para discutir o que estava sendo feito nele.

Agradeço aos professores Tiago Massoni e Cristine Gusmão, que ajudaram a minha equipe na competição Imagine Cup 2007, a qual fomos 6º colocado na fase nacional. E novamente a Fernando Buarque por ser o mentor da nossa equipe e colaborar até em finais de semana.

Capítulo 1

Introdução

Este trabalho apresenta a implementação computacional e a análise dos resultados de comportamentos sociais produzidos por seres virtuais habitantes de um mundo cibernético (*i.e.* ambiente virtual) oferecido pela *Microsoft* para a competição da *Imagine Cup*[1] na categoria *Project Hoshimi[2]*. O *Project Hoshimi* é uma categoria de batalha de programação na qual seus participantes devem codificar uma estratégia para situações e mapas distintos, nos quais o objetivo é a obtenção do maior número de pontos em um determinado número de turnos (*i.e*, a medida de tempo no ambiente do *Projeto Hoshimi*).

A codificação deve ser feita em alguma linguagem suportada pela plataforma .NET[3], *e.g.* C# (lê-se C Sharp). Isto para que após a sua compilação seja gerado um *assembly* .NET**Erro! Fonte de referência não encontrada.**, mais precisamente um arquivo DLL (*Dynamic-link Library*), a fim de que este possa ser carregado pela interface do *Project Hoshimi*. Após a geração e testes do *assembly* .NET, esse arquivo pode ser enviado à competição para ser avaliado junto com outras DLL's de outros implementadores.

A dificuldade de antever todas as possíveis situações e mapas nos quais o *assembly* pode ser carregado faz com que seja importante a utilização de técnicas de inteligência artificial para a codificação da estratégia a ser utilizada. Numa perspectiva cibernética, é fundamental que os seres implementados percebam o ambiente e tomem decisões que melhorem de alguma forma o desempenho global do sistema.

Objetivando essa melhoria de desempenho global, incorporamos e avaliamos alguns comportamentos inspirados em seres sociais, tais como comportamentos egoístas e altruístas.

1.1 Motivação

A principal motivação deste trabalho foi propor uma alternativa computacional que pudesse agregar mais desempenho ao assembly .NET para submissão à competição Imagine Cup. Imaginamos então que a combinação de uso de Inteligência Artificial (IA) e inspiração de comportamentos coletivos fossem diferenciais favoráveis ao problema posto de busca e de otimização.

Adicionalmente, nos motivou também a possibilidade de utilização no mundo real das observações produzidas ao se avaliar o impacto social emergente das interações entre agentes com comportamentos e ethos sociais distintos.

1.2 Objetivo

O objetivo deste trabalho é implementar e analisar os resultados de um *assembly* .NET que codifica comportamentos sociais, mais especificamente egoísmo e altruísmo.

Adicionalmente, é importante notar que o sistema construído deve ser capaz de aprender com as decisões próprias tomadas anteriormente e que ele vai sempre focar em melhorar seu desempenho, ou seja, ganhar mais pontos em um determinado número de turnos.

1.3 Estrutura do Documento

1.3.1 Capítulo 2: Fundamentação Teórica

Nesse capítulo descrevemos em detalhe a categoria *Project Hoshimi*, além disso incluímos conceitos julgados relevantes sobre agentes inteligentes, algoritmos de otimização e comportamentos sociais, todos utilizados no desenvolvimento do projeto.

1.3.2 Capítulo 3: Estratégias e Codificações

Nesse capítulo apresentamos os como e por que de cada parte da estratégia de codificação. Além disso apresentamos as funções incorporadas sua relação com o objetivo a ser alcançado na competição.

1.3.3 Capítulo 4: Experimentos e Resultados

Nesse capítulo descrevemos como foram realizados os experimentos e quais os resultados obtidos. Para fins de comparação, mostramos que os experimentos foram realizados com equipes de indivíduos diferentes em dois mapas distintos, todos com objetivos diversos.

1.3.4 Capítulo 5: Conclusões e Trabalhos Futuros

Nesse capítulo apresentamos as conclusões obtidas através dos resultados dos experimentos, incluímos algumas reflexões e uma lista de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Antes de partirmos para a apresentação da implementação, é necessário um embasamento teórico sobre diversos pontos importantes utilizados neste trabalho.

Primeiramente vamos descrever e explicar os principais pontos sobre o *Project Hoshimi*, desde o histórico da categoria até os critérios que definem quem é o vencedor. Explicaremos depois a abordagem de inteligência artificial baseada em agentes inteligentes que foi utilizada para a codificação do *assembly*.

Outros pontos a serem abordados aqui, são alguns algoritmos inteligentes de busca e otimização utilizados para resolver o problema do caixeiro viajante[5] e busca de do melhor caminho a ser percorrido entre dois pontos diferentes.

2.1 Project Hoshimi

2.1.1 Contextualização

O *Project Hoshimi[2]* é uma categoria de batalha de programação de uma competição organizada pela *Microsoft*, a *Imagine Cup[1]*, na qual os participantes devem codificar suas estratégias na linguagem .NET envolvendo uma equipe de vários indivíduos (*bots*) com o objetivo de conseguir o maior número de pontos.

O *Project Hoshimi* disponibiliza um SDK (*Software Development Kit*) para que o participante possa criar suas estratégias e testá-las através de um visualizador 2D ou 3D[6]. A cada ano um novo tema é proposto para os participantes da competição. Por exemplo de 2005 a 2007, os *assemblies* se baseavam num SDK com um *background* bastante interessante: curar partes do corpo humano, realizando os objetivos e desafios propostos pelo mapa. Nesse ambiente os *bots* (ou *nanobots* como eram chamados) navegavam pelos mapas através de correntes sanguíneas combatendo os inimigos (chamados de *neurocontrollers*). A Figura 2-1 e a Figura 2-2 mostram um *assembly* desenvolvido utilizando o SDK do *Project Hoshimi* 2007, e colocado para ser testado nos visualizadores 3D e 2D respectivamente.

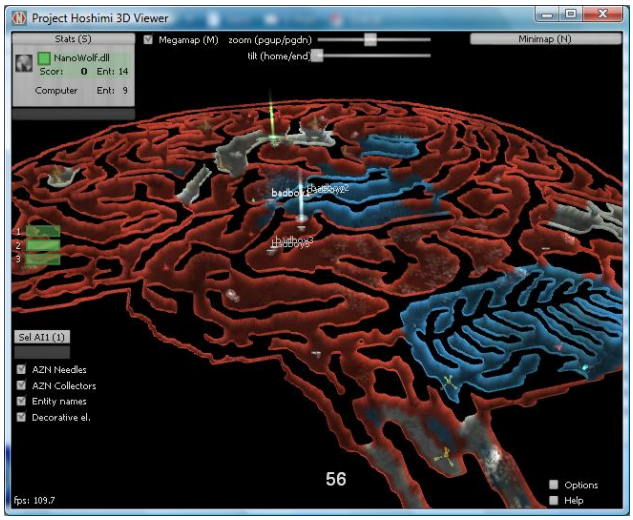


Figura 2-1: Visualizador 3D do Project Hoshimi 2007.

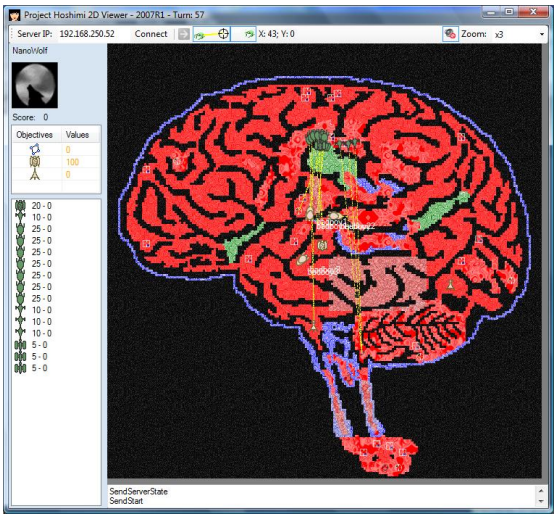


Figura 2-2: Visualizador 2D do Project Hoshimi 2007.

É importante mencionar que o *software* do *Project Hoshimi* carrega até dois *assemblies* para um mesmo jogo, e é assim que se realiza a competição nas fases mais adiantes: Jogador *VS.* Jogador.

2.1.2 O Tema do *Project Hoshimi* 2008

O SDK utilizado para a implementação deste trabalho foi o do *Project Hoshimi* 2008. Assim sendo, é importante descrever as diferenças de versões anteriores e as novas funcionalidades.

A *Imagine Cup* é uma competição que sempre se baseia em temas sociais, *e.g.* educação, e a categoria *Project Hoshimi* era a única que não se enquadrava nos temas propostos, então para 2008 também o tema do jogo foi alterado. O tema da *Imagine Cup* 2008 é o meio ambiente: "Imagine um mundo onde a tecnologia permita um meio ambiente sustentável".

O tema do jogo então não é mais para curar partes do corpo humano, com seus *bots* navegando por vasos sanguíneos, mas pode ser entendido como "curar" países da poluição de fábricas existentes neles. Pela história não ser mais dentro do corpo humano, muitas nomenclaturas foram alteradas no SDK, mas o motor do jogo continua quase o mesmo.

Uma funcionalidade nova no *Project Hoshimi* é o conceito de *Shields* (escudos) e *SafeAreas* (Áreas seguras). Alguns *bots* podem criar escudos em sua volta e assim criar áreas seguras, ou seja, se algum outro *bot* estiver nesta área, ele não será atingido por eventuais ataques, mas o escudo irá perdendo a sua força até que não exista mais escudo algum.

Outro detalhe muito importante do novo *Project Hoshimi*, é que nossos *bots*, ao atacarem (atirarem), podem atingir outros *bots* da mesma equipe se estes estiverem na área de efeito do ataque realizado. Nas edições anteriores, isso não era possível; os ataques somente atingiam o inimigo. Eis aqui uma preocupação com as implicações (sócias) das ações dos agentes.

2.1.3 Detalhamento do SDK do *Project Hoshimi* 2008

Nesta seção, vamos detalhar os pontos julgados importantes para o entendimento do que foi implementado e analisado.

2.1.3.1 Ponto de Aterrissagem (*Landing Point*)

O ponto de aterrissagem é o ponto de onde todos os *bots* da equipe irão iniciar o jogo. A primeira estratégia a ser desenvolvida é para decidir onde colocar o ponto de aterrissagem. Não podemos colocá-lo em áreas azuis (oceano), pois não é um ponto válido. Caso seja colocado em alguma área azul, o motor do jogo escolherá um ponto de aterrissagem para a sua equipe.

2.1.3.2 Terreno (*Terrain*)

Os bots aterrissam em um terreno, o qual contém as diferentes áreas:

- Água: não é permitido andar pela água.
- Área de velocidade normal: os *bots* andam em uma velocidade normal.
- Área de velocidade baixa: os *bots* andam em uma velocidade baixa.
- Área de velocidade muito baixa: os *bots* andam em uma velocidade muito baixa.
- Córregos de ar: existe muito ar nessa área. Se os *bots* andarem na mesma direção do córrego, eles andarão mais rápido, caso andarem na direção oposta, andarão mais devagar.

Por exemplo, os *Explorers* (um tipo de *bot* do mundo virtual) não são afetados pelos diferentes tipos de áreas, mas são afetados pelos córregos de ar. A Figura 2-3 mostra um exemplo de um mapa.

Figura 2-3: Exemplo de mapa do Project Hoshimi.

Além dos diferentes tipos de área, o terreno também contém:

- Ponto de *Oxy*: pontos onde se podem coletar moléculas *Oxy*.
- Ponto de *Hoshimi*: pontos onde se deve construir *needles* (outro tipo de *bot*) para a injeção de moléculas *Oxy* na terra.
- Ponto de Objetivo: pontos que devem ser percorridos para completar objetivos de navegação ou navegação única. Os objetivos vão ser explicados em uma seção mais a frente.

Coletar moléculas *Oxy* (algo análogo a oxigênio) e injetá-las na terra, é metáfora para o combate a poluição.

2.1.3.3 A Equipe

Para que possamos montar nossa estratégia, o *Project Hoshimi* oferece alguns tipos de *bots* que podem compor a nossa equipe, pelas regras, podemos ter no total de 40 *bots* no jogo. A seguir descrevemos os principais, que foram utilizados na implementação deste trabalho:

- **AI**: é o *bot* principal do jogo (não confundir com a sigla de inteligência artificial). Ele é que constrói os outros *bots* da nossa equipe. Ao começar o jogo, ele é colocado no ponto de aterrissagem e a partir daí é iniciada a implementação de nossa estratégia com a criação de bots e conseqüentes ações. O AI é o único *bot* que pode criar *needles* e *blockers* e estes são criados onde o AI está posicionado. Os outros *bots* criados aparecem no ponto de aterrissagem escolhido. A Figura 2-4 e a Figura 2-5 ilustram este *bot* em uma visualização 3D e 2D, respectivamente.

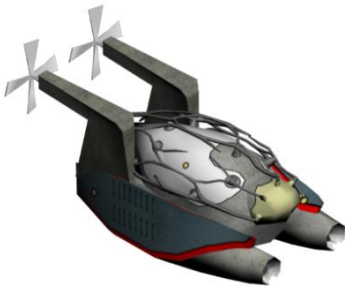


Figura 2-4: AI em uma visualização 3D.

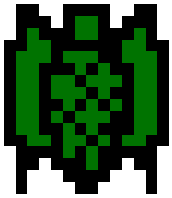


Figura 2-5: AI em uma visualização 2D.

- ***Collector***: esses *bots* podem coletar moléculas *Oxy* de pontos *Oxy* e transferir para os *needles*. Além disso, os *collectors* podem atacar os inimigos. A Figura 2-6 mostra uma visualização do *collector* no ambiente 2D.

Figura 2-6: Collector em uma visualização 2D.

- ***Container***: esses *bots* se parecem com o *collector*, mas eles carregam mais *Oxy* e não tem a capacidade de se defender (atacar). A Figura 2-7 e a Figura 2-8 mostram visualizações 3D e 2D do *container*.

Figura 2-7: Container em uma visualização 3D.

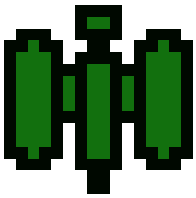


Figura 2-8: Container em uma visualização 2D.

- ***Explorer***: esses *bots* se movem mais rápido que os outros, além de ter um campo de visão maior que o dos outros *bots*. São indicados para objetivos de navegação, patrulha e exploração do mapa. A Figura 2-9 e a Figura 2-10 mostram visualizações 3D e 2D do *explorer*.

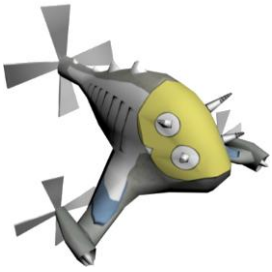


Figura 2-9: Explorer em uma visualização 3D.

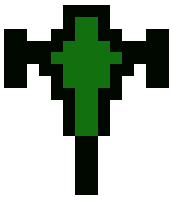


Figura 2-10: Explorer em uma visualização 2D.

- ***Needle***: esses bots injetam moléculas de *Oxy* na terra para combater a poluição. Eles devem ser criados em pontos *Hoshimi* para que consiga armazenar moléculas *Oxy* e conseqüentemente ganhar mais pontos. A Figura 2-11 mostra o *needle* em uma visualização 2D.

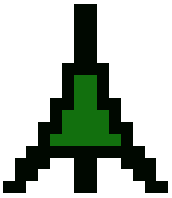


Figura 2-11: Needle em uma visualização 2D.

- ***LPCreator***: esse *bot* é o único que pode criar outro ponto de aterrissagem onde novos bots construídos possam aparecer. Só é permitido criar um *LPCreator* por jogo e este só vive durante os próximos 500 turnos. Durante esse tempo, o *LPCreator* pode fechar um ponto de aterrissagem e criar outro em qualquer ponto do mapa (exceto na água). A Figura 2-12 mostra o *LPCreator* em uma visualização 3D.

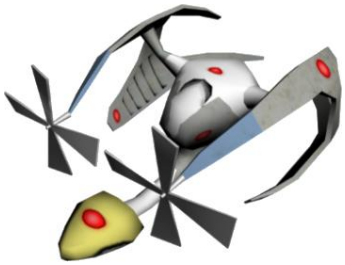


Figura 2-12: LPCreator em uma visualização 3D.

2.1.3.4 Características gerais dos *Bots*

Cada tipo de bot tem suas próprias características. A Tabela 2-1 mostra tais características e descrições.

Tabela 2-1: Características dos bots.

Característica	Descrição
ContainerCapacity	Máximo de gás *Oxy* que o *bot* pode carregar
CollectTransferSpeed	Quantidade de gás que é coletado de um ponto de *Oxy*, ou transferido para um *needle* durante um turno.
Scan	Distância da visão dos *bots*. Isso detecta fábricas ou *bots* de outra equipe.
MaxDamage	Quando o *bot* está atacando, é o máximo de dano que pode ser tirado por turno
DefenseDistance	O *bot* só pode atacar em um ponto menor ou igual a essa distância
Constitution	É a quantidade de vida do *bot* no início do jogo. Se essa quantidade zerar, o *bot* morre.
Shield	Quando um bot tem essa característica positiva, todos os bots da mesma equipe que estiverem perto não vão sofrer ataques. Mas o escudo irá perder sua força. Somente o AI e o *Needle* que podem ter *Shield*

2.1.3.5 Objetivos gerais das missões

Uma missão é um conjunto de objetivos que devem ser cumpridos. Dependendo do objetivo, pode-se ganhar algum bônus ao completar tal objetivo e fazer a pontuação aumentar. Na Tabela 2-2 abaixo descrevemos todos os tipos de objetivos.

Tabela 2-2: Objetivos do *Project Hoshimi*.

Objetivo	Descrição
AIAlive	O *bot* AI deve estar vivo em um determinado turno.
NavigationObjective	Algum *bot* (especificado ou não pelo objetivo) deve percorrer pontos no terreno.
UniqueNavigationObjective	O mesmo que *NavigationObjective*, a diferença é que todos os pontos devem ser percorridos por um único *bot*.
FactoryObjective	A destruição de todas as fábricas.
ScoreObjective	Alcançar uma determinada pontuação em um determinado número de turnos.

Em um jogo do tipo Jogador VS. Jogador, o vencedor é aquele que tiver o maior número de pontos. Não é necessário completar todos os objetivos, mas vale lembrar que realizar objetivos aumenta a pontuação.

2.2 Agentes Inteligentes

2.2.1 Definição

Um agente é qualquer entidade que possa perceber seu ambiente através de sensores e atuar sobre ele por meio de atuadores[7]. Seguindo o exemplo contido no livro de Russel[7], um agente humano tem olhos, nariz e ouvidos como sensores para perceber o ambiente; as mãos, pés e boca dos humanos podem ser entendidos como atuadores. Em um agente robótico, câmeras são sensores e motores são atuadores.

Outra definição de agente o considera como sendo uma entidade com uma capacidade encapsulada de resolução de problemas [10]. Assim, o agente conteria as seguintes propriedades:

- **Autonomia**: execução de maior parte de suas ações sem interferência direta de outros agentes (*e.g.* humanos ou computacionais), possuindo controle total sobre suas ações e estado interno. Nesse caso, para ter autonomia, o agente deve ter um certo grau de inteligência;
- **Habilidade Social**: por impossibilidade de resolução de certos problemas ou por outro tipo de conveniência, interagem com outros agentes (humanos ou computacionais), para completarem a resolução de seus problemas, ou ainda para auxiliarem outros agentes. Disto surge a necessidade de que os agentes tenham capacidade para comunicar seus requisitos aos outros e um mecanismo decisório interno que defina quando e quais interações são apropriadas;
- **Capacidade de Reação:** percebem e reagem às alterações no ambientes em que estiverem inseridos;
- **Capacidade Pró-Ativa**: agentes, do tipo deliberativo, além de atuar em resposta às alterações ocorridas em seu ambiente, apresentam um comportamento orientado a objetivos, tomando iniciativas quando julgarem apropriado.

Utilizando-se o *Project Hoshimi* como exemplo, temos todos os *bots* são agentes do sistema (jogo), pois eles podem perceber o ambiente e agir sobre ele realizando-objetivando alguma ação-resultado.

2.2.2 Características dos Ambientes

Os ambientes existentes de sistemas de agentes inteligentes podem ser variados:

- **Acessível x Inacessível**: se o sensor do agente permite acesso de um estado completo do ambiente, temos um ambiente acessível; caso contrário, temos um ambiente inacessível. Um ambiente é efetivamente acessível se os sensores do agente conseguirem detectar todos os fatores relevantes para uma tomada de decisão;
- **Determinístico x Não Determinístico**: o ambiente é classificado como determinístico se conseguirmos determinar o próximo estado através do estado atual e das ações escolhidas pelo agente;
- **Estático x Dinâmico**: dizemos que um ambiente é dinâmico se este mudar enquanto o agente está pensando na ação que irá escolher. Caso o ambiente não mude, então ele é estático;
- **Discreto x Contínuo**: o ambiente é discreto se houver um número finito de ações e percepções;

Partindo dessas características e conhecendo o ambiente do *Project Hoshimi*, podemos classificá-lo como: Semi-acessível, Não Determinístico, Dinâmico e Discreto.

2.2.3 Sistemas Multi-Agentes (SMA)

Definimos Sistemas Multi-Agentes como sistemas em que dois ou mais agentes interagem ou trabalham em conjunto de forma a desempenhar um determinado conjunto de tarefas. Para que não haja objetivos conflitantes entre os agentes, existe a necessidade ter algum tipo de coordenação do sistema, prevenindo-o da anarquia e do caos. Essa coordenação pode atuar tanto em SMAs cooperativos quanto em competitivos, havendo algum planejamento ou negociação de recursos e objetivos no sistema.

2.2.3.1 Coordenação

Coordenação é o processo pelo qual um agente raciocina sobre suas ações locais e as ações (antecipadas) de outros agentes com o objetivo de garantir que a comunidade funcione de maneira coerente [8].

A coordenação de estratégias que conciliem os interesses individuais de cada agente para que as atividades relacionadas desenvolvam-se de modo coordenado é um dos aspectos fundamentais a serem considerados no projeto de Sistemas Multi-Agentes.

O conceito de coordenação define aspectos gerais de interação entre agentes de forma a viabilizar a coesão entre seus comportamentos e ações em relação aos objetivos globais do sistema. A partir disso, apresenta-se uma taxonomia para os processo de coordenação (vide Figura 2-13) que apresentas duas abordagens principais[9]:

a) Cooperação: é a coordenação entre agentes não-antagônicos[1];

b) Competição: é a coordenação entre agentes antagônicos.

[1] O conceito de não-antagônico define que os interesses dos agentes não são conflitantes.

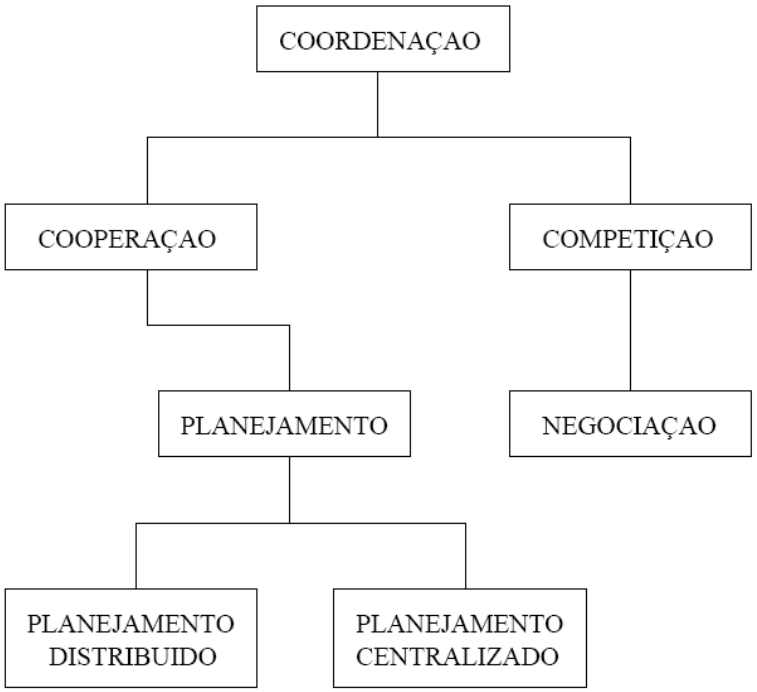


Figura 2-13: Taxonomia da coordenação.

Segundo Lesser e Corkill *apud* [8], os objetivos do processo de coordenação visam garantir que:

- Todas as partes necessárias ao sistema estejam inseridas nas capacidades funcionais de ao menos um agente;
- Os agentes interajam de maneira a permitir que suas atividades sejam desenvolvidas e integradas em uma solução global;
- Os membros da sociedade atuem com propósitos e consistentemente;
- Todos esses objetivos sejam atingíveis dentro das limitações computacionais impostas e dos recursos disponíveis.

A cooperação acontece quando vários agentes planejam e executam suas ações de uma forma coordenada. Alguns objetivos genéricos de cooperação entre agentes são:

- Diminuição do tempo de execução de uma tarefa através do paralelismo;
- Aumento do escopo de tarefas executáveis através do compartilhamento de recursos;
- Maior probabilidade de finalização de uma tarefa em função de sua dupla incumbência, a ser realizada possivelmente através de distintos métodos de execução; e
- Diminuição da interferência entre tarefas evitando interações prejudiciais.

A competição acontece quando vários agentes se preocupam com os seus objetivos somente e isso pode causar conflitos de interesses que pode ser resolvido por meio de negociações. O processo de negociação atua sobre o melhoramento de concordância acerca de pontos de vista em comuns ou planos através de compartilhamento de estruturas de informações relevantes, ocorrendo entre agentes com objetivos diferentes no qual decisão conjunta é alcançada por dois ou mais agentes, cada um tentando alcançar seus objetivos pessoais.

2.3 Algoritmos selecionados

2.3.1 Simulated Annealing

O *Simulated Annealing (SA)* é um algoritmo inteligente iterativo utilizado para resolver problemas de otimização global, que foi proposto originalmente por KIRKPATRICK et al (1983). Este algoritmo se baseia em uma analogia com a termodinâmica ao simular o resfriamento lento da matéria após ser aquecida. Simulated Annealing pertence à mesma classe dos métodos das Redes Neurais e Algoritmos Genéticos, no sentido que simulam métodos Naturais.

Este método de otimização faz uma analogia com o processo de recozimento (*annealing*) da metalurgia. Sabe-se da metalurgia que, se o metal é resfriado em condições apropriadas, uma estrutura cristalina melhor pode ser obtida (Kirkpatrick, 1983). No recozimento o metal é aquecido a altas temperaturas, causando um choque violento nos átomos. Se o metal for resfriado de forma brusca, a microestrutura tende a um estado randomicamente instável, porém, se o metal é resfriado de forma suficientemente lenta, o sistema procurará um ponto de equilíbrio caracterizado por uma microestrutura ordenada e estável.

A analogia com o processo termodinâmico é bastante direta. A função objetivo substitui a energia, os diversos estados da matéria seriam os ótimos locais, e a estrutura cristalina sendo o ótimo global. O parâmetro mais importante, e que regula o processo do SA, é a chamada temperatura. O valor da temperatura inicial e o fator de decrescimento dessa temperatura (resfriamento) são fatores muito importantes para um bom desempenho do *Simulated Annealing*. Assim as variáveis de um projeto são perturbadas randomicamente e armazena-se o melhor valor da função objetivo a cada perturbação. A temperatura é então reduzida (*annealing*) e novas tentativas são executadas. Este procedimento continua até escaparmos de um mínimo local. Ao final do processo é possível que se obtenha um mínimo global.

Metropolis et al (1953)[17] introduziram um método numérico simples, extensão de *Monte Carlo* [18] que representa o estado de equilíbrio de um conjunto de átomos a uma dada temperatura. Seja ΔE a energia de um sistema de átomos a uma temperatura T. Em cada passo do algoritmo, é dado um deslocamento aleatório a um átomo, o que implica uma nova energia do sistema ΔE. Se esta nova energia ΔE é menor ou igual a zero ($\Delta E \leq 0$), o deslocamento é aceito, caso contrário ($\Delta E > 0$), a probabilidade da configuração ser aceita será dada pela equação:

$$P(\Delta) = \exp\left(-\frac{\Delta E}{T}\right) \qquad \text{(Equação 2-1)}$$

Um número randômico r, uniformemente distribuído, deve ser gerado no intervalo [0,1]. Se $r \leq P(\Delta E)$ a nova configuração é aceita. Se $r > P(\Delta E)$ a configuração anterior é utilizada para iniciar um novo passo. Os parâmetros do algoritmo são: a função custo, que representa a energia do sistema; as variáveis de projeto, que descrevem sua configuração e a temperatura, que é um parâmetro de controle.

Exemplo de algoritmo:

```
1. s0 = permutação de N pontos (e.g., 3942067518, para N = 10)
```

```
2. t = algum número grande, maior que a maior diferença de custo
   possível
3. Repetir:
      1. Selecionar s = N(s0), onde N é a função de geração de
         vizinhança.
      2. custo = [ d(i, i + 1) + d(j, j + 1) ] - [ d(i, j) + d(i + 1,
         j + 1) ]
      3. Se custo < 0 então s0 = s
      4. Senão:
            1. Gera um número aleatório x entre 0 e 1
            2. Se x < exp (-custo / t) então s0 = s
      5. t = 0.9 * t
      6. Verificar condição de parada:
            1. Pelo número de iterações (se maior que um limiar,
               pára).
            2. Pelo custo (se menor que um limiar, pára).
4. s0 é a solução
```

O *Simulated Annealing* foi utilizado neste trabalho para resolver o problema do caixeiro viajante[5] que vamos descrever mais adiante.

2.3.2 Pathfinding: algoritmo A*

O *Pathfinding* (procura pelo caminho) é um dos problemas mais populares em jogos que utilizam inteligência artificial. Esse termo é utilizado em aplicações computacionais quando se quer descobrir o melhor caminho de um ponto A até um ponto B. O A* (a estrela) é a escolha mais popular para *pathfinding* por ser razoavelmente flexível e pode ser utilizado em uma larga escala de contexto[12].

O A* é como outros algoritmos de busca por grafo que pode procurar em uma área enorme do mapa. Parece-se com o algoritmo de Dijkstra[12] para encontrar o caminho mais curto, e com o algoritmo BFS[12] (*Best-First Search*) que utiliza uma heurística para se guiar. Esse algoritmo BFS se originou do método de busca *Depth-First Search*, no qual um grafo de adjacências é percorrido recursivamente selecionando apenas um de seus vizinhos. No BFS, o funcionamento é o mesmo do seu antecessor, mas com uma melhoria: não se seleciona apenas um vizinho adjacente, mas sim o vizinho adjacente que possui a maior chance de indicar o melhor caminho. O segredo do sucesso é a combinação desses dois algoritmos citados acima. A figura abaixo mostra situações em que o A* mostra suas características:

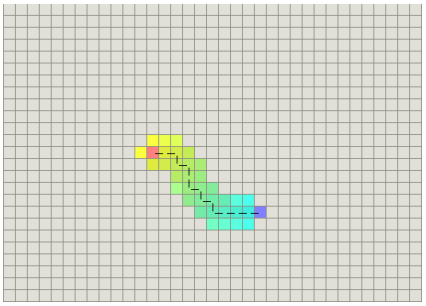


Figura 2-14: A* com propriedade do Dijkstra.

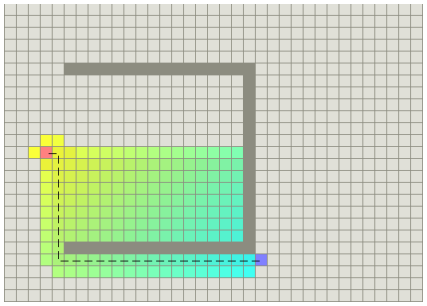


Figura 2-15: A* com propriedade do BFS.

Na terminologia padrão utilizada ao falar do A*, o *g(n)* representa o custo do caminho do ponto inicial para qualquer vértice *n*, e *h(n)* representa o custo estimado do movimento para mover do vértice *n* até o destino final, podemos chamar o *h(n)* de heurística. A cada iteração, é verificado o vértice que tem o menor *f(n)* = *g(n)* + *h(n)*.

2.3.3 Heurísticas

A função de heurística mostra ao A* uma estimativa de um custo mínimo de qualquer vértice *n* até o destino final. Por isso é importante a escolha de uma boa função de heurística.

2.3.3.1 Distância de Manhattan

A heurística padrão em A* é a utilização da distância de *Manhattan*. Essa distância se caracteriza por poder percorrer somente na vertical ou horizontal. Há um custo *D* para cada movimento vertical ou horizontal. Este custo depende do ambiente, *e.g.* tipos diferentes de terreno, e deve ser calculado por alguma função de custo. A **Figura 2-16** mostra o algoritmo do A* buscando o melhor caminho. Já equação abaixo, mostra a heurística utilizando a distância de *Manhattan*.

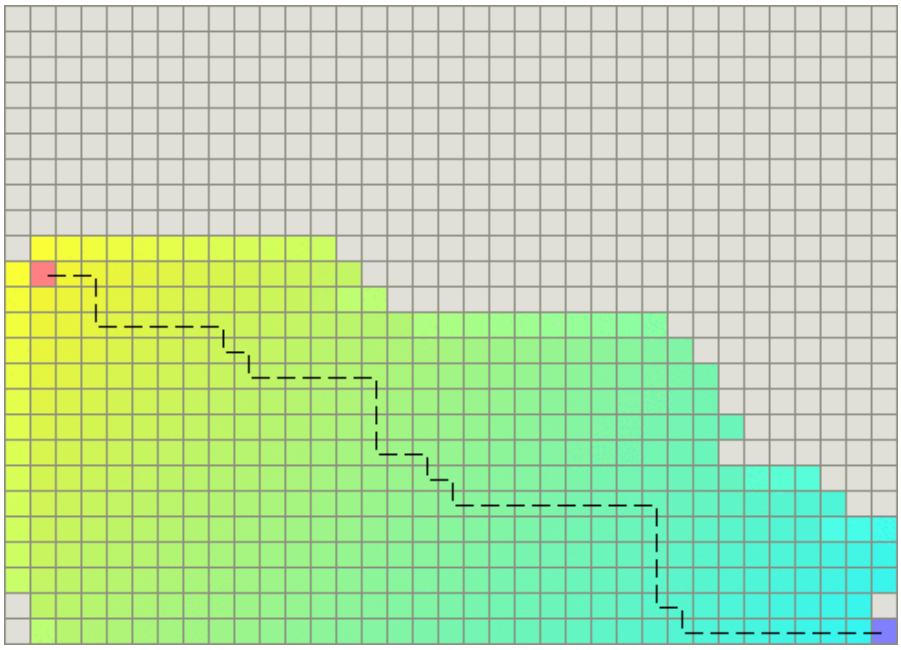


Figura 2-16: A* utilizando a distância de Manhattan como heurística.

$$h(n) = D + (abs(n.x - goal.x) + abs(n.y - goal.y)) \quad \text{(Equação 2-2)}$$

2.3.3.2 Distância Euclidiana

A heurística da distância euclidiana deve ser utilizada caso os agentes possam se mover em qualquer direção, não só vertical ou horizontal. Caso seja permitido percorrer casas na diagonal, pode-se utilizar esta heurística. Apresentamos abaixo como ficaria o caminho utilizando-se a distância euclidiana e também sua equação:

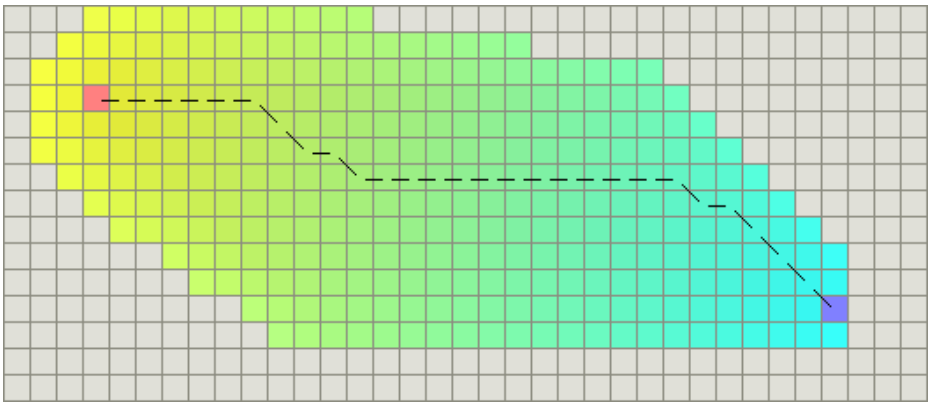


Figura 2-17: A* utilizando a distância euclidiana como heurística.

$$h(n) = D + \left(\sqrt{(n.x - goal.x)^2 + (n.y - goal.y)^2}\right) \quad \text{(Equação 2-3)}$$

2.3.3.3 Breaking Ties

Na Figura 2-16, da seção 2.3.3.1, é mostrado como o algoritmo do A* se comporta dados o ponto inicial e final que se quer percorrer. Podemos notar que há muitas áreas que são percorridas pelo laço, mas não são utilizadas de nada; são percorridas (adicionadas a lista aberta, ver algoritmo na seção 2.3.4) pois tem o mesmo menor *f* encontrado. A fim de resolver este problema, podemos adicionar quebras de laço (*breaking ties*) na heurística[12].

Na Figura 2-18 podemos ver um exemplo de um A* utilizando uma heurística com a distância de *Manhattan* adicionada de *breaking ties*. É notório a diferença de pontos que não foram percorridos como possíveis pontos de passagem, como era anteriormente.

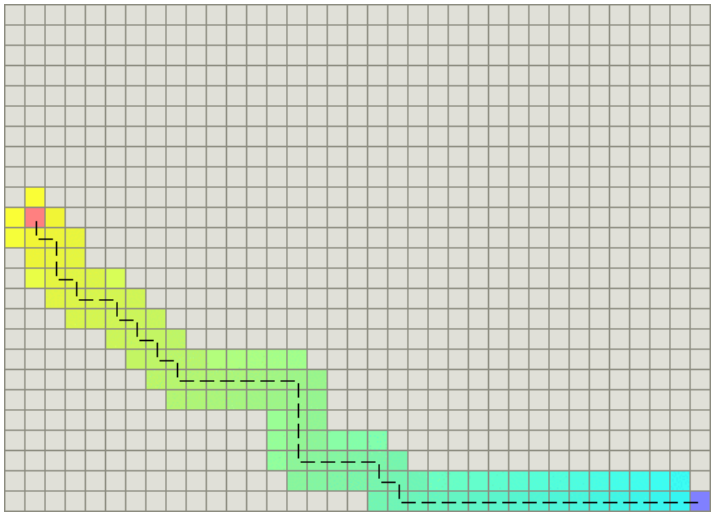


Figura 2-18: Heurística de Manhattan com breaking ties.

2.3.4 Implementação do algoritmo A*

```
funçao AStarSearch(ini,fim) {
      enfileirar(ini);
      enquanto (lista aberta não estiver vazia)
      {
            n = desenfileirar();
            se (n == fim)
            {
                  Construacaminho();
                  retorne caminho encontrado;
            }
            enquanto(n.temmaisfilhos())
                  filho = n.proximofilho();
                  novog = n.g + filho.custo;
                  se ((contem(filho) ou lista_fechada_contem(filho)) e filho.g
            <= novog)
                        continue;
                  Fim se
                  filho.pai = n;
                  filho.g = novog;
                  filho.h = Estimativa(filho);
                  filho.f = filho.g + filho.h;
                  se(lista_fechada_contem (filho))
                        remove_lista_fechada(filho)
                  Fim se
                  se(lista_nao_contem (filho))
                        enfileirar(filho);
                  senao
                        reenfileirar(filho);
                  Fim se
            Fim enquanto
            Adicionar_lista_fechada(n);
      Fim enquanto
      Retorne caminho impossível;
Fim
```

2.4 Comportamentos Sociais

Nesta seção sobre comportamentos sociais, vamos descrever alguns dos comportamentos sociais que foram implementados e simulados neste trabalho: altruísmo e egoísmo.

2.4.1 Altruísmo

A definição de altruísmo foi criado pelo filósofo francês *Auguste Comte* (1798-1857) para caracterizar um interesse dos seres humanos a dedicar-se aos outros.

> "Viver para os outros, não é somente a lei do dever, mas também a lei da felicidade". *Auguste Comte*.

Essa doutrina é muito discutida até os dias atuais. Muitos pensam que altruísmo é uma falácia, pois acreditam que somos guiados por genes egoístas e assim lutamos sempre pela sobrevivência[14]. Apesar destes pensamentos existirem, um estudo liderado pelo neurocientista brasileiro *Jorge Moll Neto* mostra que atitudes altruístas ativam regiões de prazer do cérebro. Ao fazermos uma boa ação, segundo ele, acionamos no cérebro o sistema de recompensa. O mesmo se acende em situações de prazer, como comer chocolate, fazer sexo, ganhar dinheiro ou consumir drogas[13].

2.4.2 Egoísmo

A definição de egoísmo é: preocupação com os nossos próprios interesses. Esta é uma das maiores características dos seres pensantes, os homens. O livro O Gene Egoísta, de *Richard Dawkins*, apresenta uma teoria evolucionária que procura explicar a evolução das espécies na perspectiva dos genes e não na do indivíduo ou da espécie. Segundo a Teoria do gene egoísta, o gene é a unidade fundamental da evolução.

Existem vários tipos de egoísmos, como individual, social, político, religioso ou econômico. O que tem em comum entre eles é a maneira de pensar, sempre em si, mesmo sendo a longo prazo. Por exemplo, há pessoas que praticam caridade dando um pouco do que têm aos necessitados e com isso não se consideram egoístas. Praticam-se caridade sem esperar recompensa na vida terrena, certamente devem acreditar numa recompensa em outra vida[15].

2.4.3 Cooperação – Egoísmo ou Altruísmo?

Após esta pequena explanação sobre egoísmo e altruísmo podemos nos perguntar: quanto (o que) foi implementado neste trabalho quanto a estes comportamentos sociais?

Por conter vários agentes no *Project Hoshimi* e apenas um objetivo comum, somar mais pontos, sistema considera alguns comportamentos como sendo altruístas e/ou egoístas. Por exemplo:

- Um *container* pode coletar gás *oxy* de um ponto mais perto de onde ele se localiza, mais perto do ponto *hoshimi*, para onde o *container* vai transferir o *oxy*, ou de um ponto onde a soma das distancias de coleta e transferência seja menor. Tendo em vista que há fábricas que podem matar o *container*, ele poderia escolher o ponto mais perto dele para realizar a

coleta, assim teríamos um comportamento egoísta individual. Mas se a melhor escolha para o sistema seria coletar de um ponto onde a soma das distancias de coleta e transferência fosse menor, o *container* poderia escolher essa opção e esta, ser considerada como altruísta por pensar no sistema. Pensamento contrário, ou egoísta, pois se vai ser melhor para o sistema, vai ser para o *container* também;

- Um container pode transferir gás *oxy* para pontos *hoshimi* que já contenham *needles* construídos ou esperar em algum ponto *hoshimi* até que o *needle* seja construído para fazer a transferência. Claro que ir para um ponto onde já tem um *needle* construído é a melhor alternativa para o sistema, mas para o *container* pode não ser por ser arriscado caso o ponto seja muito longe.

Depois de citar alguns exemplos de comportamentos que podem acontecer, vamos a um caso mais clássico: imaginemos quatro *containers* para carregar 50 unidades de gás *oxy* cada um para os *needles* já existentes. Visto que cada *needle* suporta no máximo 100 unidades de gás *oxy*, como deveriam se comportar os *containers* para o preenchimento dos *needles*?

Os *containers* devem ter em "mente" que cooperar para que o sistema ganhe mais pontos vai ser melhor para cada agente também. Nesse caso, os agentes devem se organizar e cooperar sem haver conflito e competição entre eles. Por um lado estão sendo altruístas, mas por outro, egoístas.

Caso os agentes fossem todos tentar transferir para o *needle* mais próximo, o sistema provavelmente iria perder tempo (e deixar de ganhar pontos), pois haveria agente que foi ao local e não conseguiu realizar seu objetivo.

Capítulo 3

Estratégia e Codificações

Nesta seção estão descritos os pontos relevantes da estratégia que foram considerados e codificados neste trabalho, incluímos aqui também informações que possibilitam o entendimento das escolhas realizadas pelos agentes inteligentes quando da operação do sistema proposto.

3.1 Configurações iniciais

Iniciaremos a descrição das contribuições e codificações pelas configurações iniciais escolhidas para o sistema. As configurações iniciais se referem à equipe inicial de *bots*, ou seja, ao iniciar o jogo certa quantidade de *bots* de cada tipo é criada para realizar o objetivo do mapa.

Dado que o mapa pode conter vários objetivos diferentes, pode-se basear neles para inferir quantos *bots* serão necessários criar. Por exemplo, se um mapa possui objetivos de navegação, objetivos de pontuação e objetivo de navegação única (*UniqueNavigationObjective*), pode-se criar vários *explorers* para suprir os objetivos de navegação; em contrapartida, se um mapa possui apenas objetivo de pontuação e de destruição de fábricas, não faz muito sentido a criação de muitos *explorers*, mas sim de *containers* ou *protectors* (*collectors* nomeados de *protectors* somente para objetivo de destruição de fábricas).

Para a realização deste trabalho, foram construídos dois mapas com objetivos de pontuação (*ScoreObjective*) e de destruição de fábricas (*FactoryDestructionObjective*). Como foi explicado na seção 2.1.3.5, o objetivo de pontuação deve ser completado obtendo-se um certo número de pontos em um determinado número de turnos. Pode-se alcançar essa pontuação também ao se completar outros objetivos que possuam bônus associados, ou coletando-se gás *oxy* e transferindo o mesmo para *needles* que é uma ação central no presente trabalho. O objetivo de destruição de fábrica corresponde a destruição de todas as fábricas dentro do tempo máximo de turnos do sistema, *i.e.* 1500 turnos. Este objetivo deve ser completado utilizando *collectors* para atacar as fábricas (o qual referenciamos como *protectors* quando esses estiverem engajados em tal objetivo).

Como as regras permitem um total de até quarenta *bots* como parte da equipe, optou-se por não se criar inicialmente muitos *containers* ou *protectors*, pois existe a possibilidade de haver

situações em que algum *bot* tenha de ser "morto" para a criação de *needles* (objetivando aumento de pontuação). A Tabela 3-1 apresenta a configuração inicial do jogo.

Tabela 3-1: Configuração inicial do jogo.

Bot	Quantidade
AI	01[2]
Container	04
Protector	01
Explorer	01

Inicialmente existe somente um *AI*, pois ele é o *bot* principal do jogo. Optou-se também por quatro *containers*, já que são necessários dois deles para preencher um *needle*, e os *needles* devem ser criados pelo *AI* ao percorrer os pontos *hoshimi* no mapa. Portanto, os quatro *containers* inicialmente criados são mais que suficientes no início do jogo; posteriormente, verifica-se a necessidade de criação de outros.

A seguir, criou-se um *protector* e um *explorer* para o objetivo de destruição de fábricas. Uma unidade de cada desses *bots* é suficiente para iniciar o encaminhamento do objetivo. Da mesma forma, no decorrer do jogo a necessidade de criação de mais *protectors* ou *explorers* será analisada. Na Figura 3-1 está apresentado um mapa e uma configuração inicial do jogo. Nesta figura, podem ser identificados um ponto *hoshimi* representado pelo círculo laranja; uma fábrica representada pelo círculo azul; e um *AI*, representado pelo círculo vermelho.

Figura 3-1: Início do jogo (configuração inicial).

[2] Note-se que o AI é um *bot* tipo único e portanto não poderemos aumetar seu número no jogo.

3.2 Ações codificadas

3.2.1 Ponto de aterrissagem (*Landing point*)

Depois da configuração inicial, o ponto de decisão seguinte é a escolha do ponto de aterrissagem, *i.e.* o ponto aonde todos os *bots* do time irão "nascer" no jogo. Para escolher esse ponto, é necessário se considerar os objetivos e os inimigos que estão no mapa. Em suma, deve-se prestar a máxima atenção nesta escolha a fim de melhorar a pontuação e vantagem futura para com os outros jogadores do *Project Hoshimi*.

A escolha do ponto de aterrissagem realizada neste trabalho focou na rapidez do ganho de pontos, ou seja, consideraram-se locais com maiores concentrações de pontos *hoshimi* do mapa. A heurística adotada foi: dividiu-se o mapa em 16 partes e verificaram-se quantos pontos *hoshimi* havia em cada uma delas. Assim, o ponto de aterrissagem ficará sempre na parte que se encontrar o maior número de pontos *hoshimi*.

Esse método de escolha é bem simples e não implica em um grande esforço computacional que poderia consumir turnos do jogo enquanto era calculado o melhor ponto para começar. Adicionalmente, pensou-se em algumas situações em que essa heurística pudesse não funcionar, mas as opções encontradas envolviam um esforço julgado não justificável.

Na Figura 3-2 está apresentada a heurística utilizada para encontrar o melhor ponto (ou região) de aterrissagem.

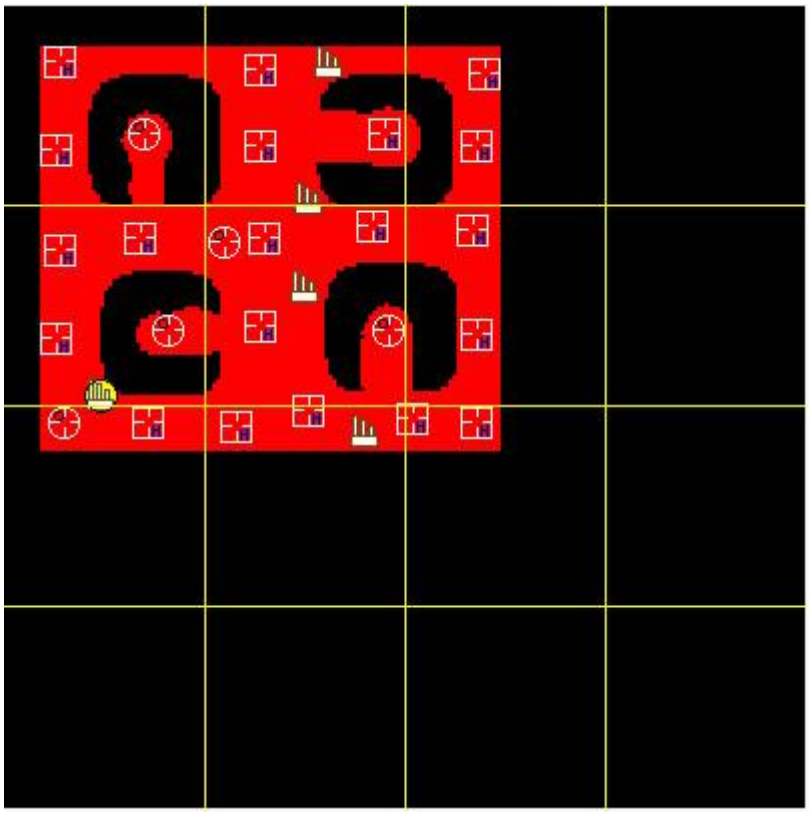


Figura 3-2: Mapa dividido para encontrar a região que contém a maior concentração de pontos *hoshimi*.

3.2.2 Funções objetivo

A fim de codificar um sistema inteligente e que ele pudesse verificar quando precisa fazer o que, lançou-se mão do conceito de funções objetivo as devem ser maximizadas pelo sistema. A idéia central foi a de se criar uma função para cada objetivo do mapa e também uma função global que fosse a soma dessas funções. Além disso, conceberam-se funções locais em cada agente. Isto para que seja verificado quão bem esses estão realizando suas tarefas dados os objetivos globais do sistema. Descrevemos através das equações abaixo as funções objetivo:

$$F_{(Global)} = \sum F_{(objetivo)} \quad \text{(Equação 3-1)}$$

A soma das funções objetivo, se dá pela equação abaixo:

$$F_{(Objetivo)} = \sum F_{(score)} + \sum F_{(alive)} + \sum F_{(navigation)} + \sum F_{(factory)} \quad \text{(Equação 3-2)}$$

Posto desta forma, o objetivo do jogo é maximizar a função global, ou seja, a soma das funções objetivos. Para o objetivo deste trabalho, implementou-se a função de pontuação (*i.e.* aqui referenciada de função de coleta) global e local (*i.e.* a função interna ao agente para avaliação de seus comportamentos). Abaixo está descrito como foram implementadas tais funções.

A função de coleta (*i.e.* pontuação) vai medir a quantidade de pontos por turno conseguidos pelo sistema (*i.e.* função global) e por cada *bot* individualmente (*i.e.* função local). A tabela abaixo apresenta um conjunto de limiares para a função de coleta global que objetivam aumentar o desempenho do sistema (*i.e.* maiores pontuações).

Tabela 3-2: Valores da função de coleta.

F(coleta)	Valoração	Observação
> 2.0	UnNecessary	O *AI* pode focar a algum outro objetivo que sua função esteja ruim.
> 1.3	Great	Nada a fazer, apenas monitorar.
> 0.9	Good	Nada a fazer, apenas monitorar.
> 0.8	Bad	Caso o rendimento tenha piorado (valor anterior > valor atual da função), o AI deve criar mais *containers*.
< 0.8	Critical	O AI deve criar mais *containers* para elevar essa função global de coleta.

A função global (de coleta) é descrita abaixo:

$$F_{(coleta)} = \left(\frac{CS/_{CT}}{OS/_{OT}} \right) \qquad \text{(Equação 3-3)}$$

Onde:
CS: Current Score (*i.e.* Pontuação Atual)
CT: Current Turn (*i.e.* Turno Atual)
OS: Objective Score (*i.e.* Pontuação do objetivo)
OT: Objective Turn (*i.e.* Turno para o término do objetivo)

Caso a função global seja menor que 0,8, *i.e.* crítica, o *AI* vai criar mais dois *containers* para ajudar no objetivo, e estes *containers* vão "nascer" com probabilidade dos mesmos já existentes de *containers* no jogo, *e.g.* 30% de *containers* egoístas e 70% altruístas.

Já a função local, de cada agente, é dada pela divisão entre os pontos conseguidos e os turnos que foram necessários para a consecução dos pontos. Importante, se o valor desta função for menor que a divisão da função global pelo número de *containers*, então o *bot* não conseguiu realizar sua meta e alterna de comportamento (*i.e.* de comportamento egoísta para altruísta ou de comportamento altruísta para egoísta). Esses comportamentos e ações estão descritos na seção seguinte.

3.3 Escolha das ações pelo sistema

Após explicar a função objetivo utilizada neste trabalho, passa-se a explicar a estratégia e as ações de cada *bot* que tem por objetivo o maior número de pontos. Descreveremos as ações dos *bots* que foram citados na configuração inicial do sistema, focando a análise dos comportamentos sociais destes.

3.3.1 AI

Os objetivos do AI nos mapas criados para este trabalho são principalmente construir *needles* nos pontos *hoshimi* e, obviamente, não morrer (*i.e.* sendo atingido por alguma fábrica).

Decorrente da ação de AI, a construção de *needles* impõe a necessidade de bastante reflexão. Não se pode simplesmente construir needles em locais aleatoriamente escolhidos; há de haver uma estratégia para essa ação. Uma alternativa é analisar e otimizar o caminho percorrido pelo *AI*, ou seja, achar o menor caminho para que o *AI* consiga percorrer o maior número de pontos *hoshimi* e poder construir apropriadamente os *needles* que irão ser preenchidos com *oxy*.

Esse problema é bastante conhecido na área de computação como problema do caixeiro viajante[5]. Ele pode ser resolvido de diversas maneiras, mas o que é necessário aqui é uma solução de custo computacional baixo. Isto, pois não se deseja desnecessariamente a perda de turnos ao iniciar o jogo. Com isso em mente, escolheu-se o algoritmo *Simulated Annealing*[11] para resolver esse problema. O algoritmo é basicamente um loop onde são realizadas permutações de caminhos aleatoriamente, sempre se verificando a menor soma no comprimento desses caminhos. A Figura 3-3 mostra um caminho encontrado por este algoritmo.

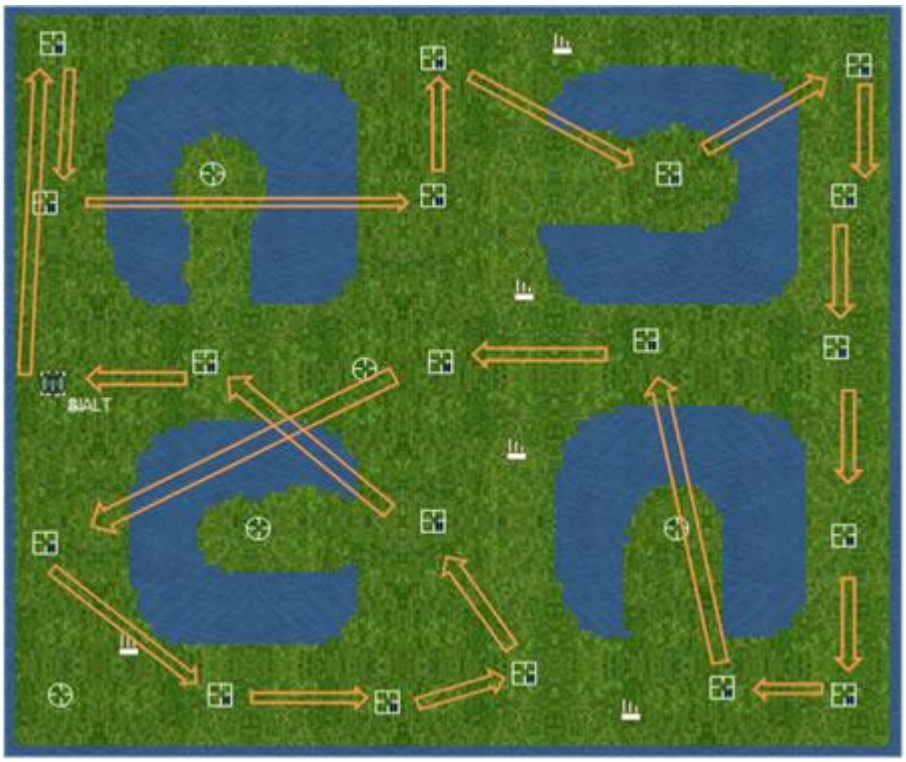


Figura 3-3: Caminho possível encontrado pelo algoritmo Simulated Annealing.

3.3.2 Explorer

A principal característica do *explorer* é a visibilidade. Ele enxerga cerca de três vezes mais longe que o container e por isso foi escalado para a equipe. Existe uma variável interna do jogo que mostra onde os inimigos estão, *OtherBotsInfo*, mas isso somente se algum *bot* do time enxergá-los. O *explorer* tem uma função de patrulha no mapa; ele percorre os mesmos pontos que serão percorridos pelo *AI* para construir os needles e verifica se existe algum inimigo (fábrica) nas redondezas. Assim, caso veja alguma fábrica, a variável *OtherBotsInfo* é preenchida com os seus dados e o *protector* pode dirigir-se a ela para atacá-la.

Apesar de não implementado assim neste trabalho, o *explorer* é muito útil também nos objetivos de navegação (percorrer determinados pontos no mapa), pois ele é o *bot* mais veloz e não é afetado com os diferentes tipos de terreno. Neste trabalho, o *explorer* foi designado para patrulha e, conseqüentemente, para a cooperação com o objetivo de destruição de fábricas.

A tabela abaixo apresenta os atributos implementados para o *explorer*.

Tabela 3-3: Atributos do explorer.

Atributo	Valor máximo	Valor implementado
Container Capacity	0	0
Collect Transfer Speed	0	0
Scan	30	30
Max Damage	0	0
Defense Distance	0	0
Constituition	20	10
Shield	0	0
Max Total Possible	40	40

3.3.3 Protector

O *protector* tem unicamente o objetivo de atacar as fábricas do jogo, seja para completar o objetivo de destruição de fábricas, ou para evitar que muitos *bots* da equipe sejam mortos ao passar por perto das fábricas.

É interessante observar as características do *protector* (*collector*) e perceber que o atributo de ataque é maior que o de visualização, ou seja, ele ataca sem saber por si mesmo se existe algum inimigo. A tabela abaixo apresenta os atributos do *protector*.

Tabela 3-4: Atributos do protector.

Atributo	Valor máximo	Valor implementado
Container Capacity	20	0
Collect Transfer Speed	5	0
Scan	5	5
Max Damage	5	5
Defense Distance	12	12
Constituition	50	25
Shield	0	0
Max Total Possible	50	50

Uma das informações mais importantes que a tabela acima inclui é a limitação 50u distribuídos entre os atributos do *protector*. A última coluna mostra a distribuição no *protector* implementado.

3.3.4 Container

O objetivo do *container* é coletar gás *oxy* de um ponto de *oxy* e transferir para um *needle* localizado em algum ponto *hoshimi*. Esse *needle* deve ser construído pelo *AI*. A partir daí vamos codificar os comportamentos egoístas e altruístas do *container*.

O comportamento codificado para o *container* é mais complexo que os demais, até por que este trabalho está voltado para os comportamentos sociais deste *bot*. A tabela de atributos do *container* está posta abaixo.

Tabela 3-5: Atributos do container.

Atributo	Valor máximo	Valor implementado
Container Capacity	60	50
Collect Transfer Speed	5	5
Scan	0	0
Max Damage	0	0
Defense Distance	0	0
Constituition	60	15
Shield	0	0
Max Total Possible	70	70

Pelos atributos apresentados do *container*, observa-se que ele pode carregar até 60u de gás *oxy*, mas optou-se para uma capacidade de carga de 50u em troca de uma "vida" maior para eventuais ataques de fábricas.

3.3.4.1 Comportamento altruísta

Comportamento altruísta é aquele que o agente pensa no grupo primeiramente ao invés de pensar em si. Por exemplo, um *container* altruísta deve agir em prol de uma maior pontuação do sistema, sem se preocupar com o que pode acontecer consigo. Assim, a codificação realizada resultou na seguinte máquina de estados:

1. Registrar o ponto *hoshimi* que já possui *needle* construído e não tem nenhum outro *container* se dirigindo para ele. Caso não possua, registrar o próximo ponto *hoshimi* onde será construído um *needle*;
2. Registrar o ponto *oxy* que seja mais perto do local do *container* e do ponto *hoshimi* registrado;
3. Dirigir-se ao ponto *oxy* registrado;
4. Coletar a quantidade máxima de *oxy* que o *container* suporta;
5. Dirigir-se ao ponto *hoshimi* registrado;
6. Caso haja *needle* no ponto *hoshimi* de destino, transferir todo a quantidade de gás *oxy* para o *needle*. Caso não exista, esperar 50 turnos para ver se o *AI* constrói o *needle* onde o *container* está localizado. Se passar esses 50 turnos, o *container* muda de comportamento, para egoísta. Foram realizados experimentos em que era esperado a função local ser menor que seu limiar para a troca de comportamento.

3.3.4.2 Comportamento egoísta

Comportamento egoísta é aquele que o agente pensa em si antes de pensar no grupo do qual faz parte. Um *container* egoísta deve agir em prol de si mesmo, sem se importar se aquela atitude vai ser relevante para o sistema ou não. Para justificar o comportamento egoísta, podemos entender que o *container* não quer morrer, que ele deseja continuar fazendo parte do jogo, e assim toma atitudes que o colocam em primeiro plano. Ainda assim, como foi concebido, ele tenta cumprir sua missão: transportar *oxy*. Sua codificação resultou na seguinte máquina de estados:

1. Registrar o ponto *hoshimi* **mais perto** que já tem *needle* construído e não tem nenhum outro *container* se dirigindo para ele. Caso não exista, registrar o ponto *hoshimi* mais próximo de onde o *container* se localiza.
2. Registrar o ponto *oxy* mais perto de onde o *container* se localiza.

3. Dirigir-se ao ponto *oxy* registrado;
4. Coletar a quantidade máxima de *oxy* que o *container* suporta;
5. Dirigir-se ao ponto *hoshimi* registrado;
6. Caso haja *needle* no ponto *hoshimi* de destino, transferir todo a quantidade de gás *oxy* para o *needle*. Caso não exista, esperar 50 turnos para ver se o *AI* constrói o *needle* onde o *container* está localizado. Se passar esses 50 turnos, o *container* muda de comportamento, para altruísta. Foram realizados experimentos em que era esperado a função local ser menor que seu limiar para a troca de comportamento.

3.4 Comunicação entre os agentes

A comunicação entre os agentes acontece através de um *blackboard* (memória compartilhada), onde todos os *bots* armazenam seus estados e a intenção de próxima ação. Da mesma forma que armazenam nessa memória compartilhada, os *bots* verificam quais as ações ou estados dos outros *bots* para poder tomar alguma decisão e não agir causando algum tipo de conflito. Nesse *blackboard* é onde colocamos quais *containers* estão indo para determinado *needle*, por exemplo, para que não haja mais *containers* do que seja suficiente para preencher o *needle*.

Capítulo 4

Experimentos e Resultados

A validação das heurísticas apresentadas anteriormente foi realizadas através de diversos testes com dois mapas distintos e dois *assemblies* diferentes. Em todos os casos se objetivou a análise dos comportamentos sociais dos *bots* em relação ao desempenho do time (de agentes). Abaixo descrevemos detalhes sobre as configurações desses *assemblies*, mapas e experimentos.

4.1 Assemblies

Para que se pudesse apropriadamente avaliar os experimentos, precisou-se acrescentar um módulo supervisor ao código preparado com as heurísticas propostas. *eCollector* é o nome dado para os *assemblies* criados que serão carregados pelo SDK do *Project Hoshimi*.

Antes de descrever as diferenças entre as versões dos eCollectors, vamos descrever o que foi codificado para obtenção dos resultados:

- O algoritmo *Simulated Annealing*, apesar de implementado, não foi utilizado para a realização dos experimentos finais. Esta decisão foi tomadas pois nesse caso haveria mais uma variável a analisar que seria o menor caminho encontrado a ser percorrido pelo *AI*. Como o *AI* está sempre indo para o ponto *hoshimi* mais perto dele, esse cálculo de caminho se tornou desnecessário visto que o *AI* nasce num mesmo local como comentado na seção 3.2.1). Essa observação foi importante, pois em testes preliminares onde se utilizou o *Simulated Annealing*, muitos deles não convergiam na busca do melhor caminho. Assim, o *AI* percorria caminhos não necessariamente curtos o que prejudicava a função global do sistema e conseqüentemente a função local dos *bots*.
- Na observação comentada acima, os *containers* que antes sabiam a rota do *AI*, agora esperavam ele se mover para um ponto *hoshimi* para depois saber o mais próximo dele. Ademais não se preocupou em cálculos de rotas para os pontos mais próximos do ponto onde o *AI* nasce, pois isso não fez parte do objetivo selecionados para este trabalho.
- O *pathfinding* (*A**) não foi totalmente implementado e por isso não foi utilizado nos *assemblies* criados.

- A fim de se analisar melhor o sistema, todos os *assemblies* criados permitem um máximo de dez *containers*. Como o sistema inicia com quatro deles, há possibilidade de se chegar até dez, dependendo da necessidade do sistema.

4.1.1 eCollector 1.0

O *eCollector* 1.0 é a primeira versão do *assembly* criado para realizar os experimentos. Nessa primeira versão, quando o container se encontra em algum ponto *hoshimi* que não contenha *needle* ainda, o *bot* espera 50 turnos e se não for construído o *needle*, o comportamento do *container* é alterado.

4.1.2 eCollector 1.1

O que difere o *eCollector* 1.0 para o 1.1 é somente a mudança de comportamento dos *containers*. Nessa segunda versão do assembly, além do *container* esperar 50 turnos no ponto *hoshimi* que não possui *needle*, para haver a mudança de comportamento, a sua função local (vide seção 3.2.2) deve ser menor do que o um limiar específico. Dessa forma o *container* pode manter um mesmo comportamento, porque sua função local ainda satisfaz o sistema.

4.2 Mapas

Para a realização dos experimentos, não foi utilizado nenhum mapa já contido no SDK do *Project Hoshimi*, a razão foi de se julgá-los muito simples para as avaliações que se pretendia fazer. Assim, criamos dois mapas diferentes que estão descritos abaixo.

4.2.1 Mapa A

O mapa A contém somente o objetivo de pontuação (*ScoreObjective*), contendo 28 pontos *hoshimi* e 6 pontos *oxy*. O mapa tem uma área navegável de aproximadamente 1/3 do mapa inteiro. A figura abaixo mostra o mapa A em detalhe. O objetivo deste mapa é conseguir 4400 pontos em 1500 turnos.

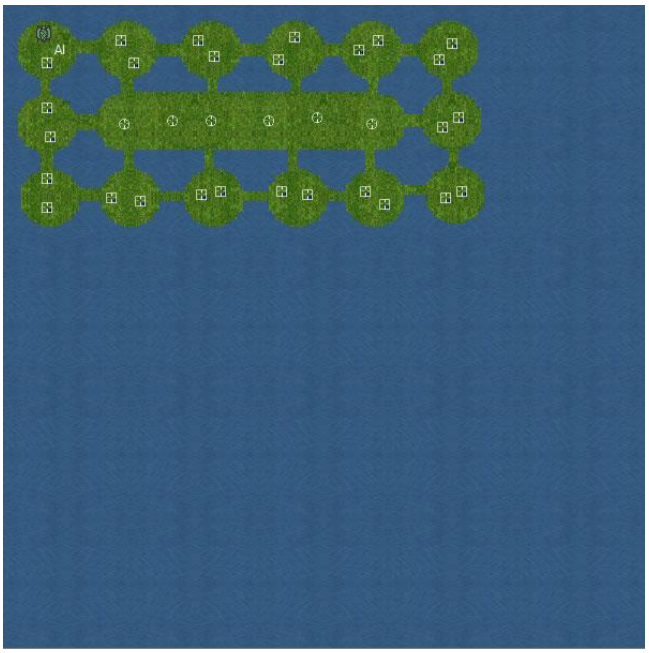


Figura 4-1: Mapa A concebido apenas com objetivos de pontuação.

4.2.2 Mapa B

O mapa B, além de conter o objetivo de pontuação, também contém o objetivo de destruição de fábricas, as quais podem eventualmente exterminar algum *bot* da equipe. Esse mapa se assemelha ao mapa A, mas contém algumas fábricas perto dos pontos *oxy* que podem prejudicar o desempenho dos agentes. A figura abaixo mostra o mapa em detalhes.

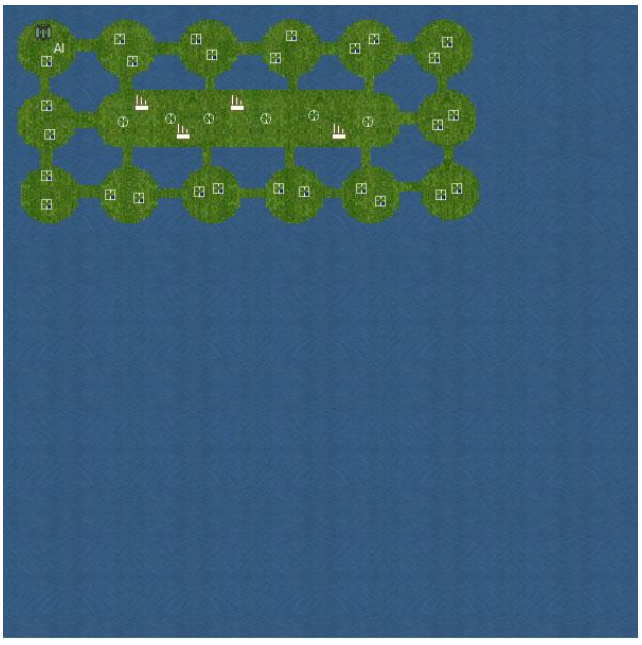


Figura 4-2: Mapa B com objetivo de navegação e destruição de fábricas.

4.3 Realização dos testes

Para que os testes sejam realizados, precisa-se carregar o *assembly* no SDK do *Project Hoshimi* e esperar aproximadamente cinco minutos para um único teste terminar. Para automatizar essa etapa de realização de testes, foi criado uma ferramenta a parte para que se realize a quantidade de testes que se deseja, informando-se o *assembly* e o mapa desejado, sem a necessidade de acompanhamento, ou seja, configura-se a ferramenta para realizar 5 execuções, *i.e.* testes, no mapa A com o *assembly eCollector* 1.0. Essa ferramenta aguarda que cada teste seja finalizado para poder iniciar o próximo teste.

Para a posterior análise de resultados, o módulo supervisor do sistema coleta os dados gerados em intervalos de cem turnos, totalizando quinze coletas em um mesmo jogo.

4.4 Resultados

Nesta seção são apresentados os resultados obtidos aos se carregar os *assemblies* concebidos nos mapas criados. Para cada combinação avaliada (*i.e. assembly* + mapa) analisaremos os gráficos dos experimentos realizados, que são: (i) função global do sistema, (ii) pontuação do sistema, (iii) indivíduos com comportamentos altruístas e (iv) indivíduos com comportamentos egoístas. Todos esses indicadores geraram gráficos em função do número de turnos decorridos.

4.4.1 eCollector 1.0 + Mapa A

Os resultados dessa combinação mostram que no início do jogo, nos primeiros 400 turnos, foram criados novos *bots* egoístas ou altruístas, dependendo da quantidade destes existentes no sistema (que indicam a probabilidade de comportamento dos novos *bots*) por necessidade da função global do sistema. Após o sistema ter estabilizado sua função global, ou seja, quando não se fazia mais necessário a criação de *bots* para realizar os objetivos, a mudança de comportamento dos *bots* convergiu para altruísta, pois eles conseguem que sua função local satisfaça o sistema.

No Gráfico 4-1, pode-se observar a evolução da função global do sistema. No Gráfico 4-2, pode-se observar a evolução da quantidade de indivíduos egoístas no sistema. Nota-se que a quantidade cai radicalmente ao passar dos turnos chegando a zero. No Gráfico 4-3, pode-se observar a quantidade de indivíduos altruístas no sistema. Nota-se também que a quantidade aumenta ao passar dos turnos chegando a dez (o máximo de *containers* permitido pelo *assembly*). Por fim, no Gráfico 4-4, observa-se a pontuação geral do sistema com o decorrer dos turnos. Note-se que houve cinco execuções para fins de análise de variação entre elas.

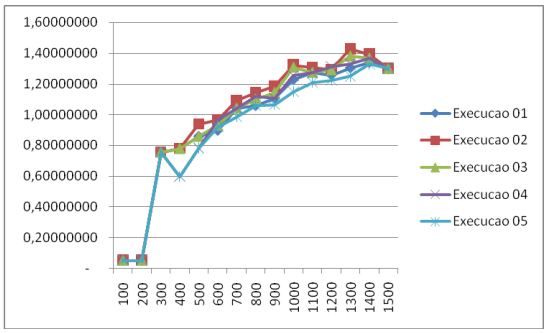


Gráfico 4-1: Função global do sistema.

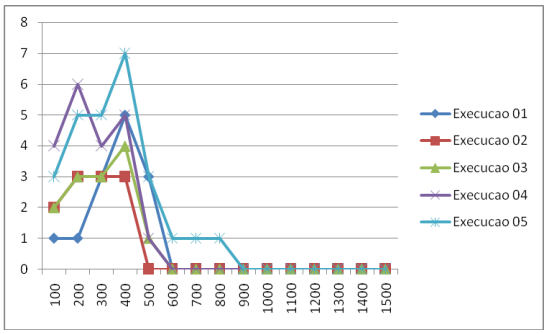


Gráfico 4-2: Indivíduos com comportamentos egoístas.

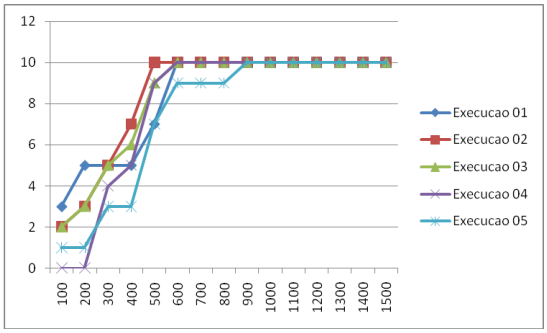


Gráfico 4-3: Indivíduos com comportamentos altruístas.

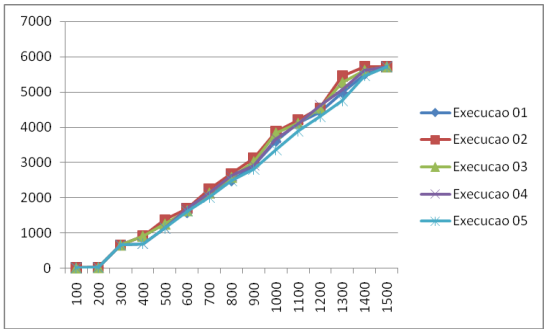


Gráfico 4-4: Pontuação do sistema.

4.4.2 eCollector 1.0 + Mapa B

Os resultados obtidos com essa combinação diferiram razoavelmente dos resultados da seção anterior. As fábricas contidas nesse mapa matam os *containers* que chegam perto para coletar o gás *oxy*. Os *protectors* levam algum tempo para matar as fábricas e estabilizar o sistema para o objetivo de coleta. Analisando-se os gráficos abaixo, pode-se notar as diferenças entre as execuções 02 e 03, onde a execução 02 consegue-se o maior número de pontos e tem a convergência para o comportamento altruísta mais rapidamente. Na execução 03, observa-se através do Gráfico 4-6 e Gráfico 4-7, que muitos agentes morrem, pois a queda em um gráfico não está significando a alta em outro; e somente após 1100 turnos que o sistema tenta se estabilizar criando mais *bots*. No Gráfico 4-5 e no Gráfico 4-8, respectivamente, pode-se notar a evolução da função global do sistema e sua pontuação geral.

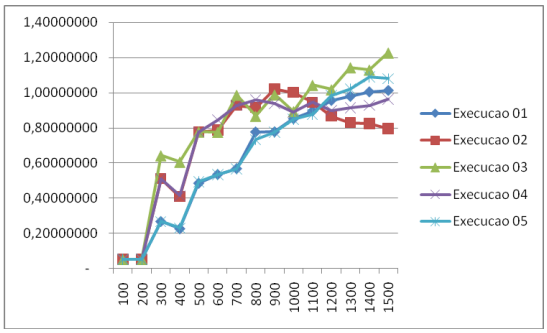


Gráfico 4-5: Função global do sistema.

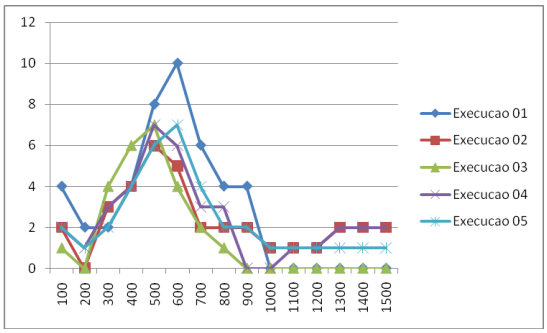


Gráfico 4-6: Indivíduos com comportamentos egoístas.

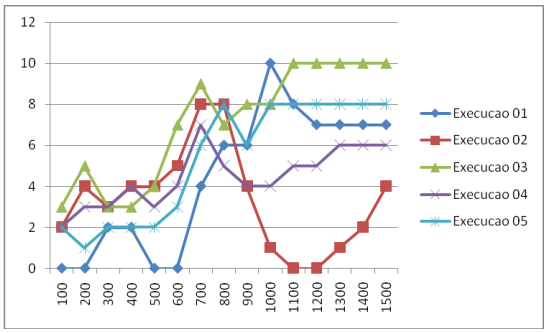


Gráfico 4-7: Indivíduos com comportamentos altruístas.

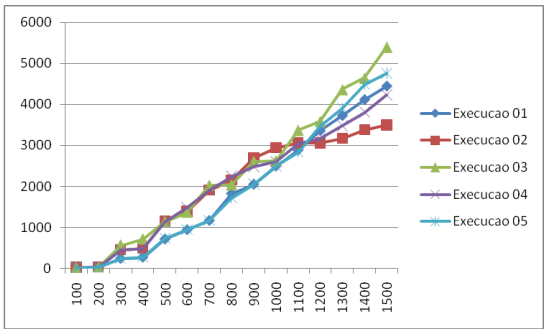


Gráfico 4-8: Pontuação do sistema.

4.4.3 eCollector 1.1 + Mapa A

Com a versão do *assembly* onde os *bots* "insistem" em continuar o seu comportamento até que sua função local não satisfaça o sistema, os resultados para este mapa diferiram um pouco em relação ao *assembly* anterior. Apesar da insistência em seus comportamentos, há convergência para comportamentos altruístas. Da mesma forma que nas seções anteriores, abaixo são apresentados os gráficos obtidos para este experimento.

Pode-se notar no Gráfico 4-12 que a média da pontuação é similar com a, do Gráfico 4-4, com o *eColletor* 1.0.

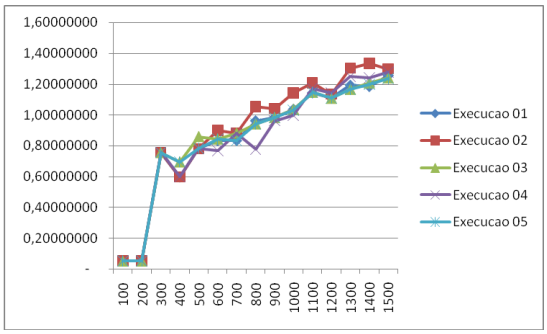


Gráfico 4-9: Função global do sistema.

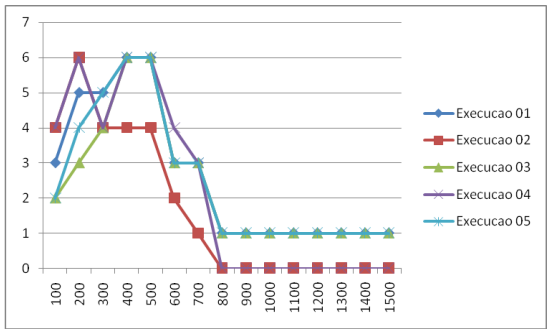


Gráfico 4-10: Indivíduos com comportamentos egoístas.

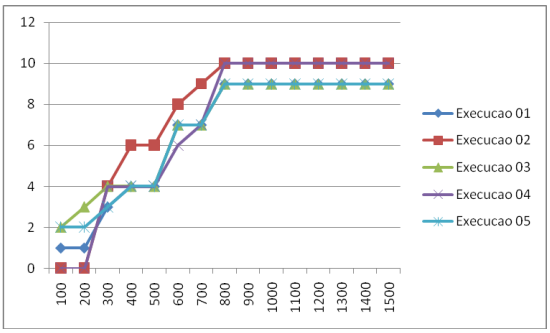


Gráfico 4-11: Indivíduos com comportamentos altruístas.

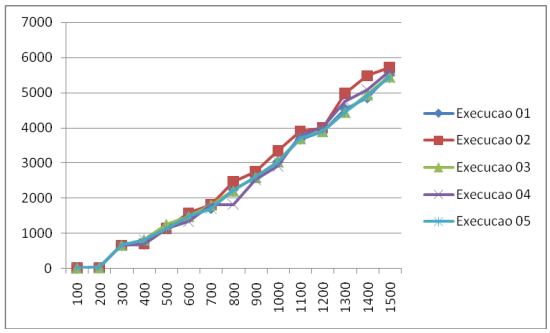


Gráfico 4-12: Pontuação do sistema.

4.4.4 eCollector 1.1 + Mapa B

Nesse último experimento realizado, podemos notar claramente a insistência dos bots egoístas e altruístas. No **Erro! Fonte de referência não encontrada.** e **Erro! Fonte de referência não encontrada.**, respectivamente, pode-se notar o aumento de *bots* egoístas por estes conseguirem viver mais que os altruístas. À medida que não existe mais "perigo", os *bots* mudam seu comportamento para o altruísmo, e caso sua função local não satisfaça o sistema. Pelo **Erro! Fonte de referência não encontrada.** nota-se que a execução 04 obteve o maior número de pontos e verificando os outros gráficos, percebe-se como foi que o sistema conseguiu se adequar: inicialmente, enquanto havia o perigo de fábricas no sistema, tem-se mais bots egoístas, e posteriormente, há a mudança de comportamento para o altruísmo. O **Erro! Fonte de referência não encontrada.** apresenta a função global do sistema.

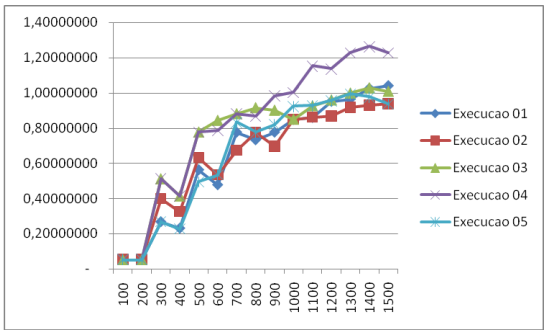


Gráfico 4-13: Função global do sistema.

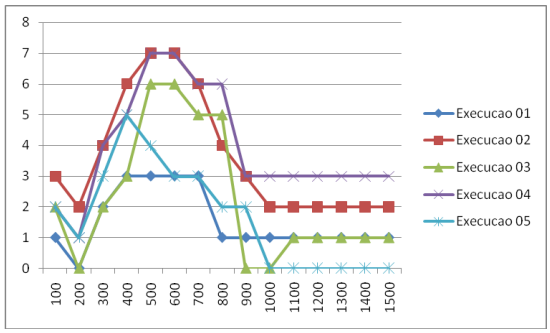


Gráfico 4-14: Indivíduos com comportamentos egoístas.

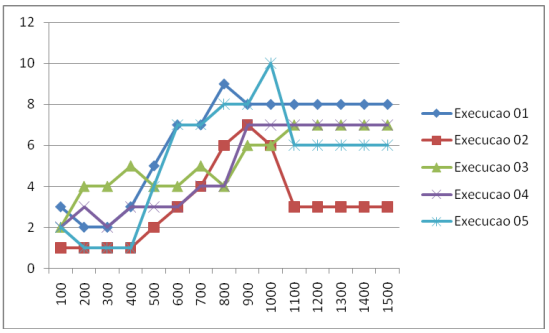


Gráfico 4-15: Indivíduos com comportamentos altruístas.

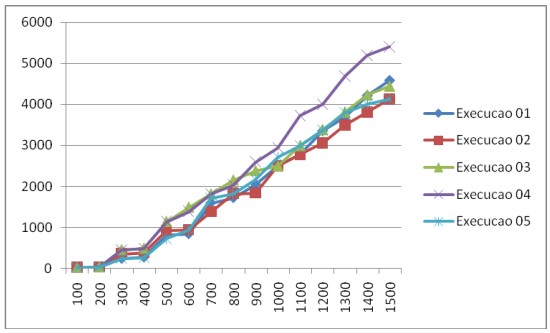


Gráfico 4-16: Pontuação do sistema.

Capítulo 5

Conclusões e Trabalhos Futuros

Este trabalho objetivou a implementação de comportamentos sociais no ambiente do *Project Hoshimi*. Adicionalmente analisaram-se os resultados das simulações realizadas de forma a fim de se verificar o comportamento do sistema com vistas à maximização da sua pontuação. Nesta seção, apresentam-se as conclusões obtidas dos experimentos realizados na seção anterior. O trabalho conclui com a proposição de alguns trabalhos futuros que podem ser interessantes para futuras análises.

5.1 Conclusões

Após serem realizados vários experimentos com *assemblies* e mapas diferentes, podem-se verificar algumas semelhanças nos resultados obtidos:

- Em todos os experimentos, houve convergência para o comportamento altruísta, ou seja, os *bots* perceberam que sendo altruístas poderiam contribuir mais para que o sistema ganhe maior número de pontos;
- Em mapas que contém fábricas, *e.g.* Mapa B, o sistema tentou equilibrar ou até dar preferência aos *bots* egoístas, pois os altruístas morriam mais depressa. Após essas fábricas terem sido exterminadas, o sistema volta a convergir para um comportamento altruísta;
- Outro ponto importante de se notar é que nos primeiros 500 turnos aproximadamente a função global do sistema não é satisfeita e com isso são criados novos *containers*. A partir daí, quando temos um número de *bots* que não são mais aumentados ou reduzidos, podemos verificar como ocorre a mudança de comportamento. Antes desses 500 primeiros turnos, os *bots* são criados de acordo com a probabilidade de *bots* que já existem no sistema;
- Os *bots* que agiram em prol do sistema conseguem completar os objetivos (de coleta) do sistema mais rapidamente. É importante notar que combinando-se as configurações iniciais ideais para o mapa e comportamentos altruístas dos *bots*, a rapidez com que o sistema ganha pontos só dependerá da velocidade na qual o *AI* construir *needles*. Somente aí é importante a utilização de algum algoritmo para verificar o melhor caminho a ser percorrido pelo *AI*.

- Finalmente, em casos onde haja algum perigo no mapa, é preferível uma maior densidade de *bots* com comportamentos egoístas. Pelo menos enquanto persista o perigo, somente então, o *bot* deve alternar para comportamentos altruístas.

5.2 Trabalhos Futuros

Muitos outros objetivos do *Project Hoshimi* não foram explorados neste trabalho. Por isso abaixo é incluída uma lista de trabalhos futuros. Esses trabalhos devem acrescentar muito ao desempenho dos atuais *assemblies* gerados, dotando-os de mais inteligência e possibilitando avanços de mais fases na competição *Imagine Cup*. São as sugestões:

- Criação de funções locais e globais para todos os outros objetivos e analise de qual seria a melhor forma de maximizá-las; Por exemplo, funções para objetivos de navegação, verificando-se os pontos a serem percorridos e o turno em que o objetivo deve estar alcançado, o qual define a sua prioridade.
- Implementação de como dividir recursos (*bots*) em objetivos diferentes;
- Implementação do conceito de patrulha para casos em que haja dois jogadores (jogador VS. Jogador).

Bibliografia

[1] IMAGINE CUP. http://imaginecup.com (Acessado em: 05/11/2007).
[2] IMAGINE CUP – PROJECT HOSHIMI PROGRAMMING BATTLE PORTAL. http://imaginecup.com/Competition/mycompetitionportal.aspx?competitionId=13 (Acessado em: 05/11/2007).
[3] MICROSOFT .NET HOME PAGE. http://www.microsoft.com/net/ (Acessado em: 05/11/2007).
[4] WHAT IS NA ASSEMBLY C NET. http://www.securitypronews.com/articles/applicationdevelopment/spn-19-20040310WhatisanAssemblyCNET.html (Acessado em: 05/11/2007).
[5] O PROBLEMA DO CAIXEIRO VIAJANTE. http://to-campos.planetaclix.pt/neural/hop.html (Acessado em 06/11/2007).
[6] PROJECT HOSHIMI. http://www.project-hoshimi.com (Acessado em 06/11/2007)
[7] S. Russell and P. Norving, *Artificial Intelligence: a modern approach.* New Jersey: Prentice-Hall, 1995.
[8] JENNINGS, Nicholas R. *Coordination Techniques for DAI*. In: O'HARE, Greg; JENNINGS, Nicholas (Eds.). *Foundations of distributed artificial intelligence*. [S.l.]: John Wiley and Sons, 1996. cap.6.
[9] HUHNS, Michael N.; STEPHENS, Larry M. *Multiagent Systems and Societies of Agents*. In: WEISS, Gerhard (Ed.).*Multiagent Systems – A Modern Approach*. [S.l.]: MIT Press, 1999. cap.2.
[10] WOOLDRIDGE, Michael; JENNINGS, Nicholas R. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, v.10, n.2, p.115–152, 1995.
[11] SIMULATED ANNEALING. http://www.dei.unicap.br/~almir/seminarios/2004.2/ts04/annealing/ (Acesssado em 08/11/2007).
[12] AMIT'S A* PAGES. http://theory.stanford.edu/~amitp/GameProgramming/ (Acessado em: 08/11/2007).
[13] ALTRUISMO ATIVA REGIÃO DE PRAZER DO CÉREBRO. http://www.estado.com.br/editorias/2007/01/03/ger-1.93.7.20070103.8.1.xml (Acessado em 09/11/2007).
[14] MAX: RE-DEFINIR ALTRUISMO. http://maxcouti.blogspot.com/2007/08/re-definir-altruismo.html (Acessado em 09/11/2007).
[15] ARTIGO SOBRE O EGOÍSMO. http://www.duplipensar.net/artigos/2005-Q1/egoista-eu.html (Acessado em 09/11/2007).
[16] A VIRTUDE DO EGOÍSMO. http://aartedepensar.com/leit_rand.html (Acessado em 09/11/2007).

[17] THE METROPOLIS ALGORITHM.
http://www.tcm.phy.cam.ac.uk/~ajw29/thesis/node18.html (Acessado em 26/11/2007).
[18] O MÉTODO DE MONTE CARLO.
http://www.ime.usp.br/~durham/cursos/mac115/pub/eps/ep2/ (Acessado em 26/11/2007).

Anexo A

Código Fonte

Apresentaremos aqui algumas partes relevantes do código fonte desenvolvido.

1. **Método de cálculo da função objetivo de coleta:**

```
            public override void CalculateFunction(int currentTurn)
            {
                  ScoreObjective so = Objective as ScoreObjective;

                  decimal actualScorePercent = decimal.Divide(Player.Score,
Player.CurrentTurn);
                  decimal targetScorePercent = decimal.Divide(so.Score,
so.ScoreTurn);

                  this.Value =
Convert.ToDouble(decimal.Divide(actualScorePercent, targetScorePercent));

                  //Atribuicoes de valores
                  //UnNecessary: Value > 2 ou Objetivo = Done ou Failed
                  //Great: Value >= 1.3
                  //Good: Value >= 0.9 e menor que 1.3
                  //Bad: Value >= 0.6 e menor que 0.9
                  //Critial: Value < 0.6

                  if (Value > 2 || so.Status == ObjectiveStatus.Failed ||
so.Status == ObjectiveStatus.Done)
                        Evaluation = HoshimiFunctionEvaluation.UnNecessary;
                  else if (Value >= 1.3)
                        Evaluation = HoshimiFunctionEvaluation.Great;
                  else if (Value >= 0.9)
                        Evaluation = HoshimiFunctionEvaluation.Good;
                  else if (Value >= 0.8)
                        Evaluation = HoshimiFunctionEvaluation.Bad;
                  else
                        Evaluation = HoshimiFunctionEvaluation.Critical;

                  if (this.History.Count > 0)
                  {
```

```
                        this.Delta = this.Value -
this.History.Values[this.History.Count - 1];
                  }
                  this.History.Add(currentTurn, Value);
            }
```

2. Método para avaliar a função global e verificar é necessário alterar o sistema:

```
            private void PerceiveEnvironment()
            {

                  foreach (HoshimiFunction hf in Functions)
                  {

                        hf.CalculateFunction(this.CurrentTurn);

                        if (hf is ScoreFunction)
                        {

                              if (((hf.Evaluation ==
HoshimiFunctionEvaluation.Critical || hf.Evaluation ==
HoshimiFunctionEvaluation.Bad) && hf.Delta < 0) || (hf.Evaluation ==
HoshimiFunctionEvaluation.Critical))
                              {
                                    if (QtdBots[BotType.Container] < 9)
                                    {
                                          AI.StopMoving();
                                          NbrToCreate = 2;
                                          AI_WhatToDoNext =
WhatToDoNextAction.BuildContainer;
                                    }
                                    else if (QtdBots[BotType.Container] < 10)
                                    {
                                          AI.StopMoving();
                                          NbrToCreate = 1;
                                          AI_WhatToDoNext =
WhatToDoNextAction.BuildContainer;
                                    }

                              }

                              foreach (Bot b in Bots)
                              {
                                    if (b.BotType == BotType.Container)
                                    {
                                          ((Container)b).ChangeProb = hf.Value;
                                          ((Container)b).ScoreFunction =
Convert.ToDouble(decimal.Divide((hf.Objective as ScoreObjective).Score,
(hf.Objective as ScoreObjective).ScoreTurn));
                                    }
                              }
                              WolfBrain.LogResults(hf);

                        }

                  }

            }
```

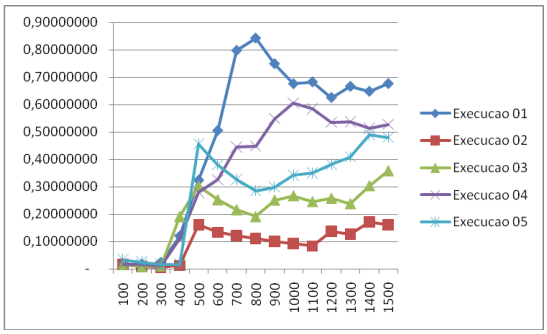
3. **Método de função local de cada *container*, para verificação de mudança de comportamento:**

```
            public void ChangeBehavior()
            {

                  //Verificar a diferenca entre o turno atual e o turno em que
comecou o objetivo.
                  int dif = Wolf.CurrentTurn - initialTurn;
                  double function = (110.0 / dif);
                  double objFunction = (ScoreFunction / numberOfContainers);
                  double d = Wolf.NextRandom.NextDouble();
                  //if (function < objFunction || beginWaitTurn > 0) //nao
insistir no comportamento caso a funcao nao seja menor.. (outra analise)
                  if (function < objFunction) //insistir no comportamento caso
a funcao nao seja menor.. (outra analise)
                  {
                        Behavior = Behavior == BehaviorType.Altruistic ?
BehaviorType.Egoistic : BehaviorType.Altruistic;
                        initialTurn = Wolf.CurrentTurn;
                  }

            }
```

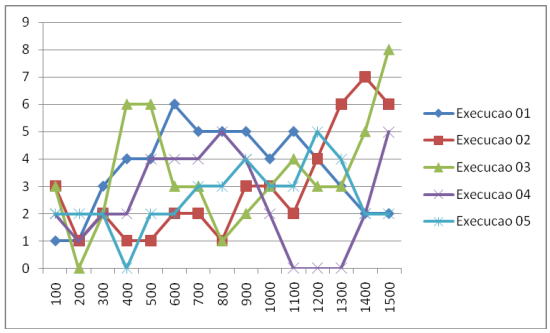
Anexo B

Resultados utilizando o Simulated Annealing

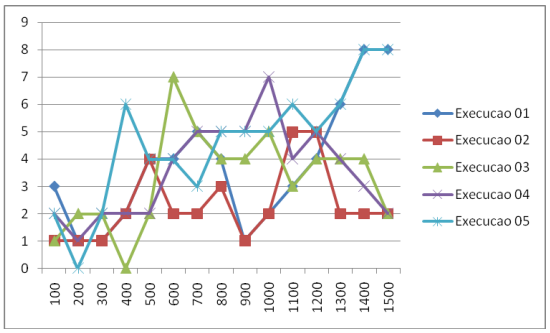
Os resultados abaixo mostram o assembly 02, utilizando o *Simulated Annealing* para a escolha do melhor caminho no mapa B.



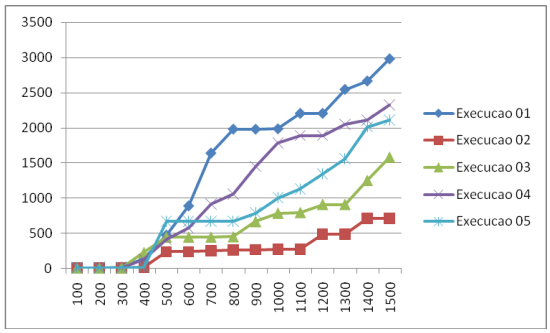
Função global do sistema.



Indivíduos com comportamentos egoístas.



Indivíduos com comportamentos altruístas.



Pontuação do sistema.