UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

TRABALHO DE GRADUAÇÃO

UMA ESTRATEGIA EFICIENTE DE COLETA MULTIAGENTE PARA JOGOS RTS.

José Carlos de Moura Júnior <jcmj@cin.ufpe.br>

Orientador: Geber Ramalho <glrt@cin.ufpe.br>

Recife, Setembro de 2006

ASSINATURAS

Este Trabalho de Graduação é resultado dos esforços do aluno José Carlos de Moura Júnior, sob a orientação do professor Geber Lisboa Ramalho e Patrícia Cabral de Azevedo Restelli Tedesco, sob o título inicial de "*UMA ESTRATEGIA EFICIENTE DE ABASTECIMENTO MULTIAGENTE PARA JOGOS RTS.*", posteriormente modificado para "*UMA ESTRATEGIA EFICIENTE DE COLETA MULTIAGENTE PARA JOGOS RTS.*". Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação.

_____

José Carlos de Moura Júnior

_____

Geber Lisboa Ramalho

_____

Patrícia Cabral de Azevedo Restelli Tedesco

"Como eu bebia muito, fiquei encarregado de arrumar garrafas para fazer bomba molotov"

— Chico Buarque

À MEMÓRIA DE MANOEL GONÇALO DE MOURA E MANOEL DA SILVA PAIZINHO.

AGRADECIMENTOS

"Quem pagará o enterro e as flores se eu me morrer de amores?"
— VINICIUS DE MORAES

Gostaria de agradecer a todas aquelas pessoas que contribuíram de alguma maneira para a conclusão deste trabalho:

- Primeiramente a gostaria de agradecer a minha namorada, que sempre me apoiou e entendeu a minha eterna indisponibilidade de tempo (na maioria das vezes).
- Gostaria de agradecer a minha família, especialmente meus pais, que sempre estiveram presentes nos momentos em que eu necessitei.
- Um agradecimento especial a Geber Ramalho e Patrícia Tedesco, que me orientaram nos momentos em que eu estava totalmente perdido para elaborar esse trabalho.
- Aos meus amigos que contribuíram durante o desenvolvimento da ferramenta e da elaboração da solução.
- A Manifesto Game Studio que cedeu o espaço físico necessário para o desenvolvimento da ferramenta.
- Ao o glorioso Sport Club do Recife que tantas alegrias me proporcionou nesse ano, pois esse ano nosso time vai ser mesmo campeão.

- A Jynx Playware e a todos os amigos que eu fiz, principalmente Scylla que concedeu folgas necessárias para a conclusão deste trabalho.

RESUMO

Em jogos de estratégia em tempo real (RTS), os agentes são responsáveis por várias tarefas, dentre a quais a de abastecimento (foraging), sair a procura de recursos, coletá-los e levar de volta a um centro. Na maioria dos casos o abastecimento deve ser garantido não por um agente único, mas por um conjunto deles (portanto, um sistema multiagente). É preciso que exista uma coordenação de modo que essa coleta seja de feita de modo eficiente, a ação de cada agente deve ser definida pensando no coletivo, e deve haver uma negociação entre eles para a definição da responsabilidade de cada um. O objetivo desse trabalho é propor uma técnica eficiente para este foraging, de maneira a maximizar a quantidade de recursos coletados em um determinado tempo. Para implementar e testar a eficiência desta estratégia foi desenvolvido um simulador o JARTS, um simulador voltado para pesquisas de Inteligência Artificial em jogos RTS.

Palavras-chave: jogos RTS, Foraging, JARTS, Simulador

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

“A journey of a thousand miles must begin with a single step”
— LAO-TZU

Os jogos de estratégias existem desde a antiguidade, como o Xadrez, existente desde o ano 600 d.c. , e que até hoje continua muito popular. Com o advento dos computadores um gênero de jogo que se popularizou bastante foi o de estratégia de tempo real, também conhecidos como jogos RTS, em que o usuário deve comandar um grupo de agentes, em tempo real, diferentemente do Xadrez, em que as jogadas ocorrem em turnos.

Os agentes controlados pelo computador são conhecidos como NPC (non player character), e eles precisam ter um comportamento natural, de maneira que não consiga diferenciar se quem está comandando aquele agente é um ser humano ou um computador. O objetivo da Inteligência Artificial é dar um nível maior de inteligência a esses NPC’s tornando o comportamento deles muito mais natural, assemelhando-se ao comportamento de um agente controlado por um humano.

Os jogos do tipo RTS possuem uma grande quantidade de problemas que podem ser resolvidos com a ajuda da Inteligência Artificial. Um problema como definir a estratégia de coleta de recursos, envolve uma série de outros problemas como coordenação multi-agente, path finding, alocação de recursos, entre outros, e a coleta de recursos é apenas um dos vários problemas existentes a serem resolvidos pela Inteligência Artificial em jogos RTS, pois o jogador precisa tomar decisões tanto de ordem tática como estratégica.

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver uma abordagem eficiente para o abastecimento de recursos multi-agente de um jogo RTS e a construção de um simulador que permita a análise comparativa de tal abordagem. O cenário em que o jogo ocorre é um mundo quadrangular divdido em tiles de tamanhos iguais, e 32x32 tiles é o tamanho. Neste mundo não existe unidades inimigas e o jogador inicia a partida com 20 trabalhadores e 1 Control Center.

Este trabalho segue a seguinte linha de raciocínio, primeiramente são feitas explicações sobre o que são jogos RTS, e que contribuições a Inteligência Artificial pode trazer aos jogos deste gênero. Em seguida é apresentada a importância dos simuladores nesse gênero de jogo e é feita uma análise sobre os simuladores existentes, como ORTS e Stratagus. No capítulo seguinte o problema do foraging (coleta de recurso) é abordado , explicado em detalhes e o estado da arte é mostrado. Em seguida é proposta uma abordagem para o problema e explicado como se deu o seu desenvolvimento. Na sequência é mostrada a ferramenta construída para a análise da abordagem e seu estudo comparativo, que será utilizado pelos alunos da cadeira de Agentes Autônomos na instancia de 2006.1. No capítulo final são mostrados os resultados obtidos com a estratégia e um estudo comparativo entre a estratégia desenvolvida nesse trabalho e aquelas que foram implementadas pelos alunos da cadeira de Agentes Autônomos.

CAPÍTULO 2

JOGOS RTS

Neste capítulo será explanado o que são jogos RTS, seu funcionamento e suas caractéristicas básicas, em seguida seguida será mostrado o uso da IA nesse tipo de jogo e a importância da utilização de simuladores nesse tipo de jogo.

2.1 O QUE SÃO

Estratégia de tempo real (RTS, do inglês Real Time Strategy), é um gênero de jogo, normalmente caracterizados por serem jogos de guerra, em que a disputa acontece em tempo real. [RTS - Wikipedia]

Nesse tipo de jogo o jogador tem controle total sobre suas unidades e ele é responsável direto pela tática e estratégia a ser seguida, através dessas unidades é que o jogador vai poder interagir com o mundo, tais unidades podem ser um prédio, um tanque de guera, ou um mero aldeião. As principais ações a serem executadas pelo jogador que é comum em todos os jogos RTS são as seguintes:

- Coleta de Recurso: O jogador necessita de unidades especiais que fazem a coleta de recurso, e precisa que essa coleta seja feita de forma otimizada, pois tais recursos serão consumidos para ele executar as tarefas de construir edificações, treinar unidades e desenvolver tecnologia. A variedade de recursos pode variar de jogo para jogo, em Command & Conquer, por exemplo, o usuário coleta um único tipo de recurso (minério), enquanto em Age Of Empires II, ele necessita coletar 4 tipos de recursos (ouro, pedra, madeira e comida).

- Construir edificações: Para treinar as suas unidades e desenvolver tecnologias, o jogador precisa construir edificações, onde tais ações serão executadas. Cada tipo de edificação tem um propósito diferente, uma serve para treinar unidades do tipo A, outra para treinar unidades do tipo B, uma terceira edificação serve para evoluir a unidade do tipo B para do tipo C.

- Treinar unidades: O jogador precisa treinar unidades para combater o exército inimigo, e como foi visto anteriormente cada uma dela é treinada em uma edificação diferente. Unidades podem ser navios, canhões, trabalhadores, entre outros, e cada unidade tem caractéristicas diferentes e por isso finalidades diferentes, um por exemplo, coleta madeira, enquanto outra atira flechas. Cabe ao jogador decidir qual unidade treinar em função de sua tática/estratégia.

- Desenvolver tecnologia: Também conhecida como evolução, ela é necessário nesse tipo de jogo para que o jogador tenha maiores chances no combate com o seu inimigo, pois desenvolvimento de tais tecnologias vão tornar as suas

unidades melhores. Ele pode evoluir uma unidade para que ela tenha um poder de destruição maior, ou para que ela se movimente mais rápido, é possível até evoluir edificações para que tais tenham maior resitência contra ataques inimigos.

- Combate o exercito inimigo: Todas as ações anteriores são feitas visando o combate com o inimigo, pois o objetivo principal desse gênero de jogo é destruir todas as unidades inimigas. No combate o jogador faz decisões tanto do nível estratégico, como decidir se vai atacar ou não, quanto no nível tático, onde ele decide que tipo de formação dar ao seu exército ou onde cada unidade deve atacar.

Além de todas essas tarefas, o jogador precisa tomar decisões no nível estratégico, como decidir se é melhor atacar naquele momento, ou se é melhor se defender e coletar mais recursos para treinar mais unidades e fazer um ataque mais forte posteriormente. Ele precisa saber também como gerenciar os recursos da melhor maneira possível, saber se naquele momento é mais importante construir 10 unidades do tipo A, ou construrir apenas apenas 5 e utilizar o restante dos recursos para evoluí-las, tornando elas mais fortes.

Em um jogo RTS o jogador precisa tambem tomar decisoes em um nível mais baixo, no nível tático. Por exemplo, no nível estratégico ele toma decisões como atacar ou defender, enquanto no nível tático, ele vai definir como vai atacar, qual a formação que as suas unidades vão ter, se vão em bando, ou isolados, decisões deste tipo.

2.2 IA EM RTS

Inteligência Artificial em jogos é qualquer implementação que passe uma noção de inteligência em algum nivel, tornando o jogo mais imersivo, desafiador e o mais importante, divertido. A IA é utilizada em duas situações, na primeira ele executa alguma tarefa para o jogador humano, como calcular a rota das suas unidades, e a segunda situação é quando ela controla os personagens que não são controlados pelo jogador humano, simulando um jogador adversário por exemplo. Dessa maneira a indústria de jogos vem evoluindo suas tecnicas e utilizado-as em seus jogos, como uma maneira de atrair mais jogadores, a tabela a seguir mostra como a IA é utilizada em jogos desde muito tempo.

Ano	**Descrição**	**IA utilizada**

1962	Primeiro jogo de computador, Spacewar, para 2 jogadores.	Nenhuma
1972	Lançamento do jogo Pong, para 2 jogadores.	Nenhuma
1974	Jogadores tinham que atirar em alvos móveis em Pursuit e Qwak.	Padrões de movimento
1975	Gun Fight lançado, personagens com movimentos aleatórios.	Padrões de movimento
1978	Space Invaders contém inimigos com movimentos padronizados, mas também atiram contra o jogador.	Padrões de movimento
1980	O jogo Pac-man conta com movimentos padronizados dos inimigos, porém cada fantasma (inimigo) tem uma "personalidade" sobre o modo em que caça o jogador.	Padrões de movimento
1990	O primeiro jogo de estratégia em tempo real, Herzog Wei,é lançado. Junto, os jogadores puderam noticiar uma péssima busca de caminho.	Máquina de estados
1993	Doom é lançado como primeiro jogo de tiro em primeira pessoa.	Máquina de estados
1996	BattleCruiser: 3000AD é publicado como o primeiro jogo a utilizar redes neurais em um jogo comercial	Redes neurais
1998	Half-Life é lançado e analisado como a melhor IA em jogos até a época, porém, o jogo utiliza IA baseada em scripts.	Máquina de estados / Script
2001	O jogo Black & White é alvo da mídia a respeito de como as criaturas do jogo aprendem com as decisões feitas pelo jogador. Utiliza redes neurais, reinforcement e observational learning.	Diversos

Tabela 1 - Historia da IA em jogos

Em jogos de estratégia de tempo real (RTS), é possível destinguir diversos módulos de Inteligencia Artificial.Um dos mais básicos pode ser um sistema de path-finding, que vai muito além de descobrir um caminho entre o ponta A e o B, ele precisa ser eficiente ao ponto de calcular as rotas de varias unidades em uma fração de segundos, conseguindo obter a melhor rota e evitar colisões entre os caminhos.

Existem também sistemas de IA que tratam de probelmas em um nível mais alto, como modulos responsáveis pela economia, desenvolvimento, ou até um módulo

responsável pela análise de terreno, presente em praticamente todos os jogos RTS, que vai auxiliar em decisões como onde construir edificações específicas, de maneira que elas não fiquem vulneráveis. Baseado em um mapa de influência gerado pela análise de terreno um módulo de IA responsável pelo combate pode determinar qual a melhor formação do exército.

Os sistemas de IA de um jogo RTS podem resolver problemas específicos como coleta de recurso, considerando tendo informação perfeita do mundo e um grupo de agentes autônomos qual a melhor maneira deles se organizarem de modo a coletar a maior quantidade de recurso no menor tempo possível.

Os jogos *Dark Reign: The Future Of War*, *Battlezone*, e *Civilization: Call To Power,* são bons exemplos de como a IA pode ser dividida em módulos. Nesses jogos a IA é dividida em duas tarefas principais. IA Tática e IA estratégica, que posteriormente vai ser dividida em três partes: Análise, Alocação de Recursos e IA de Alto Nível (personalidade). Cada módulo desses podem ser implementados utilizando maquinas de estado, sistemas baseados em regra ou logica fuzzy. [Davis 1999]

IA Tática, entre várias tarefas, define, por exemplo, que cada unidade vai verificar se existe alguma unidade inimiga nas redondezas, e quando encontrar cada unidade escolhe qual a melhor unidade para atacar e o faz. A melhor unidade a ser atacada é definida maximizando quanto de poder de fogo do inimigo eu posso retirar o mais rapidamente, desta maneira as unidades mais fortes acabam destruindo as unidades mais fracas primeiramente, e dessa maneira é garantido que unidades que não sejam de combate so vão ser atacadas quando não houver mais unidades de combate.

IA Estratégica como o próprio nome diz toma decisões de nível estratégica com o uso de informações adquiridas através de outros módulos como de análise, alocação de recurso e módulo de alto nível

Análise: O objetivo do modulo de analise é definir os objetivos estrategicos atuais e classifica-los por prioridades. Os objetivos estratégicos pode ser exploração, reconhecimento, construção de base, objetivos defensivos e objetivos ofensivos.

Alocação de Recursos: Cada recurso pode ser um trabalhador, um arqueiro, um prédio, ou uma quantidade de ouro por exemplo. Um objetivo pode ser treinar uma quantidade grande de uma uniade. Como mostra a figura abaixo, cada tarefa pode necessitar de mais de um recurso, e cada tarefa tem uma prioridade diferente, então elas são organizadas de maneira decrescente e as tarefas são realizadas primeiramente.

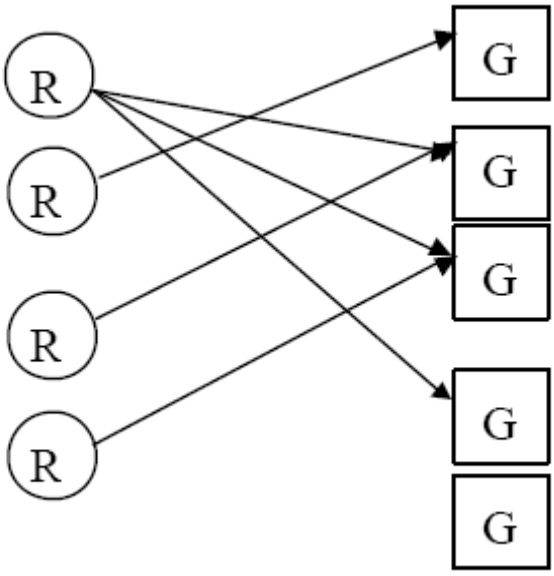

Ilustração 1 - Alocação de Recursos

Alto Nível: Como o próprio nome diz esse módulo têm uma visão mais alto nível do problema, ela pode coordenar as tropas como uma espécie de general. Ela pode tomar decisões do tipo formar alianças.

Como foi dito anteriormente o objetivo da IA em jogos RTS é tornar o jogo mais imersivo e divertido, porém alguns jogos para impor um nível de dificuldade mais alto para o jogador acabam 'roubando', ou seja, suas unidades coletam ouro mais rápido que as unidades do jogador, ou eles tem informações que se fosse um jogador humano não teria, e quando isso transparece para o jogador, o usuario final, o jogo acaba perdendo a diversao. [Davis 1999] A construção de tecnicas de IA eficientes, têm por objetivos conseguir um desempenho excelente sem infringir as regras.

2.3 CONCLUSÃO

Nesta seção foi explicado o que são jogos RTS, como funcionam, e suas ações básicas. Foi visto também que é extremamente importante fazer a coleta de recurso de maneira o mais eficiente possível, pois todas as ações seguintes, construir edificações, treinar unidades, desenvolver tecnologias e combater o exercito inimigo, consomem recursos e o jogador precisa cumprir todas essas ações de maneira rápida, de modo que se o seu inimigo coletar recursos de maneiras mais eficiente que ele, o inimigo vai poder construir mais edificações, treinar uma maior quantidade de unidades, evoluir todas elas, e finalmente vai acabar tendo uma maior vantagem no momento do combate. Em um segundo momento foi mostrada a empregabilidade da Inteligencia Artificial em jogos RTS que problemas eles ja resolvem e quais podem resolver.

CAPÍTULO 3

SIMULADORES

3.1 IMPORTÂNCIA DOS SIMULADORES DE RTS

A pesquisa de novas técnicas, abordagens e algoritmos na área de Inteligência Artificial necessitam de simuladores para viabilizar possíveis comparações entre as diferentes soluções para o mesmo problema.

Por exemplo, no desenvolvimento de um jogo de futebol, eu preciso implementar um agente que seria o jogador, porém se faz necessario a existencia de um ambiente de simulação em que esse agente possa atuar, e assim análisar o seu comportamento fazendo um estudo comparativo com os outros agentes.

Um exemplo bem sucedido de simulador de Inteligencia Artificial é o RoboCode, uma aplicação educativa criada pela IBM, que permite a criação de tanques de guerra, e torna possível a disputa entre a implementação de diferentes simuladores. Existem hoje vários campeonatos em que os competidores programam seus tanques e eles competem entre si.

Jogos RTS oferecem uma quantidade unica de de desafios para a Inteligencia Artificial, como planejamento sem informação perfeita, aprender a tática do adversário, analisar o terreno observável dentre outros que foram mostrados anteriormente. Porêm as pesquisas de IA nessa área encontram uma barreira, a ausência de simuladores. Impedindo de tal maneira estudos comparativos entre diferentes abordagens.

Para implementar uma abordagem nova, ele precisaria criar um simulador, se preocupando com parte grafica para visualisar o comportamento do agente, vai ter que implementar toda uma simulação de um jogo RTS com suas restrições. O que acaba tirando o foco do pesquisador do principal que é desenvolver uma solução para os problemas de um RTS.

3.2 REQUISITOS

Para a implementação da abordagem selecionada é necessária a utilização de um simulador, que permitirá um estudo comparativo da solução. Tal simulador precisa de alguns requisitos básicos como foco no problema, ou seja, o simulador precisa manter o desenvolvedor focado na parte de Inteligência Artificial, sem se preocupar em desenvolver

um jogo RTS inteiro. Documentação é outro requisito necessário para o simulador, uma vez que se o desenvolvedor sentir muita dificuldade para implementar a sua solução ele vai acabar desistindo do uso de tal ferramenta. E por último, mas não menos importante, estabilidade, o simulador tem que permitir o desenvolvedor evoluir a sua soluçao sem se preocupar se o simulador vai conseguir reproduzir tudo aquilo que ele deseja, ou se o simulador vai falhar.

3.3 ESTADO DA ARTE

3.3.1 STRATAGUS

O Stratagus é uma engine, open-source, para desenvolvimento de jogos RTS, que funciona independente de plataformas. Ela da suporte tanto ao jogo single-player (em que o usuário joga contra o computador), como ao jogo multiplayer (no qual o usuário joga com um outro ser humano, através da internet). Uma versão pre-compilada pode ser obtida através de sua página (http://stratagus.sourceforge.net). Enquanto a ultima versão do codigo que é constantemente atualizada pode ser encontrada através do seu repositorio CVS.[STRATAGUS]

Os jogos que utilizam a engine Stratagus são implementados através de scripts, utilizando a linguagem de script LUA. Alguns dele disponibilizam o seu código na página do Stratagus.Existem jogos bastente populares que utilizam a engine Stratagus, como o Wargus (um clone de um jogo bastante popular de RTS o WarCraft II lançado pela Blizzard Entertainment) que é mostrado na figura abaixo.

Ilustração 2- Wargus – Stratagus

Infelizmente o Stratagus não possui documentação sobre o desenvolvimento de um jogo utilizando sua engine. Em sua página ele afirma que se você deseja desenvolver um jogo com ela, você deve baixar algum jogo feito e utiliza-lo como template. Além disso o fato de ele não ser um simulador acaba tirando um pouco o foco do problema. Uma vez que o usuário precisa desenvolver um jogo por completo para poder implementar a sua solução.

3.3.2 ORTS

O ORTS [ORTS], acrônimo de Open Real-Time Strategy, desenvolvido por Michael Buro da Universidade de Alberta no Canada, é um ambiente de programação, para o estudo de problemas de Inteligencia Artificial em jogos RTS, como path finding, tratamento de informação imperfeita. O código do ORTS é disponibilizado através de seu repositorio CVS.

O ORTS utiliza uma arquitetura cliente-servidor. Do lado do servidor o usuário tem a possibiliadade de configurar o tipo de jogo que ele deseja executar, quantidade de jogadores, quantidade de unidades que cada jogador irá possuir no início, se os jogadores vão ter informação perfeita ou não. Através de script o usuário pode definir todos os tipos de unidade, estruturas e interações que elas vão ter no jogo. No lado do cliente, onde o usuário implementa a Inteligencia Artificial, ele se conecta ao servidor e define as ações dos objetos.

A cada loop o servidor manda para o cliente a visão do jogador, e recebe as ações de cada um dos objetos do jogador, que em seguida são executadas. Esse loop é executado muitas vezes por segundo. O cliente pode ter uma visão 3D do mundo renderizado com OpenGL, como mostra a figura abaixo.

Ilustração 3 - Visão 3D do cliente - ORTS

Na AIIDE 2006(Artificial Intelligence and Interactive Digital Entertainment) , houve uma competição baseado no ORTS [ORTS AIIDE 2006], em que os competidores tinham que desenvolver soluções para os 3 tipos de jogos pré configurados, que são os seguintes:

- Jogo 1: Jogo single-player (um único jogador), com informação perfeita, o jogador tem conhecimento total de todo o mapa, gerado randomicamente formando uma matriz de 32 por 32 tiles, com vários obstáculos e recursos espalhados pelo mundo. O jogador possui um Contol Center e 20 Workers. O objetivo é utilizar os workers para coletar o máximo de recurso e levar para o Control Center num tempo pre-determinado.

- Jogo 2: Jogo Multi-player (2 jogadores), cada jogador possui um grupo de tanques e alguns Control Centers. O objetivo de cada jogador é destruir os Control Centers inimigos e proteger os seus. No final quem tiver destruido a maior quantidade de prédios inimigos vence.

- Jogo 3: Um jogo RTS completo em que 2 competidores, além da coleta de recurso e do combate, precisam treinar unidades e fazer o gerenciamento destes recursos, saber o momento de armazenar recurso, que unidade treinar, e tomar decisões estratégicas e táticas.

O ORTS, da mesma maneira que o Stratagus, peca pela falta de documentação. Possui documentação escassa e de dificil compreensao. Além disso código do ORTS disponibilizado atraves do seu CVS, muitas vezes possuem falhas de compilação e de execução, impedindo o usuário de trabalhar com uma ferramenta altamente instavél. Para conseguir executar o ORTS é necessário rodá-lo no Linux, ou utiizar um emulador do Linux como MingWin, através de um passo-a-passo, oferecido no site do ORTS, de dificil entedimento.

3.3.3 OUTROS

GLEST

Glest é um jogo de estratégia em tempo real, que ocorre em uma época medieval, open-source, configurável através de XML e uma série de ferramentas. O Glest tem documentação muito escassa. As informações encontradas sobre ele tratam somente do Glest como jogo e não como engine. [GLEST]

BOSON

Boson é um jogo de estratégia de tempo real, utiliza OpenGL, as bibliotecas do KDE e é desenvolvido para rodar em computadores com o sistema operacional Linux. Não possui documentação sobre o seu desenvolvimento. [BOSON]

3.4 ANÁLISE

Neste capítulo será analisado cada um dos simuladores apresentados na seção anterior. Eles serão avaliados segundo os seguintes quesitos:

- Foco no problema – Ele permite o usuário se focar unicamente no problema de Inteligência Artificial em jogos RTS.
- Estabilidade do Sistema – O sistema é estável, permitindo o usuário fazer as suas simulações necessárias.
- Documentação – O sistema possui documentação suficiente para auxiliar os desenvolvedores.

	Foco no problema	Estabilidade do Sistema	Documentação
Stratagus		●	
ORTS	●		
Glest		●	
Boson		●	

Tabela 2 - Análise dos simuldores

Como mostra a tabela acima nenhum dos sistemas apresentados atende minimamente as necessidades básicas requeridas para desenvolver uma pesquisa na área de Inteligência Artificial focada em Jogos RTS. Devido a isso foi decidido desenvolver uma ferramenta que seria utilizada na implementação da solução abordada nesse trabalho. Informações sobre essa ferramenta, detalhes de implementação e modo de uso serão vistos no capítulo 6.

CAPITULO 4

COLETA

Esse capítulo abordará a importância da coleta de recursos em um RTS, e as técnicas e soluções utilizadas para coelta em geral.

4.1 O PROBLEMA

O problema da coleta de recursos, também conhecido como foraging, é o primeiro problema com que o jogador se depara no momento de jogar um jogo do tipo RTS, pois como foi visto no capitulo sobre jogos RTS, tal tarefa é a base para que o jogador adquira recursos para conseguir efetuar as etapas seguintes, contruir edificações, treinar unidades, desenvolver tecnologias e combater o exército inimigo.

O problema que esse trabalho se propõe a desenvolver uma abordagem foi visto em uma competição que ocorreu em Junho de 2006 na AIIDE 2006 (Artificial Intelligence and Interactive Digital Intertainment) . Na competição o usuário precisava programar um grupo de agentes que tinham coletar recursos espalhados por um cenáro e levá-los para um centro, simulando o problema da coleta de recurso dos jogos RTS.

O foraging é um problema complexo em que é necessário utilizar a capacidade máxima dos agentes, fazer com que eles carreguem o máximo de recursos, é preciso coletar o máximo de recurso no menor tempo possível e além de tudo isso é necessário fazer a coordenação e sincronismo entre os agentes.

O foraging têm várias caracteristicas que o descrevem, abaixo vão ser descritas as principais caracteristicas que definem o tipo dele:

- Mono Agente X Multi Agente: No mono agente, como o próprio nome diz, um único agente é responsável pela coleta, enquanto no Multi Agente um grupo de agentes é responsável pela coleta.
- Central Place X Various Place: Refere-se a quantidade de pontos de depósito a disposição dos agentes. No caso do central place, tudo que for coletado precisa ser levado para um único ponto. Se os agentes tiverem mais de um ponto de depósito, então é Various Place foraging.
- Recorrent X One Shot: One Shot, siginifica que o agente so precisa visitar um determinado ponto uma única vez, enquanto no recorrent o mesmo ponto tem que ser visitado várias vezes.

O foraging que ocorre nos jogos RTS, e também o usado na competição, pode ser descrito como Multi Angent Central Place Recorrent Foraging. Multi Agent, pois o fato de serem varios agentes é crucial para a determinação da solucao, pois é preciso fazer o a coordenação entre eles, fazendo que o trabalho do grupo seja maior que a soma do trabalho das partes. Central Place, pois os recursos coletados deve ser levados para um ponto central e lá serem depositados. Recorrent, porque o agente precisa ir na mina varias vezes coletar recursos se abastecer e levar para a base, até que a mina se esgote.

Na competição do AIIDE 2006, os agentes também possuiam informação perfeita sobre o mundo, onde está cada mina os obstáculos, a localização do Control Center e também a localização de cada um dos outros agentes.

4.2 ESTADO DA ARTE

Existem várias técnicas de foraging, cada uma com suas particularidades, e consequentemente vantagens e desvantagens, nesta seção iremos apresentar algumas delas depois iremos classificar elas segundo o problema Multi Angent Central Place Recorrent Foraging.

4.2.1 COOPERATIVA DE LIMPEZA: UM ESTUDO SOBRE ROBÓTICA DE FORMIGAS

O problema abordado nesse trabalho foi Multi Agent Various Place One Shot Foraging. O objetivo de tal trabalho foi implementar um grupo de robôs capazes de limpar uma sala, e tal solução possui algumas restrições, como cobertura total, ou seja, toda a sala precisa ser limpa, e tolerante a falha, se algum dos robos falhar os outros tem que prosseguir a limpeza e completar a missão.[Wagner e Bruckstein 1995].

Na natureza, é muito comum animais que trabalham em sociedade, como formigas, abelhas e passaros, agirem de modo cooperativo para atingirem um objetivo comum e conseguirem performances supreendentes. O artigo sugere uma abordagem semelhante, para coordenar um grupo de robôs sem a supervisao central, utilizando apenas interações locais entre os robôs. Quando essa abordagem descentralizada é utilizada, o grande overhead de comunicação (caracteristicos de sistemas centralizados) não acontece, podendo assim aumentar a simplicidade dos agentes.

Para tal objetivo, é considerado toda a sala como um grafo completo, em que cada nó do grafo representa uma localidade da sala que está inicialmente toda suja, e cada nó é ligado aos nós que representam as áreas vizinhas. Todos os nós são marcados inicialmente como sujos, e em cada área que é feita a limpeza, o nó que a representa é retirado do gafo.

No algoritmo básico de funcionamento busca-se preservar a conectividade desse grafo das posições sujas, permitindo o agente limpar apenas os pontos que não são críticos, que significa dizer que o agente não pode limpar nós que vão dividir o grafo de posições sujas. Isso garante que os robôs só vão parar quando completarem a sua mssão. Uma vantagem importante dessa abordagem, além da simplicidade dos agentes é a tolerância a falha. Mesmo que muitos agentes parem de trabalhar antes de terminar a limpeza, os que continuaram trabalhando vão conseguir terminar a limpeza. Abaixo segue uma figura mostrando a limpeza através de várias interações.

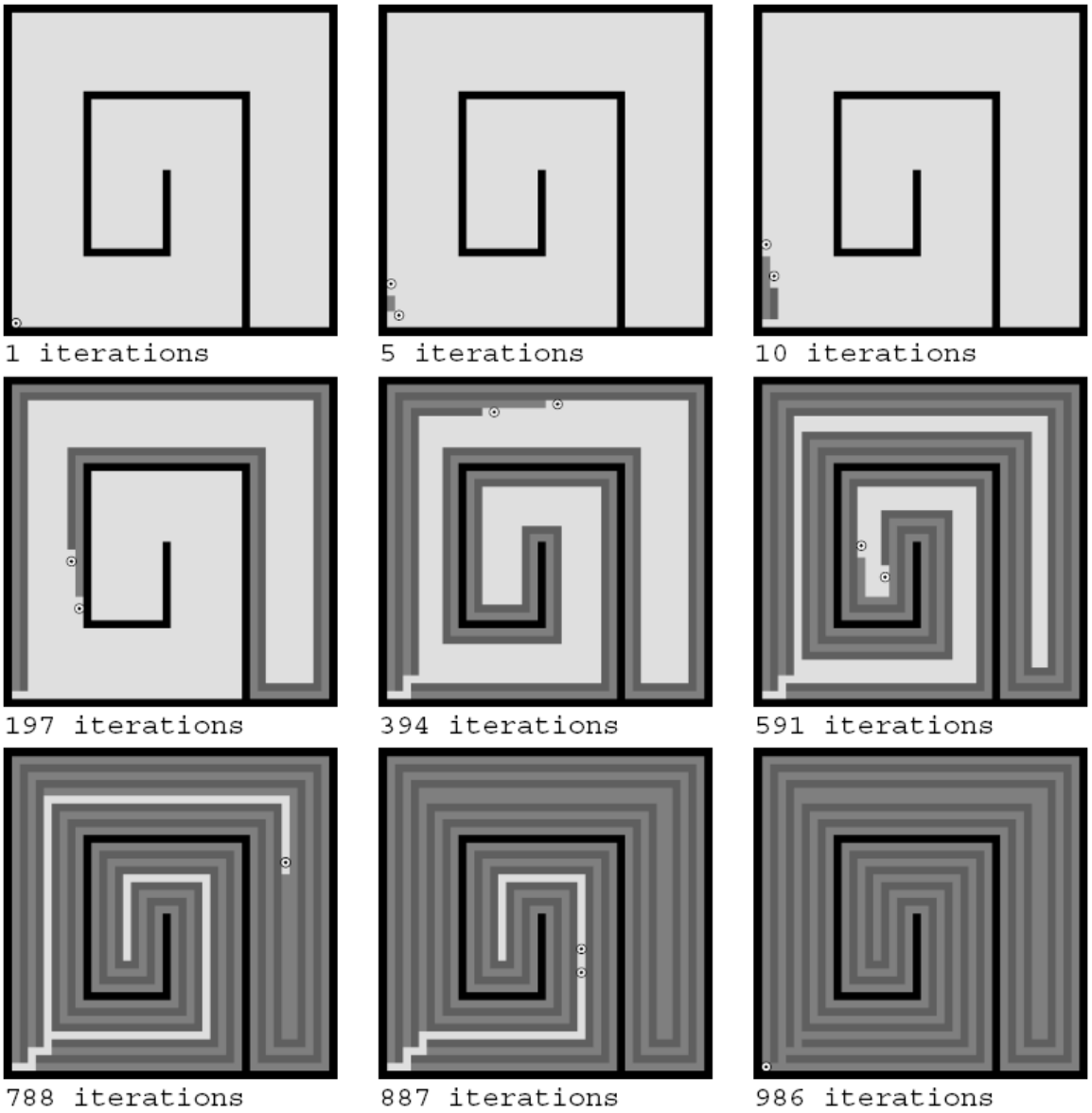

Ilustração 4 - Cooperative Cleaners

4.2.2 DE AÇÕES LOCAIS Á TAREFAS GLOBAIS: ESTIGMERGIA E ROBÔS COLETORES

O foraging abordado nesse trabalho foi Multi Agent Various Place One Shot Foraging Estigmergia é um método de comunicação em que as partes individuais do sistema se comunicam entre si, por meio de modificações feitas no ambiente. O principio da estigmergia aparece no comportamento de carregamento de corpos em colônias de formigas. Observações comprovaram que estes insetos tendem a agrupar corpos de formigas mortas, em cemitérios que ficam em lugares afastados do formigueiro, que acabam crescendo com o tempo e consequentemente vão ficar proximos do formigueiro. Se uma grande quantidade de corpos estiverem proximos ao formigueiro, as formigas deste formigueiro vão pegar esses corpos, carregar por um tempo e soltar em outro lugar, em pouco tempo vai ser visto que os corpos estão sendo agrupados em pequenos clusters, e ao longo do tempo a quantidade destes clusters vão diminuindo enquanto o tamanho dos restantes vão aumentando, eventualmente até que todos os corpos vão estar agrupados em um ou dois grandes clusters, afastados do formigueiro. [Beckers et al. 1994]

O objetivo do trabalho de Beckers e seus colegas foi desenvolver um sistema utilizando vários robos para coletarem um conjunto de objetos espalhados por uma sala, e formar um único cluster com todos eles, semelhante com o cemitério das formigas. Os robos sao equipados com dois sensores de infra vermelho para evitar obstaculos e um sensor que ativa quando uma certa quantidade de objetos é carregada. Os robos possuem apenas tres comportamentos, e apenas um esta ativo em um dado momento. Quando nenhum sensor esta ativo o robo executa o comportamento padrao de se mover sempre em frente até que se encontre um obstaculo ou o sensor de objetos ative. Quando os sensores infra vermelho detectam um obstaculo, o robo gira pra uma direcao randomica, e volta a andar em linha reta na nova direção. Se ele estiver carregando objetos no momento de girar ele consegue carregar e levar o objeto na nova direcao. Quando o robo esta carregando tres ou mais objetos um sensor é ativado e o robo solta os objetos e caminha em outra direção. Esse procedimento é repetido até que todos os objetos estejam reunidos em uma única pilha.Na figura abaixo é mostrado o tempo que levou para formar uma única pilha de com o uso de 1 à 5 robôs.

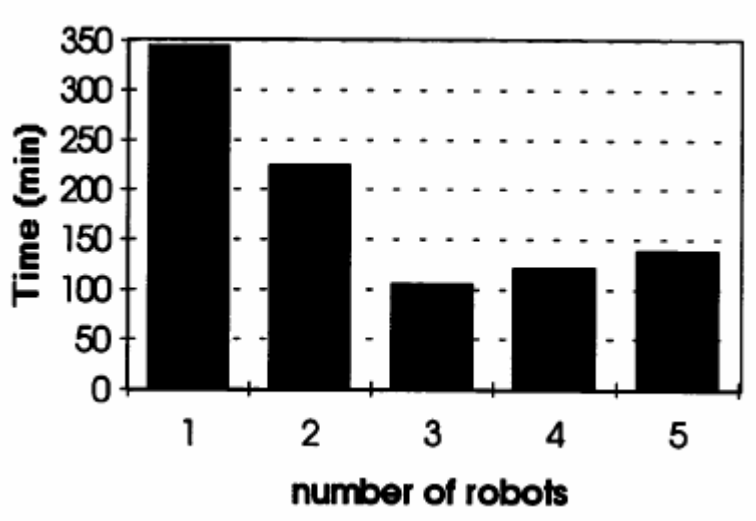

Ilustração 5 - From Local Actions to Global Tasks

4.2.3 DIVISÃO TERRITORIAL DE TAREFAS

A classe do foraging aboradado por Fontan e Mataric é o Multi Agent Central Place One Shot Foraging. O objetivo deste trabalho foi construir um grupo de robôs que vão coletar uma certa quantidade de objetos espalhados em uma sala e levar para um local determinado previamente. [Fontan e Mataric 1998]

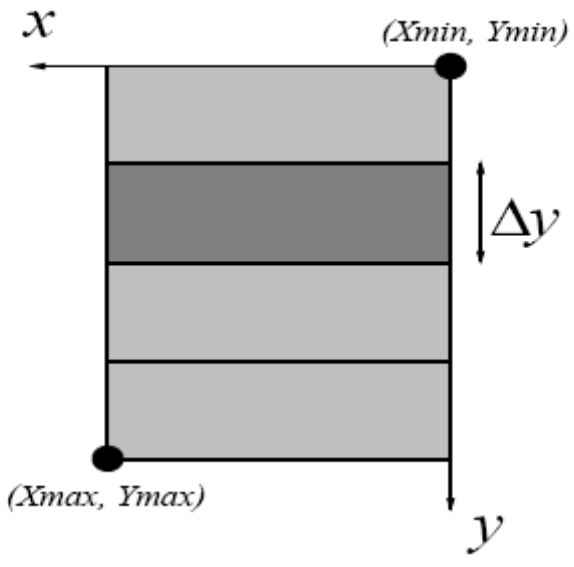

Ilustração 6 - Territorial MultiTask Division 1

O espaço onde vai ser feito a coleta (R) é definido como uma área retangular definida pelas coordenadas *(Xmin, Ymin)* e (Xmax, Ymax), como mostra a figura acima . Cada robo tem uma área de trabalho diferente e é conhecida por cada um a priori, e R é dividido em areas retangulares com alturas e larguras iguais. A largura da região é definido como |*Xmax*

- *Xmin*| , e independe da quantidade de robôs que participam da tarefa. A altura (Δy) depende da quantidade de robos e é definida como:

$$\Delta y = \frac{|\,\mathrm{Xmax} - \mathrm{Xmin}\,|}{\mathrm{Quantidade\ de\ robôs}}$$

A área de trabalho (WA) de cada robo (Ri) é definida como a regiao onde ele procura por objetos (se move de maneira randomica até encontrar os objetos), coleta eles e entrega na área de depósito.

WA(Ri) = {(Xmin, (i)Δy) , (Xmax ,(i+1) Δy) }

A area de deposito (DA) é definida como a regiao onde o robo (Ri), entraga os objetos coletados. A área de depósito final onde todos os objetos devem estar no final é definida como (a).

DA(Ri) = {(Xmin, (i-1)Δy) , (Xmax ,(i) Δy) } ; i>0

DA(Ri) = {(Xmin, Ymin) , (Xmin - a, Ymin -a) } ; i=0

Dessa maneira a área de depósito do robô (Ri) é igual a área de trabalho do robô anterior (Ri-1). Assim cada robo coleta os objetos de uma área e passa para o próximo e assim sucessivamente com mostra a figura abaixo.

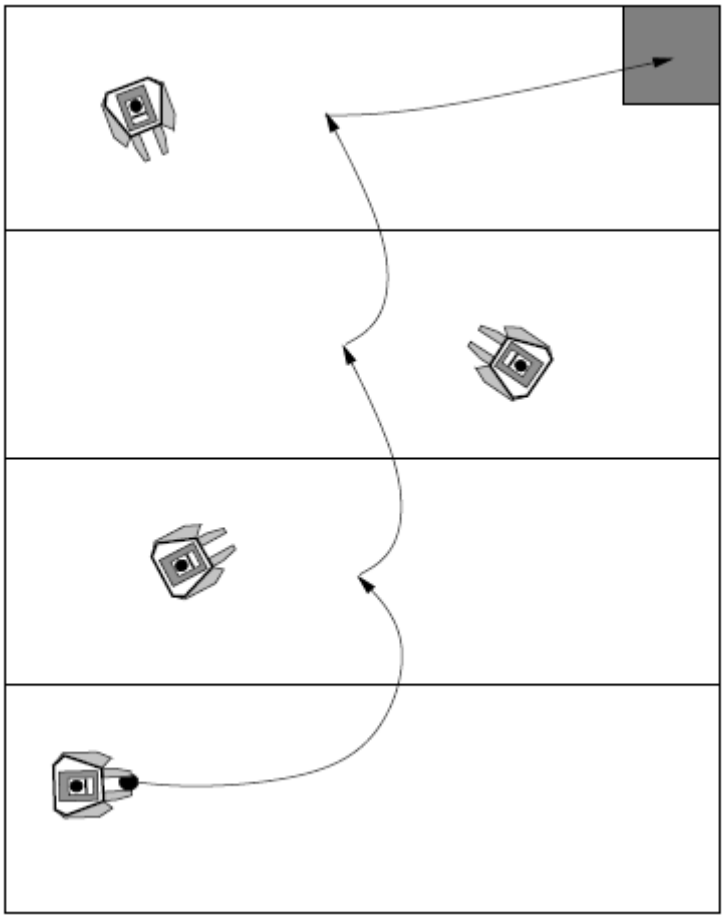

Ilustração 7 - Territorial MultiTask Division 2

4.2.4 COOPERAÇÃO ENTRE AGENTES DISTRIBUÍDOS ATRAVÉS DE AUTO ORGANIZAÇÃO.

Nesse artigo a classe do foraging é Multi Agent Central Place Recorrent Foraging. Este artigo fala sobre cooperação entre agentes distribuidos e tem o seguinte caso de estudo [Steels 1990]. O objetivo é explorar um planeta distante, coletar amostras de uma especie particular de pedra preciosa. A localização das amostras são desconhecidas, mas elas ficam tipicamente agrupadas em certos pontos. Existe um veiculo que pode dirigir pelo planeta e depois voltar para a nave para retornar a terra. Não existem mapas do planeta embora se saiba que o terreno é cheio de obstaculos. Para realizar a coleta serao utilizados um grupo de robos menores que irão fazer a coleta e levaram as amostras para o veiculo principal.

Os robos foram desenvolvidos de maneira incremental, no primeiro ciclo de desenvolvimento foram implementados tres comportamentos:

- Comportamento de Movimentação: Escolha uma direção randomica e siga em frente nessa direção.
- Comportamento de coleta: Se encontrar uma amostra e nao estiver carregando nenhuma, colete. Se encontrar o veiculo principal e estiver carregando uma amostra, solte ela no veiculo.
- Tratamento de colisao: Se encontrar um obstaculo, faça uma curva em direção randômica.

Estes comportamentos básicos resolvem o problema, porém pode demorar muito tempo para os robos encontrarem as amostras e retornarem ao veiculo principal.

Em um segundo ciclo de desenvolvimento o veiculo principal produz um gradiente no ambiente, que é mais forte proximo dele, e vai diminuindo sua forca a medida que se afasta. Agora o comportamento de coleta é dividido em outros dois comportamentos.

- Comportamento de busca: Ele executa este comportamento quando nao esta carregando nenhuma amostra. Ele se movimenta para onde o gradiente é mais fraco, procurando amostras e explorando o mundo.
- Comportamento de retorno: Ele executa este comportamento quando encontra alguma amostra e a coleta. Ele se movimenta para onde o gradiente é mais forte, indo em direção ao veiculo principal.

No terceiro ciclo de desenvolvimento, o foco foi auto organização entre os agentes. Os robos vao fazer a comunicacao entre si através de marcas deixadas no ambiente, migalhas.

Agora quando eles estão carregando uma amostra eles despejam duas migalhas, Se eu nao estou carregando amostra e eu encontro migalhas eu coleto uma delas e movo em

direcao ao lugar onde existe uma maior concentracao de migalhas. Desta maneira os robos ajudam uns aos outros a encontrar o local onde estão as amostras.

4.2.5 OPTIMAL FORAGING THEORY

Essa é uma teoria mais geral, que não fala de um problema e, particular, ela apenas descreve o comportamento, apesar de não ter sido feita nenhuma aplicação que a utilize ela pode ser adequada á diferentes tipos de foraging. Essa teoria fala sobre os animais na natureza que fazem foraging, e que devido a evolução eles o fazem de maneira excelente, e possuem um comportamento até certo ponto previsível. Segundo a teoria os animais quando vão a busca de recurso buscam maximizar a seguinte equação.

$$\frac{E}{h+s}$$

Na equação acima o E representa a quantidade de recurso que vai ser levado de volta ao ninho, h representa o tempo levado para ir até o recurso captura-lo e retornar ao ninho, e a variável s representa o tempo de busca do predador pelo recurso. [MacArthur e Pianka 1966]

4.3 CONCLUSÃO

Nesta seção iremos avaliar as técnicas aboradas na seção anterior de acordo com os seguintes quesitos, Multi Angent (se é possível aplicar a solução a um ambiente com vário agentes), Central Place (Existe um lugar central para o qual os recursos são levados), Recorrent (Ele precisa ir no recurso mais de uma vez até esgotá-lo) , respeito as limitações do jogo e Informação Perfeita. A seguir é mostrada uma tabela comparativa entre as diferentes soluções apresentadas anteriormente e quantos requisitos cada solução atendeu.

	Multi Agent	Central Place	Recorrent	Perfect Information	Limitações do jogo	Total
[Wagner e Bruckstein 1995]	●			●		2
[Beckers et al. 1994]	●					1
[Fontan e Mataric 1998]	●	●				2
[Steels 1990]	●	●	●			3
[MacArthur e Pianka 1966]	●	●	●	●	●	5

Tabela 3 - Análise das soluções de Foraging

Como mostra o resultado a solução 5 mesmo não sendo focado na Inteligencia Artificial, ela resolver melhor o problema, pois ela é a que atende a maior quantidade de requisitos, e através dela vai ser desenvolvida uma solução para o problema.

CAPITULO 5

ABORDAGEM DE COLETA PROPOSTA

O agente a ser desenvolvido precisa respeitar todos os quesitos, no qual foram avaliados as outras soluções, apresentadas no capitulo anterior. Ele precisa agir de maneira autônoma, se um agente deixar de existir os outros devem continuar a tarefa, devido ao fato que é um jogo RTS, poderiam existir inimigos que poderiam matar os agentes. Se houvesse alguma espécie de processamento centralizado em algum agente, se esse agente fosse destruído o processamento ficaria comprometido. A solução encontrada deve conseguir ter um bom desempenho no jogo 1 que foi mostrado no capitulo anterior.

5.1 A ABORDAGEM

A inteligência do agente se baseou na Optimal foraging theory, onde o objetivo de cada agente é coletar os recursos mais próximos a ele e carregar sempre a maior quantidade de recursos possível em cada viagem.

Os agentes possuem 5 estados que serão explicados a seguir:

- EVADE: É o estado inicial dos agentes, nele os agentes se movem aleatoriamente. Ele foi criado porque os agentes no início do jogo estão todos agrupados então alguns agentes estão presos pelos agentes mais externos impossibilitados de se mover. Ele passa alguns segundos neste estado para os agentes se espalharem mais, o suficiente para a maioria dos agentes poderem se mover em seguida seu estado é setado como IDLE.
- IDLE: Neste estado o agente procura um local para minerar (um lugar ao lado de uma mina) o mais próximo dele possível. Quando ele escolhe o local, ele define a sua rota e seu estado é setado para GOING TO MINE.
- GOING TO MINE: Neste estado ele vai percorrer seu caminho até chegar ao seu destino. Quando chegar ao local seu estado vai ser setado como MINING.
- MINING: Neste estado ele vai minerar até a mina esvair, ou até ele coletar o máximo de recursos que ele pode carregar. Se ele coletar o máximo de recurso possível seu estado é setado como GOING TO DELIVER, se a mina esvair antes dele preencher sua capacidade máxima ele vai para o estado IDLE.
- GOING TO DELIVER: Neste estado o agente vai calcular sua rota até o control center e vai seguir essa rota até o final, ao chegar ao control center o agente

deposita todo o recurso que ele esta carregando. E em seguida seu estado é setado como IDLE.

Abaixo podemos ver uma máquina de estados finita que representa os estados dos agentes e suas transições:

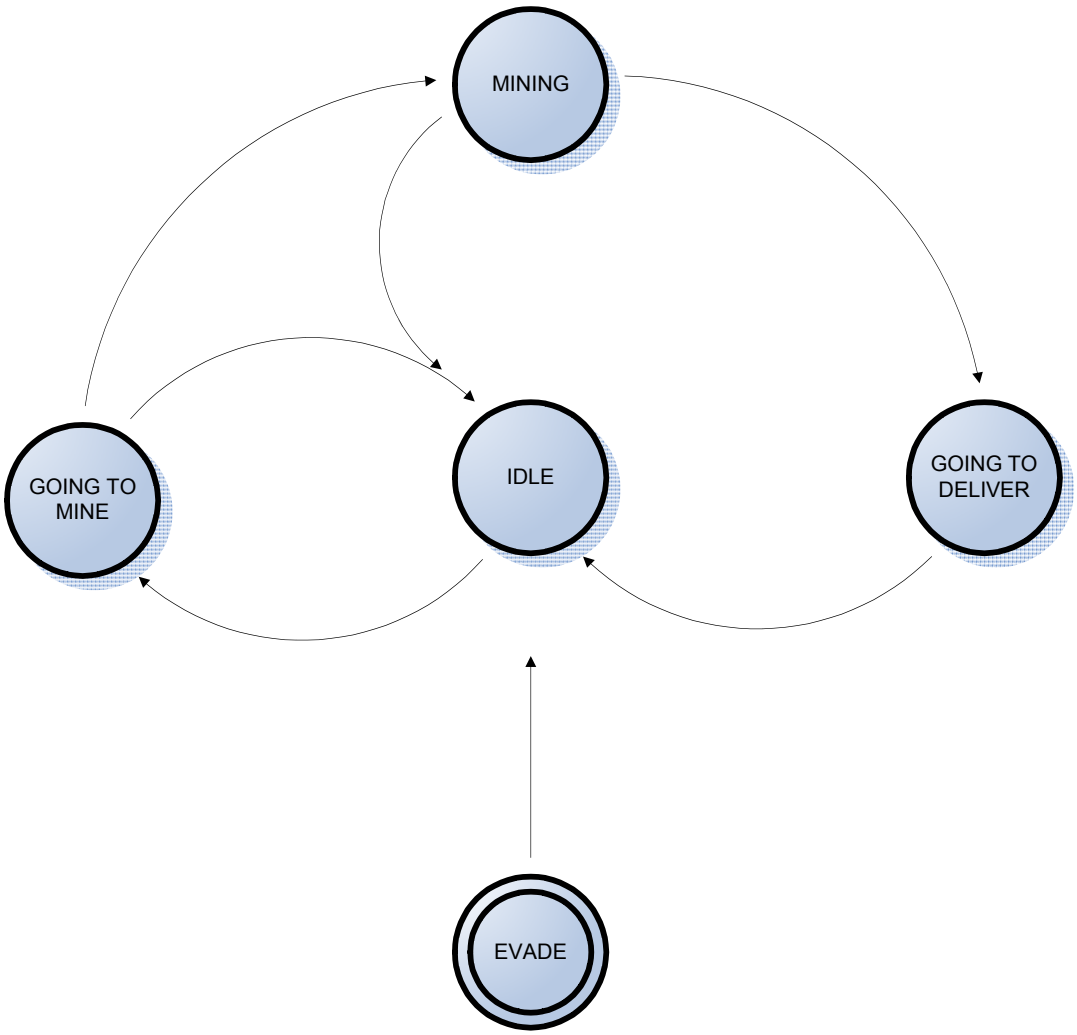

Ilustração 8 - Maquina de Estados do Agente

Para faciltar a decisão de que local ir para minerar, decisão tomada no estado IDLE, existe uma lista, acessível por todos os agentes, com todas as posições de mineração (uma posição é considerada de mineração se ela for vizinha a uma mina) e o tempo que vai demorar para ele ficar livre. Para localizar o ponto de mineração mais próximo do seu ponto atual, ele checa se existe algum ponto de mineracao entre os tiles que estão a distância de 1, se não encontrar ele procura em uma distância de 2, e assim sucessivamente até encontrar. A figura a seguir mostra um worker e sua distância para os outros tiles

3	**3**	**3**	**3**	**3**	**3**	**3**
3	**2**	**2**	**2**	**2**	**2**	**3**
3	**2**	**1**	**1**	**1**	**2**	**3**
3	**2**	**1**	**Worker**	**1**	**2**	**3**
3	**2**	**1**	**1**	**1**	**2**	**3**
3	**2**	**2**	**2**	**2**	**2**	**3**
3	**3**	**3**	**3**	**3**	**3**	**3**

Ilustração 9- Distância entre o tile e o trabalhador

Não existe comunicação direta ou explícita entre os agentes, quando um agente encontra o ponto de mineração mais próximo dele, ele calcula a rota para o ponto, e verifica se o tempo necessário para aquela mina ficar livre é menor que o tamanho de sua rota, se não for, ele procura o próximo ponto de mineração mais próximo, mas se for menor, o que significa que no momento que ele chegar no ponto ele ja estará livre, ele reserva aquele ponto na lista e diz quanto tempo vai demorar para liberá-lo. Dessa maneira os outros agentes ficam sabendo quais pontos estão reservado e quanto tempo vai demorar para ficar livre.

5.2 IMPLEMENTAÇÃO

A ferramenta escolhida para o desenvolvimento e aplicação da solução foi o JARTS (Java Real Time Strategy Simulator), os motivos são explicitados no capítulo sobre simuladores deste trabalho. O desenvolvimento do agente ocorreu de maneira incremental, a cada ciclo de desenvolvimento melhoramentos foram implementados e seus resultados foram comparados com o agente anterior.

A abordagem proposta faz o uso de 4 classes: jcmjWorker, jcmjPathFinding, jcmjNode e jcmjPoint. A classe jcmjWorker, é o agente em si, ele estende a clase Worker do JARTS, ferramenta escolhida para implementação da solução. A classe jcmjPathFinding, é responsável por encontrar as rotas dos agentes. Foi implementado utilizando o A* (A-Star). Ele utiliza a classe jcmjNode como classe auxiliar, e quando a rota é encontrada ele monta um Vector de objetos do tipo jcmjPoint, indicando a rota. A jcmjPoint é uma classe que representa um ponto em um espaço 2D.

PRIMEIRO CICLO:

No primeiro ciclo os comportamentos dos agentes foram implementados de maneira básica. No estado IDLE, ele procura o local vizinho a uma mina o mais perto dele possível, sem procurar saber se existe alguém indo para o mesmo ponto.

Quando os agentes colidem não há qualquer tipo de negociação os dos ficam parados até que um desbloqueie o caminho do outro.

SEGUNDO CICLO:

A melhoria feita no segundo ciclo, foi a reserva de local de mineração (local ao lado de uma mina), onde cada agente reserva o local onde ele vai minerar, no momento de calcular a rota e coloca o tempo que ele vai demorar para liberar aquele ponto. Com isso na hora de selecionar a sua rota o agente vai procurar a rota mais próxima a ele, que vai estar livre no momento em que ele chegar ao ponto de mineração. Desta maneira evitou-se de ir mais de um agente para a mesma mina.

TERCEIRO CICLO

No terceiro ciclo foi implementado um sistema de colisão quando dois agentes colidem ambos esperam 5 ciclos, para saber se um esta apenas passando pelo outro ou se está colidindo mesmo, e se ao final do quinto ciclo eles ainda estiverem sendo bloqueados suas rotas são recalculadas.

QUARTO CICLO

No quarto ciclo o pathfinding foi melhorado de maneira a ele procurar na lista aberta e na lista fechada em um mesmo laço, diferente dos anteriores em que essas checagens eram feitas em laços diferentes.

CAPITULO 6

JARTS

6.1 O QUE É O JARTS

O JARTS (acrônimo para Java Real Time Strategy Simulator), é um simulador de jogos RTS, voltado para pesquisadores na área de Inteligecia Artificial. Nele o usuário pode focar-se no problema e desenvolver apenas o comportamento dos agentes uma vez que o simulador se encarrega de todo o resto.

Seu funcionamento foi baseado no RoboCode (Simulador de batalha entre tanques), no qual existe uma classe básica com as principais ações dos agentes, como minerar, mover e atirar. Ficando a cargo do desenvolvedor estender essas classes, para criar comportamento mais complexos para seus agentes. Do mesmo modo que o Robocode, no JARTS, o desenvolvedor indica qual é a classe que estendeu e o simulador se encarrega de todo o resto.

São três as classes que o usuário pode estender, e elas serão explicadas a seguir:

- Worker – Representa um trabalhador, seu objetivo é coletar recursos, e abastecer a base, o Control Center. Suas ações básicas são as seguintes, Move (o agente se move para um tile desde que ele seja vizinho ao tile atual em que ele está), Mine (O agente coleta o recurso, desde que ele esteja no tile vizinho ao da mina que ele deseja minerar), Deliver (o agente deposita os recursos que ele está carregando no Control Center, ele precisa estar próximo ao Control Center) e Idle(o agente não executa nenhuma ação).

- Tank – Simula um tanque de guerra, é a unidade militar, e responsável pelo combate com o inimigo. Ele possui três ações básicas, Move, Idle (ambas semelhantes ao do agente Worker), e Shoot (o agente atira em uma outra unidade, o dano é proporcional a distância entre o tanque e o alvo).

- Control Center – Representa a base, nele os Worker depositam os recursos, e a missão dos Tanks é protegê-lo caso seja do seu proprio time ou ataca-lo caso seja o Control Center inimigo.

Da mesma maneira do ORTS, o JARTS possui categorias de jogos, e nele o usuário pode testar o comportamento de seu agente de maneira isolada, sem se preocupar com todas as influencias de um jogo de RTS. Ele pode simular um jogo semelhante ao jogo 1 do ORTS, no qual só existe um control center e vários workers de um mesmo time, onde ele pode testar o comportamento de coleta de recurso sem a interferencia de inimigos, que existiriria em um jogo completo de RTS. Existe também a possibilidade do usuário querer testar apenas a capacidade de combate de seus tanques, então ele pode simular um jogo semelhante ao jogo 2 do ORTS, no qual o foco é a batalha e ele não precisa se preocupar com coleta ou gerenciamento de recursos.

6.2 MODELAGEM E IMPLEMENTAÇÃO

A modelagem e implementação do JARTS, foram feitas de maneira a resolver os problemas que os outros simuladores disponíveis possuiam, e aproveitar os pontos fortes dos simuladores. Como foi mostrado anteriormente ele tem modos de jogo, em que cada tipo de agente pode ser testado em sua função isoladamente, semelhante ao ORTS.

Mesmo não sendo uma linguagem muito difundida para jogos, decidiu-se utilizar Java para a implementação do simulador, pois ela é uma linguagem independente de plataforma, permitindo que qualquer usuário, faça usufruto do simulador, independente do sistema operacional usado.

A modelagem do JARTS, busca uma arquitetura simples e intuitiva que visa diminuir o tempo de máximo de aprendizagem. Tanto a arquitetura quanto o modo de simulação se assemelham ao RoboCode, visto que esse é um simulador muito utilizado e de facil aprendizagem.

A seguir vamos apresentar a arquitetura básica do JARTS:

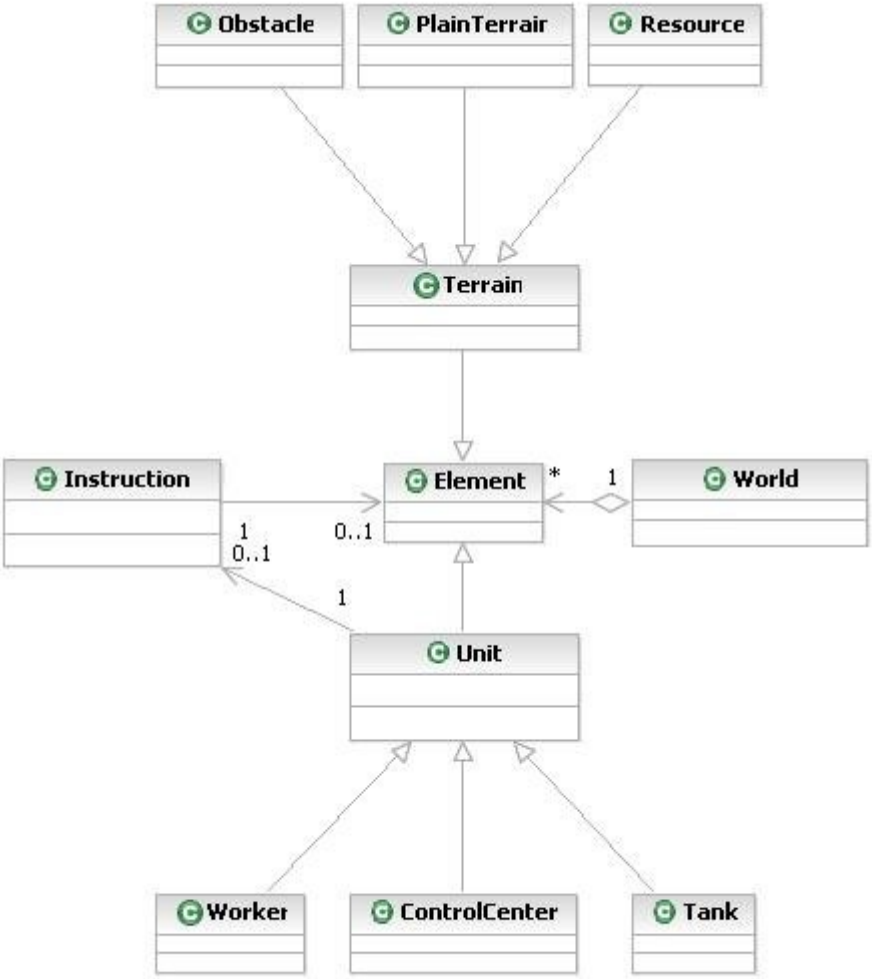

Ilustração 10 - Diagrama de classes JARTS

A classe World possui uma representação do mundo, formado por uma matriz de Elements, que podem ser elementos de dois tipos:

- Unit – A classe que representa as unidades controladas pelo usuário, como Worker, Control Center ou Tank.
- Terrain – Uma classe que representa os elementos de terreno, podendo ser de três tipos, Obstacle, que representa um obstáculo, ou seja, as unidades não podem passar por aquele lugar, Resource, que são os recursos que devem ser coletados pelos Workers, eles também não permitem a passagem de unidades, e Plain Terrain, que é um tile por onde as unidades podem passar.

Como mostra a arquitetura, cada objeto do tipo Unit possui um Instruction, que por sua vez faz referencia a um objeto do tipo Element. O objeto do tipo Instruction indica qual a

ação que o agente deve executar, e o objeto do tipo Element que ele referencia é o alvo da ação. Por exemplo, um Worker pode ter uma Instruction que indica que ele deve minerar, então o Element que ele deve refereciar é o Resource que o Worker deve minerar. Se a ação fosse Move, o Element deveria ser do tipo Plain Terrain, que indicaria para qual tile ele deveria ir.

6.3 FAZENDO SIMULAÇÃO

A implementação dos agentes se assemelha a do RoboCode, o usuário estende apenas a classe que ele deseja implementar. Se ele deseja criar um agente MyWorker, do tipo Worker, ele precisa apenas esteder a classe Worker e implementar as suas ações. Para isso ele só necessita implementar um método o updateAction() do seu Worker, que vai ser chamado a cada ciclo do jogo, e dentro do método definir qual ação o seu agente deve executar escolhendo um dos métodos da classe que o trabalhador estendeu (Worker). Se ele deseja minerar ele deve utilizar dentro do seu Worker a seguinte chamada this.mine(target), onde target deve ser um objeto do tipo Resource. Caso ele deseja se mover, ele deve utilizar o método this.move(x,y), passando como parâmetro dois inteiros, representando a coordenada destino que ele deseja ir. A outra opção de ação do agente é depositar os recurso no Control Center, para isso ele deve chamar o método this.deliver().

Depois através da interface do programa ele carrega a classe que representa o seu Worker, MyWorker, como mostra a imagem abaixo. Nesta tela ele escolhe quais foram as unidades que ele estendeu e ele deseja carregar no novo jogo.

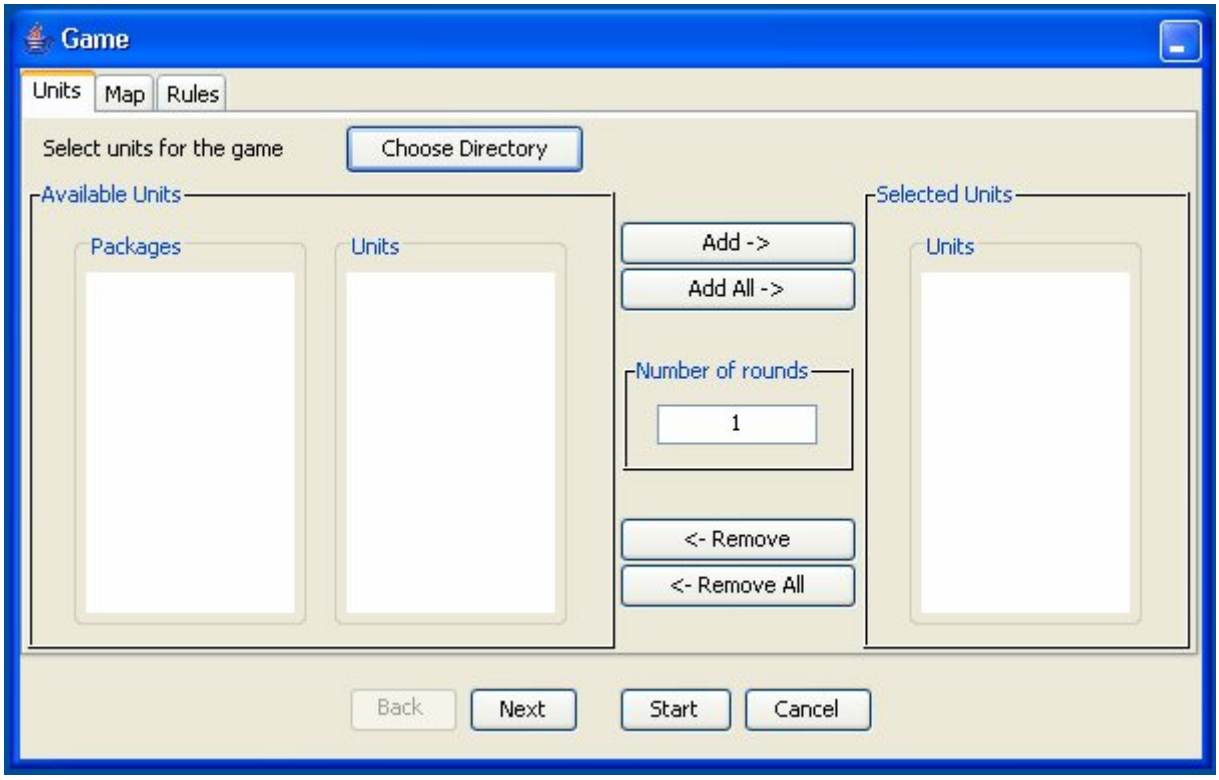

Ilustração 11 - JARTS 1

.

Depois de escolher as unidades que ele vai carregar, ele deve escolher entre 11 mapas pre-definidos qual deles ele deseja utilizar para fazer a simulação, na tela abaixo:

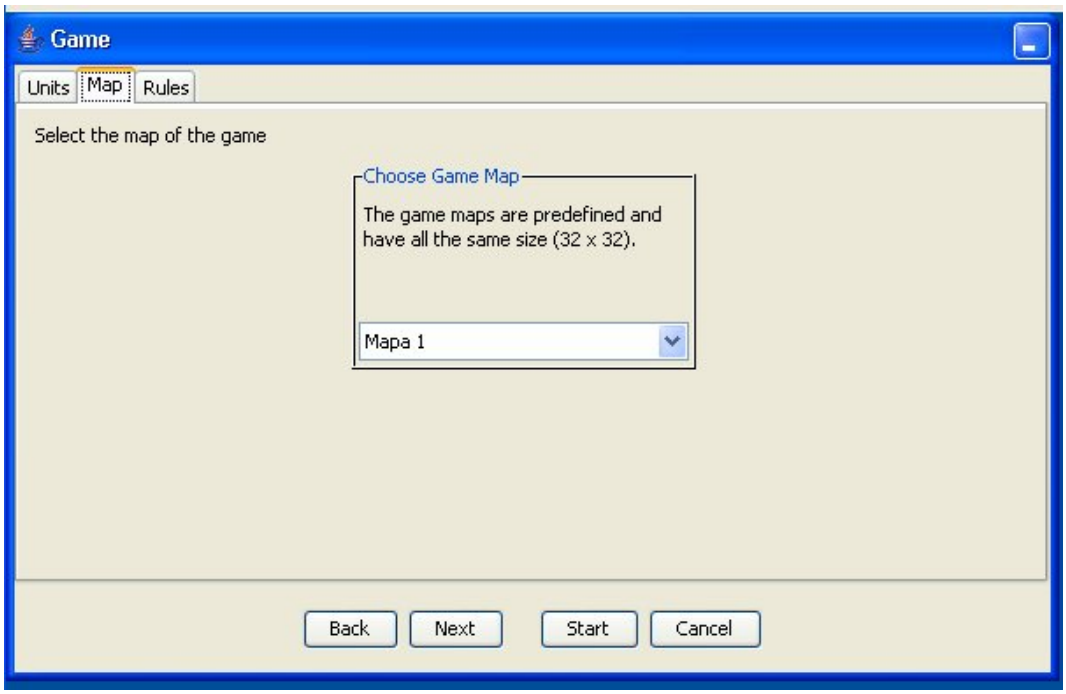

Ilustração 12- JARTS 2

Por último o usuário escolhe que tipo de jogo ele deseja disputar, o jogo tipo 1 (coleta de recursos) ou o jogo tipo 2 (combate), semelhante ao ORTS, na mesma tela ele ainda faz a escolha do tempo total da simulação e o tempo que deve durar cada ciclo, como mostra a figura a seguir:

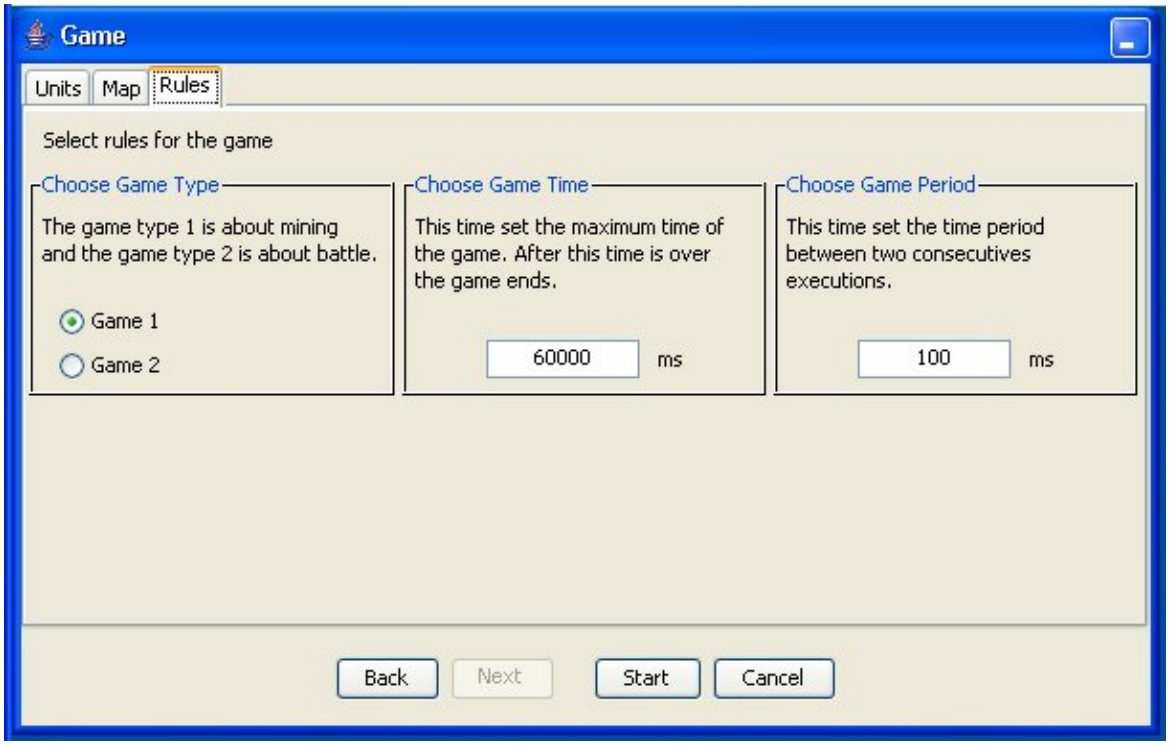

Ilustração 13 - JARTS 3

A tela de simulação do jogo tipo 1 é mostrada a seguir, nela os quadrados cinzas representam os obstacúlos, os quadrados pretos representam os terrenos planos onde o agente pode passar, os círculos verdes representam os trabalhadores, os triangulos dourados representam as minas e o quadrado de bordas azuis representam o Control Center.

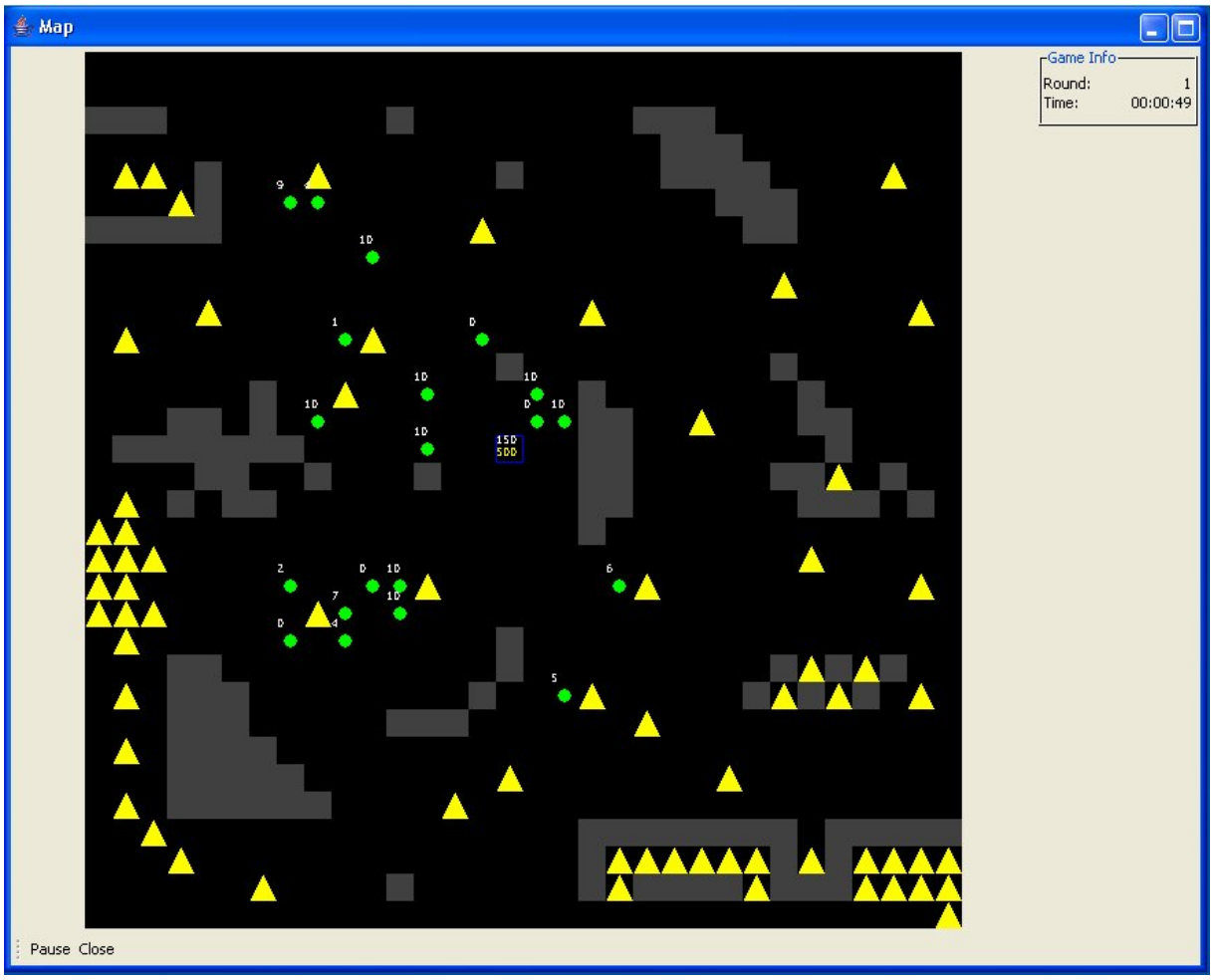

Ilustração 14 - JARTS 4

Ao lado de cada trabalhador há um numero indicando a quantidade de recursos que ele está carregando naquele momento. A quantidade total de recursos coletados é indicado no Control Center, no lado esquerdo da simulação temos a informação sobre o tempo restante e o Round que está acontecendo.

CAPITULO 7

RESULTADOS

REFERÊNCIAS

[Beckers et al. 1994] R. Beckers, O.E. Holland, J.L. Deneubourg, From local actions to global tasks: Stigmergy and collective robotics, in: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems Artificial Life IV, 1994, pp. 181–189.

[Fontan e Mataric 1998] M. Fontan and M. Mataric. " Territorial Multi-Robot Task Division." IEEE Transactions on Robotics and Automation, 14:5, 1998.

[Wagner e Bruckstein 1995] Wagner, I. A., and Bruckstein A. M. 1995. Cooperative cleaners: A study in ant-robotics. Center for Intelligent Systems, Technion, Haifa.

[Steels 1990] Steels, L. (1990). Cooperation between distributed agents through self organization. In Demazeau, Y. and Müller, J.-P., editors, Decentralized AI — Proceedings of the First European Workshop on Modelling Autonomous Agents in Multi-Agent Worlds (MAAMAW- 89), pages 175–196. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.

[MacArthur e Pianka 1966] Optimal foraging Theory
Site: http://en.wikipedia.org/wiki/Optimal_foraging_theory
Visitado em 25/9/2006

[ORTS AIIDE 2006] ORTS RTS Game AI Competition
Site: http://www.cs.ualberta.ca/~mburo/orts/AIIDE06/index.html
Visitado em 25/9/2006

[ORTS] ORTS – Open Real Time Strategy
Site: http://www.cs.ualberta.ca/~mburo/orts/
Visitado em 25/9/2006

[STRATAGUS] Stratagus – Real Time Strategy Engine
Site: http://stratagus.sourceforge.net/
Visitado em 25/9/2006

[GLEST] Glest - Wikipedia
Site: http://en.wikipedia.org/wiki/Glest
Visitado em 25/9/2006

[BOSON] BOSON
Site: http://boson.eu.org/
Visitado em 26/9/2006

[RTS- Wikipedia] Real-time strategy - Wikipedia, the free encyclopedia -
Site: en.wikipedia.org/-wiki/Real-time_strategy
Visitado em 29/8/2006

[Davis 1999] Davis, I. 1999. Strategies for Strategy Game AI. In Papers from the AAAI 1999 Spring Symposium on ArtificialIntelligence and Computer Games, Technical Report SS- 99-02, 24-27. AAAI Press.