Universidade Federal de Pernambuco

Graduação em Ciência da Computação
Centro de Informática

Um Estudo de Estratégias com Multiagentes para Jogos RTS usando o Simulador RTSCup

Pablo Alessandro Barbosa Viana <pabv@cin.ufpe.br>

Orientador: Geber Lisboa Ramalho <glr@cin.ufpe.br>

Recife, 03 de dezembro de 2008

"How the mind works is still a mystery.

We understand the hardware,

but we don't have a clue about the operating system."

James D. Watson

Agradecimentos

Gostaria de agradecer a todos que de certa forma contribuíram para a conclusão da minha graduação e deste trabalho.

Primeiramente gostaria de agradecer a minha família, por sempre confiar em mim e me incentivar a realizar os meus sonhos. Em especial a minha mãe e ao meu falecido pai, José Alexandre Viana, que sempre me apoiou e onde estiver sei que está muito feliz por esta conquista.

Ao professor Geber Ramalho e a professora Patrícia Tedesco por me darem a oportunidade de realizar este trabalho e pelo incentivo, paciência e compreensão, na elaboração do mesmo.

Aos meus amigos que foram minha força durante todo o período da graduação, Fagner Nascimento, Cássio Melo, Benito Fernandes, Sergio Sette, Victor Costa, Leandro Espíndola, Hallan Cosmo, Augusto Matos (o famoso jamaj!!!!), enfim, toda a galera do counter galáticos (me desculpem os que não foram citados).

Resumo

Jogos de estratégia em tempo real, ou RTS (*Real Time Strategy*), 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 multiagente, *pathfinding*, alocação de recursos, entre outros.

Os objetivos deste trabalho são estudar, analisar, implementar e avaliar estratégias de IA para um jogo RTS, num ambiente multiagente. Visando melhorar estratégias existentes e fazer propostas de novas estratégias.

Palavras-chave: jogos, estratégia em tempo real, *real time strategy*, inteligência artificial, coleta de recursos, *foraging*, *pathfinding*, RTSCup.

Sumário

Lista de Figuras

1 Introdução

Jogos de estratégia em tempo real, ou RTS (*Real Time Strategy*), 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 multiagente, *pathfinding*, 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. [1]

Competições de Inteligência Artificial são uma das melhores formas de estimular estudantes e pesquisadores da área a desenvolver e testar novas técnicas de IA. Muitas dessas competições de Inteligência Artificial utilizam-se de programas para simular o ambiente da competição. [2]

O RTSCup é um ambiente de simulação de jogos de estratégia em tempo real com foco na inteligência artificial.[3] Foi desenvolvido no Centro de Informática da UFPE com o objetivo de ser utilizado pelos alunos para criar e testar novas técnicas de IA, servindo como uma ferramenta para medir o desempenho e comparar os resultados de cada abordagem.

2 Jogos de Estratégia

Nesta seção, será dada uma definição do que são jogos de estratégia com um breve histórico. Depois será esclarecido como esta modalidade de jogo tem ajudado o desenvolvimento de novas técnicas de IA e em que áreas elas se aplicam.

2.1 Definição

Jogos de estratégia são jogos onde a habilidade dos jogadores na tomada de decisão é primordial para determinar o resultado do jogo. [4]

O elemento principal que distingue este tipo de jogo é exatamente a inexistência, ou baixa relevância, de fatores como sorte ou alguma habilidade física, na definição do resultado.

Todos os jogadores possuem o mesmo grau de conhecimento do jogo e as mesmas possibilidades de ações, assim sendo o resultado dependerá majoritariamente da habilidade do jogador, e conseqüentemente, das decisões que tomar contrastadas com a de seus adversários.

Muitos jogos incluem estes elementos em um maior ou menor grau, sendo difícil delimitar o tipo do jogo. É então, muito mais preciso dizer que um jogo possui elementos de estratégia em certo grau do que ele segue um princípio específico desses jogos. Sendo assim podemos considerar a classificação dos jogos num espectro contínuo onde de um lado encontram-se os jogos baseados em sorte e no outro os jogos baseados em habilidade. [4]

2.2 Tipos de Jogos de Estratégia

O xadrez é um dos mais antigos jogos de estratégia e é um ótimo exemplo para ilustrar a definição que foi dada. Ele possui diversas variantes, mas o modelo que conhecemos hoje é datado na segunda metade do século XV.

Figura 2.1 Um tabuleiro de xadrez montado

Na evolução dos jogos de estratégia em tabuleiro, um jogo se destaca por ter obtido um reconhecido sucesso internacional, o jogo WAR (em inglês *Risk*). Apesar dele não figurar puramente como um jogo de estratégia, já que possui um relativo envolvimento do fator sorte, é um jogo sempre lembrado quando pensamos em jogos de estratégia sendo responsável por massificar a associação de jogos de guerra com jogos de estratégia.

Figura 2.2 Tabuleiro do jogo de WAR

No mundo dos videogames, jogos de estratégia não mudam muito de contexto. São aqueles onde se requer raciocínio e planejamento para se obter a vitória. Na maioria, os jogadores possuem uma visão aérea do mundo do jogo controlando assim as unidades que estão sob o seu comando. [5]

Normalmente eles tomam uma das quatro formas, dependendo se ele é em turno ou em tempo real e se ele é focado na tática ou na estratégia. [5]

Jogos em turno são como os jogos de tabuleiro, cada jogador completa a sua jogada e então é passada a vez ao outro para que este execute suas ações, e assim sucessivamente. Em contraste a este estilo aparece o modelo de jogo em tempo real, onde todos os jogadores jogam ao mesmo tempo tornando o jogo mais dinâmico e desafiador.

Jogos de tática são aqueles com o foco no objetivo do jogo, ou seja, conquistar um território, destruir um exército. Jogos de estratégia são mais profundos no sentido que o jogador é responsável por todo um conjunto de micro-atividades, como coletar recursos, fazer construções, pesquisar tecnologias, criar exércitos, e é a habilidade na tomada das decisões, do que fazer a cada momento, para realizar o objetivo designado, que fará o vencedor.

Como pode ser observado, os jogos de tabuleiro mostrados até aqui podem ser classificados como jogos de tática em turno. Exemplos deste tipo de jogo para videogames são a série *Steel Panther*, *Silent Storm*, *X-COM: UFO Defense*, *Final Fantasy Tatics*, entre outros.

Figura 2.3 Jogo *X-COM: UFO Defense*

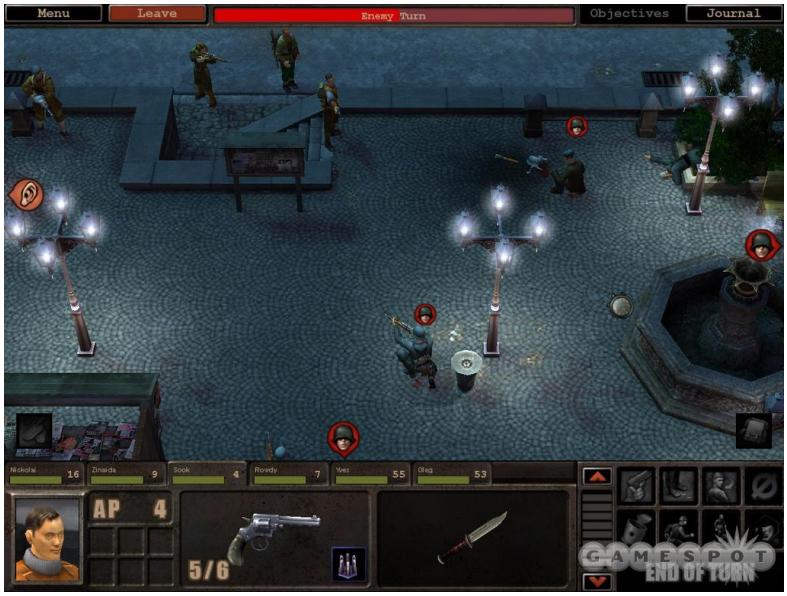


Figura 2.4 Jogo *Silent Storm*

Jogos de estratégia em turno se distinguem por incluir o micro-gerenciamento, onde, dependendo da complexidade do jogo um turno varia de poucos minutos até mesmo algumas horas, parar fazer o gerenciamento completo de todas as ações de suas unidades. Muitas soluções de IA são, hoje em dia, utilizadas para realizar algumas dessas atividades e tornar estes jogos mais dinâmicos. Um dos principais representantes deste tipo de jogo é a série *Civilization*, mas além dele podemos citar as séries *Heroes of Might and Magic, Advance Wars* (NDS).

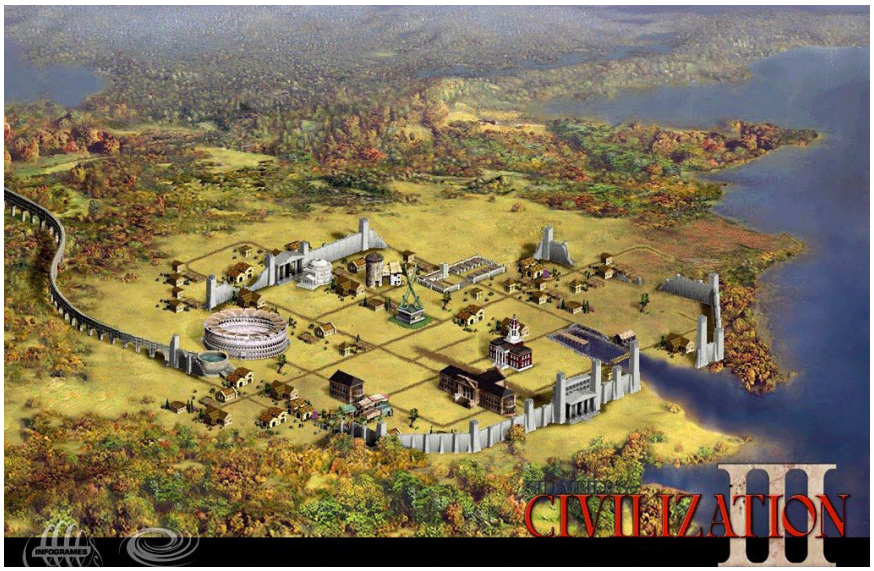


Figura 2.5 Imagem de uma cidade em *Civilization III*

Figura 2.6 *Civilization IV*

Jogos de tática em tempo real são, principalmente, jogos de guerra com o foco no aspecto operacional da batalha, táticas de guerra, posicionamento de tropas. Nesse tipo de jogo é esperado que o jogador complete os objetivos utilizando apenas as unidades de combate que lhe são oferecidas. São jogos deste tipo: *Warhammer*, *Ground Control*, *Close Combat: Modern Tatics*.

Figura 2.7 *Medieval: Total War*, exemplo de jogo RTT (*Real Time Tatics*)

O nome “estratégia em tempo real” intitula os jogos onde a ação do jogo é contínua, não em turnos, e os jogadores terão que tomar suas ações e decisões num mundo em constante mudança. [5] São comuns aos jogos RTS a obtenção de recursos, construção de bases, pesquisa de tecnologias, criação de unidades, além, claro, do controle de suas unidades, enfim, um conjunto de atividades que direcionam o jogador a se preocupar com o gerenciamento e a manutenção de suas unidades e de seus recursos.

Nestes jogos o objetivo dos jogadores é posicionar e gerenciar suas unidades e estruturas para assegurar áreas do mapa e/ou destruir os bens dos seus inimigos. [6]

As tarefas que os jogadores devem realizar num jogo RTS requerem muita habilidade. Complexas interfaces têm evoluído para auxiliar nessas tarefas. Muitas características foram “copiadas” do ambiente de desktop, notavelmente as técnicas de “*clicking and dropping*” e “*clicking and dragging*”, comumente utilizadas, respectivamente, para selecionar um local aonde se fará uma construção e para selecionar várias unidades simultaneamente. [6]

Figura 2.8 *Starcraft*

Embora os jogos RTS tenham uma extensa história, alguns títulos serviram para definir a percepção popular deste gênero, em particular os jogos lançados no período entre 1992 e 1998. [6] Entre esses podemos destacar *Dune II* (considerado por muito o pai dos jogos RTS modernos), *Warcraft*, *Warcraft II*, Starcraft, *Command & Conquer*.

Este gênero teve poucas variações no seu conceito, durante os anos. Basicamente criavam-se mais unidades, mapas maiores, terrenos diferentes, do que inovações de conceito. Uma das grandes mudanças aconteceu na transição de 2D para 3D, os primeiros jogos a contar com essa característica foram *Command & Conquer*, *Populous: The Beginning* e *Homeworld*. Apesar de críticas a esse modelo, como a complexidade do controle de câmera e o posicionamento de objetos, a partir dos anos 2002 esta tecnologia já havia se transformado um padrão nesses jogos.

Figura 2.9 ***Command and Conquer 3***

2.3 Elementos dos Jogos RTS

Como explicitado anteriormente os jogos RTS possuem características marcantes e agora faremos uma análise das principais ações existentes nesses jogos.

2.3.1 Coletar Recursos

Esta tarefa é uma das mais estratégicas do jogo. Nos jogos RTS os jogadores precisam obter recursos para poder criar novas unidades, fazer novas construções, obter novas tecnologias, fortificar seus exércitos e assim obter um poderio suficiente para realizar seus objetivos.

Dependendo do jogo existem tipos diferentes de recursos que são usados para finalidades específicas. Em *Command & Conquer*, por exemplo, existe apenas um tipo de recurso (o minério), em *Warcraft III* existem dois tipos de recursos (madeira e ouro), já em *Age of Empires II* existem quatro tipos de recursos (madeira, comida, ouro e pedra).

Esta tarefa é realizada por unidades específicas, dependendo do jogo, normalmente estas unidades são fracas para batalhar ou simplesmente não lutam. Existem jogos onde os trabalhadores, como comumente são chamados, são especialistas em um tipo específico de coleta e não podem coletar outros tipos de recursos. Mas na maioria os trabalhadores podem coletar qualquer tipo de recurso a que são comandados.

A coleta de recursos é crucial para o desenvolvimento do jogador, mas é de fundamental importância saber realizar as decisões do quanto se focar nesta tarefa visando que seu inimigo pode estar se preparando para lhe atacar a qualquer instante, é preciso manter sempre o foco no andar do jogo e não somente numa atividade.

2.3.2 Criar Construções

Para treinar suas unidades e desenvolver novas tecnologias é necessário expandir seus domínios e criar novas construções. Normalmente esta atividade também é realizada pelos trabalhadores utilizando certa quantidade de recursos, dependendo da construção que se fará.

É comum nos jogos RTS começar apenas com uma construção principal chamada de *command center* (normalmente traduzida como, centro de comando ou centro da cidade) e poucas unidades. A coleta de recurso é feita abastecendo o *command center* com o que for coletado. Com a expansão dos seus domínios podem ser feitas novas construções que servirão para depositar os recursos coletados.

Existem também construções específicas para criar cada tipo de unidade e para pesquisar cada tipo de tecnologia. Assim como normalmente são exigidas algumas construções para se fazer certas expansões.

Também é de fundamental importância decidir quais construções é relevante para se fazer dependendo da estratégia tomada para vencer a partida

2.3.3 Treinar Unidades

O treinamento de unidades é realizado por uma construção ao custo de certa quantidade de recursos, dependendo da unidade treinada. As unidades básicas, como trabalhadores, normalmente são treinadas no próprio *command center*. Já unidades de tropa possuem construções próprias para serem desenvolvidas, algumas requerem também que o jogador já tenha pesquisado certas tecnologias.

2.3.4 Pesquisar Tecnologias

A pesquisa de tecnologia é feita em certas construções ao custo de certa quantidade de recursos. É essencial evoluir o seu povo para conseguir conquistar a vitória. Existem diversas tecnologias para se pesquisar e é necessário um bom domínio do jogo para definir quais são as essenciais que lhe garantirão uma boa vantagem contra seus inimigos.

Podem existir tecnologias que deixarão suas unidades mais rápidas, as que farão seus trabalhadores coletarem mais rápido, tecnologias que fortalecem suas edificações deixando-as mais resistentes aos ataques dos inimigos, tecnologias que aumentam habilidades de certas unidades de combate, seja a longa distância, ou armados, ou cavaleiros, espadachins, ou ainda que aumentem a defesa dessas unidades, existem até mesmo tecnologias que habilitam novas tecnologias, novas construções, ou novas unidades.

Enfim, não se vence uma partida criando apenas as unidades básicas, é preciso evoluir, criando unidades mais avançadas ou melhorando as unidades atuais. Uma unidade de nível de tecnologia superior geralmente possui um melhor custo-benefício que uma unidade menos avançada. [7]

2.3.5 Exploração

È comum aos jogos RTS os jogadores não possuírem uma visão completa do ambiente do jogo. Sendo utilizado o modelo de *Fog of War*, ou seja, onde o jogador não tem a percepção de tudo que se passa ao seu redor, há sempre um nível de incerteza. Cada unidade possui um campo de visão e o jogador vê, então, apenas o que está dentro do campo de visão de todas as suas unidades. É, então, necessário explorar o mapa por novos recursos, procurar por vantagens geográficas, para posicionar seu pelotão e para encontrar seus adversários.

2.3.6 Combate

O combate é a hora crucial do jogo, o momento definitivo. O objetivo dos jogos RTS é destruir seu adversário e para isso é necessário que todas as etapas discutidas anteriormente tenham sido bem executadas para se obter a vitória no momento de embate.

A estratégia na definição de que unidades utilizar, por onde atacar, o que atacar primeiro, em que momento atacar, todas essas variáveis são de fundamental importância para definir o vencedor da batalha.

Dependendo do jogo existem unidades especialmente fortes contra um tipo e fracas contra outros tipos, assim como existem tecnologias que combatem tais fraquezas e aumentam suas forças. O momento da batalha é o momento onde será botada a prova a estratégia tomada por cada um desde o início da partida.

Figura 2.10 Exército formado caminhando para batalha, em *Age of Empires II*

2.4 Inteligência Artificial em Jogos RTS

Jogos RTS como as populares séries *Warcraft* e *Age of Empires*, modelam o combate com várias unidades em tempo real. Atualmente, o interesse na pesquisa em inteligência artificial para jogos RTS tem crescido porque estes jogos impõem muitos desafios únicos de IA que também são relevantes para outras áreas importantes, como a colaboração de robôs e a simulação de combates militares. [8]

Alguns exemplos de desafio estão relacionados com raciocínio temporal e espacial, tomada de decisão em ambientes dinâmicos e incertos, a presença de um grande número de objetos interagindo, e a necessidade de abstração e planejamento, devido à predominância de ações com efeitos locais. [8]

A maioria dos jogos RTS são altamente configuráveis o que nos permite olhá-los sobre a perspectiva de um tópico de pesquisa específico. Por exemplo, nós podemos inicialmente nos restringir a simples sub-jogos com informações completas antes de nos endereçarmos a jogos complexos e com informações imperfeitas. [8]

Seguindo este modelo diversos níveis dos jogos RTS são tratados por competições internacionais de IA, visando o crescimento da pesquisa nessa área e o desenvolvimento de novas abordagens e técnicas.

É sobre esta perspectiva que nos endereçaremos no próximo capitulo às competições de IA. Falando um pouco de como estas funcionam e especificamente, os modelos de competições que existem na área de jogos RTS.

2.5 Conclusão

Nesta seção falamos sobre jogos de estratégia em geral, dando uma definição formal para o mesmo e discutindo sobre as diversas modalidades desses jogos até chegarmos ao modelo de jogos RTS que é o tópico básico deste trabalho. Demos um breve histórico dos jogos RTS e então discutimos detalhadamente sobre as principais ações nesses jogos, definindo sua importância e visualizando a complexidade de cada uma.

Como vimos, os jogos RTS são um domínio extremamente complexo e interessante para o campo de pesquisa de técnicas de IA em tempo real. Infelizmente, o estado atual dos sistemas de IA nesses jogos é fraco. Michael Buro cita em seu trabalho [8] diversos motivos pelo quais essas Inteligências Artificiais ainda deixam muito a desejar quando confrontadas por humanos. Alguns desses motivos são, a complexidade do domínio dos jogos RTS, as restrições de tempo e dinheiro sobre os quais as produtoras tem que desenvolver esses jogos, e a não necessidade de uma IA muito avançada nos jogos que oferecem suporte a *multiplayer*. [9]

Talvez um dos maiores empecilhos para a evolução da IA nos jogos RTS tenha sido justamente o advento dos jogos *multiplayer*. A maioria dos jogos de estratégia em tempo real atuais tem como foco a disputa de partidas multi-jogadores. Isso faz com que os desenvolvedores deixem de investir tempo e dinheiro para refinar a porção *singleplayer* desses jogos e, conseqüentemente, a IA do computador. Esse fato não preocupa muito os jogadores, pois eles sabem que sempre haverá pessoas dispostas a disputar uma partida pela internet. [9]

3 Simuladores

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. [1]

Veremos neste capítulo um exemplo desse modelo que vem sendo bastante sucedido e é conhecido internacionalmente, a competição da RoboCup. E então faremos uma breve análise de como está este cenário na área de simulação de jogos RTS e abordaremos o RTSCup, o simulador que usaremos no nosso desenvolvimento.

3.1 RoboCup

A *RoboCup* é uma das mais famosas competições internacionais de inteligência artificial. Ela faz parte de uma iniciativa internacional para promover e incentivar a pesquisa e a formação na área. Para este propósito foi escolhido o jogo de futebol para ser alvo de estudo principal e foi traçada a diretriz de se criar até o ano de 2050 um time de robôs humanóides capaz de vencer o time humano campeão da Copa do Mundo. [10]

Para que um time de robôs possa realmente jogar futebol, diversas tecnologias devem ser incorporadas, como, projetar os princípios de agentes autônomos, colaboração multiagente, aquisição estratégica, raciocínio em tempo real, robótica e a fusão de sensores. [10]

O foco principal da *RoboCup* é a competição de futebol, mas os jogos são também uma importante oportunidade dos pesquisadores se reunirem e trocarem informações técnicas. Assim como uma ótima oportunidade para ensinar e entreter o público.

Além da competição de futebol outras modalidades de competições também têm espaço na *RoboCup*, como competições de simulação de resgate e atividades de cunho educacional visando atrair o interesse de estudantes para a área, com competições amadoras.

3.2 Simuladores RTS

Como vimos no capítulo anterior, os jogos RTS fornecem um domínio bastante interessante para a pesquisa de novas técnicas de IA em tempo real. Infelizmente, este campo ainda não é explorado em toda sua capacidade, principalmente pela falta de simuladores, de qualidade e código aberto, que sejam facilmente configuráveis e customizáveis.

Em seu trabalho, Victor Costa [9], apresentou as seguintes justificativas para a criação de seu simulador, seguindo os motivos citados por Michael Buro [8]:

1. As companhias de jogos não estão dispostas a liberar os protocolos de comunicações de seus jogos, nem a de adicionar interfaces de IA que permitam que desenvolvedores e pesquisadores acoplem programas que simulem a IA dos jogos. Ambas as *features* são necessárias para permitir que esses jogos sejam usados como plataforma de teste de técnicas de IA por pesquisadores interessados.

2. Em partidas *multiplayers*, a maioria dos jogos comerciais realizam todas as simulações do jogo na máquina do cliente, apenas não mostrando as informações indevidas ao jogador. Apesar de ajudar a economizar banda na hora da comunicação, essa abordagem torna esses jogos vulneráveis a certos tipos de *hacks*, em especial a aqueles capazes de revelar todo o mapa ao jogador. A solução para isso seria um simulador onde toda a lógica do jogo fosse simulada no servidor, e as únicas tarefas dos clientes seriam mandar ações (que seriam validadas pelo servidor), e receber periodicamente atualizações sobre o estado da simulação.

3. A maioria dos jogos comerciais não são flexíveis a ponto de permitir que o jogador altere as configurações do jogo, como atributos das unidades e construções. É importante que um simulador seja flexível para permitir que os pesquisadores adequem o jogo às suas necessidades.

Seguindo a análise feita por Victor Costa [9], optamos utilizar o simulador por ele aprimorado e que já vem sendo usado na Universidade Federal de Pernambuco para fazer pesquisas em jogos RTS.

3.2.1 RTSCup

O RTSCup foi criado na Universidade Federal de Pernambuco e é utilizado nas competições de Inteligência Artificial da disciplina de Agentes Autônomos. Foi primeiramente concebido por Vicente Vieira [12] e aprimorado no Trabalho de Graduação de Victor Costa [9].

Ele surgiu como uma evolução do simulador JARTS (*Java Real Time Strategy*)[11]. A principal diferença do RTSCup em relação ao JARTS está no fato dele adotar uma arquitetura cliente-servidor, enquanto que o JARTS, só roda localmente. A vantagem de se adotar a arquitetura do RTSCup é que ela não obriga o desenvolvedor da IA a ter que utilizar a mesma linguagem de programação do simulador; pode-se usar qualquer linguagem, contanto que seja possível mandar as mensagens necessárias ao servidor via socket. [9]

Figura 3.1 Tela inicial do RTSCup

3.2.2 Descrição do Ambiente

O RTSCup possui dois modos de jogo atualmente implementados. O objeto de estudo escolhido para este trabalho envolve a simulação do *game 1* do RTSCup, que se trata de um ambiente de coleta de recursos. Dependendo do mapa existe um número específico de agentes, um *command center* e diversos recursos e obstáculos, espalhados. Estes agentes serão os trabalhadores que farão a coleta dos recursos, indo até as minas, coletando ouro e retornando até o *command center,* para depositar o que for coletado.

É então esperada a utilização deste mapa para pesquisar técnicas que acelerem este processo, estando envolvidos nessa tarefa, a escolha da mina que cada agente deve minerar, a escolha do caminho a ser percorrido, e certo nível de coordenação entre os agentes para que estes não atrapalhem as tarefas uns dos outros.

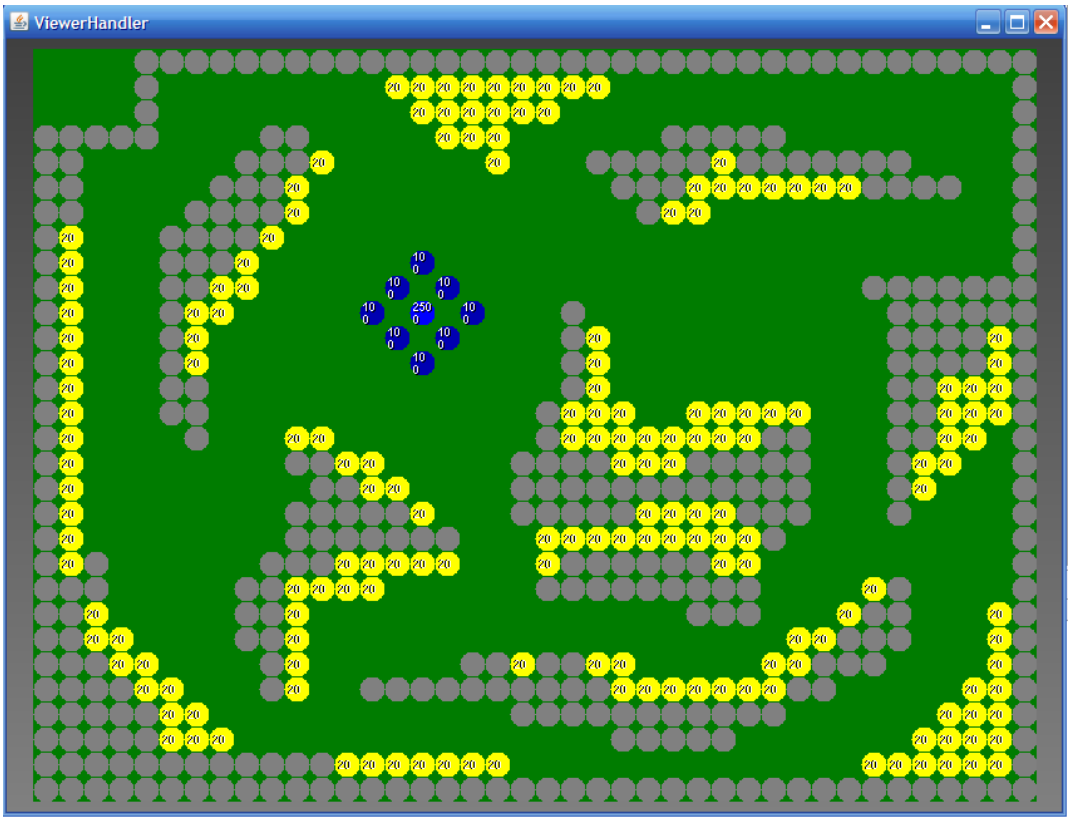
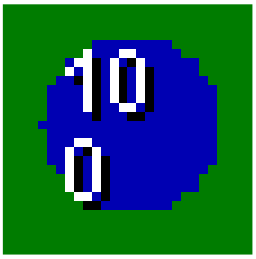
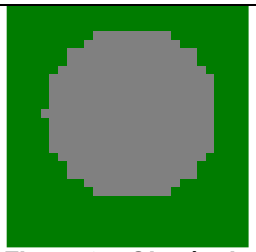


Figura 3.2 Tela inicial de uma partida no *game 1* do RTSCup

Trabalhadores: São os agentes que realizam a coleta dos recursos. Eles possuem a informação dos seus pontos de vida e da quantidade de recursos que estão carregando. No game 1 os pontos de vida não são utilizados e cada trabalhador pode carregar até 10 recursos por vez.	 **Figura 3.1 Trabalhador**
***Command Center*:** É nesta unidade que os trabalhadores devem depositar os recursos coletados. Os números indicam, respectivamente, os pontos de vida dessa construção e a quantidade de recursos total armazenada.	 **Figura 3.2 *Command center***
Minas: As minas são os recursos a serem coletados. Elas estão espalhadas por diversas localidades do mapa. O número indicado nelas especifica a quantidade de recursos que ainda existem ali. Quando este número chega a '0' a mina desaparece do mapa.	 **Figura 3.3 Recurso**
Obstáculos: Servem como as "paredes" do mapa. As unidades não podem atravessar os obstáculos tendo que contornar o caminho para chegar onde necessitam.	 **Figura 3.4 Obstáculo**

3.3 Conclusão

Neste capítulo falamos da importância de simulações para as pesquisas na área de inteligência artificial, em especial as competições em ambientes simulados, que instigam o desenvolvimento de novas soluções para os problemas estudados. Como exemplo de sucesso citamos a competição da RoboCup que acontece todo ano, em um local diferente, e reuni centenas de pesquisadores, estudiosos e curioso no assunto, e os congrega não apenas para competir ou assistir as competições, mas também para aprender e compartilhar novas idéias e assim ajudar o desenvolvimento da área.

Vimos também que apesar da grande gama de desafios que os jogos RTS apresentam ainda é baixo o número de bons simuladores para este fim. E então apresentamos o RTSCup, um simulador criado na Universidade Federal de Pernambuco para auxiliar e incentivar os alunos a desenvolver novas abordagens para os problemas de IA nos jogos RTS.

Definimos que o simulador utilizado neste projeto será o RTSCup emulando o problema da coleta de recursos, para qual os desafios e as propostas serão mostrados nos próximos capítulos.

4 Coleta de Recursos

Como foi dito anteriormente, a coleta de recursos é uma das etapas mais importantes de um jogo RTS e o problema de automatizar tal tarefa, de maneira eficaz, apresenta diversas dificuldades.

Neste capítulo explicaremos quais são estas dificuldades e como soluções já existentes endereçaram estes problemas.

No próximo capítulo será explicado como planejamos melhorar as soluções que já existem, a fim de criar um agente que seja mais eficaz na coleta de recursos.

4.1 Problemas

Como citamos no capítulo anterior a coleta de recursos apresenta basicamente três desafios: determinar qual o melhor recurso para um agente coletar, realizar o *pathfinding* (a escolha do caminho, seja até o recurso ou até o *command center*) e a coordenação entre os agente (para que cada um interfira o mínimo possível na tarefa dos outros). Cada um desses problemas será formalmente definido e explicado em maiores detalhes nos próximos tópicos.

4.1.1 Selecionar um recurso

Este é um dos principais problemas na coleta de recursos. Devemos fazer uma análise criteriosa das variáveis envolvidas para determinar a vantagem de se escolher tal recurso ao invés de outro, nos preocupando sempre em coletar o máximo de recursos no menor tempo possível. Algumas dessas variáveis serão explicadas a seguir.

4.1.1.1 Distância

A distância que o agente se encontra da mina e a distância da mesma ao *command center* são de fundamental importância na seleção do recurso a minerar. Esta informação tem um valor muito alto já que estima o número de passos que o agente fará para conseguir coletar certa quantidade de recurso.

Devido à dinâmica do ambiente multiagente é impossível prever com certeza quantos passos serão realizados então para nos aproximarmos cada vez mais deste valor utilizamos diferentes tipos de distâncias, a fim e deixar nossas estimativas mais precisas. Dentre as utilizadas neste trabalho podemos classificá-las em dois subtipos, as algébricas e as algorítmicas.

As algébricas são as obtidas diretamente através de uma fórmula matemática. Estas distâncias ignoram a existência de obstáculos entre a origem e o destino, já que não fazem nenhuma checagem deste caminho, logo subestimam a distância real. As distâncias algébricas utilizadas foram a distância de *manhattan* [15] e a distância euclidiana. A distância de *manhattan* ou distância L_1 é obtida somando-se o valor das diferenças absolutas das coordenas dos extremos, o segundo nome nos faz alusão ao fato desta distância corresponder à distancia em L de um ponto a outro. A distância euclidiana é a menor distância geométrica entre dois pontos, a qual seria se traçássemos uma reta da origem ao destino.

As distâncias algorítmicas são, claramente, obtidas através de algum processamento algorítmico do mapa. Estas distâncias consideram a existência de obstáculos, foram elas, o *shortest path* e a distância de turnos. O *shortest path* é a menor distância entre os dois pontos considerando-se a existência de obstáculos, mas desconsiderando os agentes como tais. A distância de turno considerada no trabalho de Sergio Sette [7] será explicada mais a frente.

4.1.1.2 Disponibilidade e Acessibilidade

Sergio Sette [7] destacou em seu trabalho a existência de dois fatores que também tem uma grande influência em determinar a viabilidade de um recurso, a disponibilidade e a acessibilidade.

A disponibilidade determina a possibilidade de um agente coletar uma mina específica ou ainda o quão disponível ela está. Depende, então, da quantidade de espaços vagos ao seu redor e do número de agentes que já estão minerando-a.

A acessibilidade define a facilidade de se chegar a algum recurso, se o caminho até a mina e de volta ao *command center* é de fácil ou difícil acesso. É determinado tanto pela quantidade de obstáculos no percurso até a mina quanto pela quantidade de agentes que estão passando nas imediações e conseqüentemente estão atrapalhando o acesso à mina.

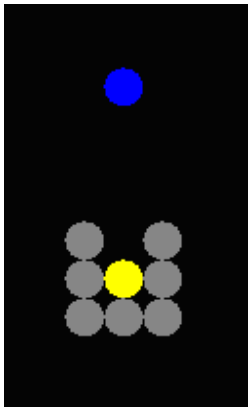


Figura 4.1 Recurso com baixa disponibilidade mas alta acessibilidade

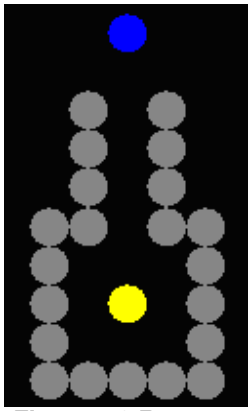


Figura 4.2 Recurso com alta disponibilidade mas baixa acessibilidade

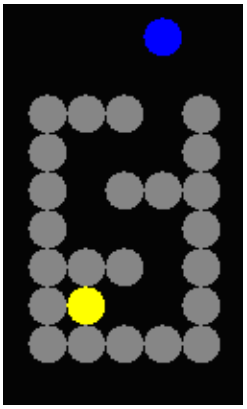


Figura 4.3 Recurso com baixa disponibilidade e acessibilidade

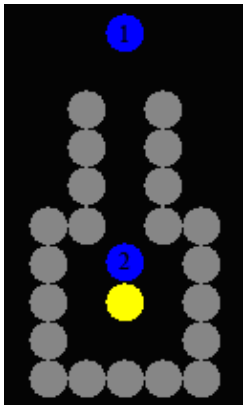


Figura 4.4 O agente 2 está impedindo o acesso do agente 1 ao recurso

4.1.2 Pathfinding

O *pathfinding* refere-se ao problema de encontrar o melhor caminho entre dois pontos do mapa. Na tarefa de coleta de recursos é preciso fazer o *pathfinding* duas vezes, uma para encontrar o caminho até a mina selecionada e outra para determinar o caminho de volta até o *command center*.

Este é um problema extremamente custoso, computacionalmente, e a escolha de um bom algoritmo é primordial para o sucesso da tarefa. Sendo então necessária uma boa análise dos principais algoritmos existentes para escolher a solução que melhor se adéqüe ao caso de estudo.

Existem diversos algoritmos para resolver este problema. Notadamente, o algoritmo de *Dijkstra* e o de *Bellman-Ford* possuem uma solução perfeita, encontrando sempre o *shortest path* entre os pontos dados. Infelizmente a complexidade desses algoritmos é da ordem de $O(V^2)$ e $O(V.E)$, respectivamente, o que os tornam impraticáveis em soluções em tempo real.

Para soluções em tempo real, o algoritmo mais utilizado para resolver este problema é o conhecido A*. [13] Este algoritmo não dá a melhor solução para o problema, mas retorna uma solução suficientemente boa e aceitável para estes casos. Mais a frente explicaremos com mais detalhes como este algoritmo funciona.

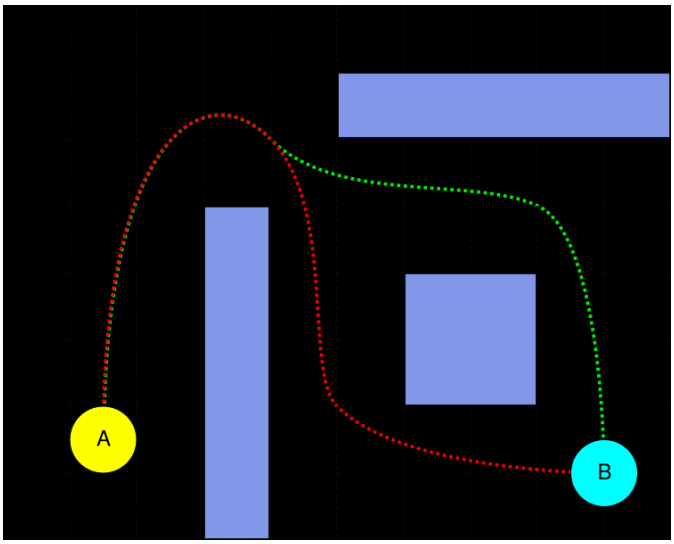


Figura 4.5 Possíveis rotas para ir de A a B

4.1.3 Coordenação

Em um ambiente multiagente, cada agente deve se coordenar uns com os outros, procurando a melhor solução para o conjunto, não para o individual. Existem casos onde tomar a decisão X seria a melhor escolha para o agente A, mas isto pode atrapalhar os planos do agente B, fazendo com que a produtividade total seja menor. O controle de organização do agente deve estar presente em todas as etapas de decisão do agente, desde a escolha do caminho, até a decisão de que recurso o agente escolherá para coletar. [7]

4.2 Soluções

Existe uma grande quantidade de artigos na área de coleta de recursos, porém a maioria se encontra na área de robótica. Estes artigos têm como foco a parte da movimentação e sensores dos robôs, falando muito pouco sobre a estratégia de coleta. [7]

Em especial. na área especifica de coleta de recursos em jogos RTS, encontramos os trabalhos de José Carlos Moura [1] e Sergio Sette [7] e discutiremos as propostas de cada um a seguir.

4.2.1 JcmjWorker

O agente implementado por José Carlos Moura [1] foi apelidado de JcmjWorker. Ele trabalha sobre o princípio de que é necessário sempre minerar a mina mais próxima (pela distância de *manhattan*) e carregar o máximo de minério possível sem tomar muitas precauções com questões de disponibilidade e acessibilidade ou de coordenação entre os agentes.

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, utilizando o algoritmo A*, 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 já 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 reservados e quanto tempo vai demorar a ficarem livres. [1]

4.2.2.1 O Algoritmo A*

O A* é o algoritmo de busca heurística mais difundido e também o mais utilizado na área de jogos, por ser um algoritmo simples, eficiente, e completo, ou seja, se existir um caminho do ponto de origem ao ponto de destino, ele irá encontrá-lo. Outra vantagem deste algoritmo é determinar locais onde o custo de passagem é maior ou menor, como regiões montanhosas ou estradas. [7]

Ele se utiliza de uma função heurística que nada mais é que uma medida estimada do custo de ir de um ponto até o objetivo. Essa função estimada é definida de acordo com a implementação e neste caso foi utilizada a distância de *manhattan*.

Este algoritmo é um algoritmo guloso, os nós são expandidos escolhendo sempre os de menor custo ainda não visitados, até que se chegue ao destino ou que não haja mais nós na lista, o que significa que não existe caminho.

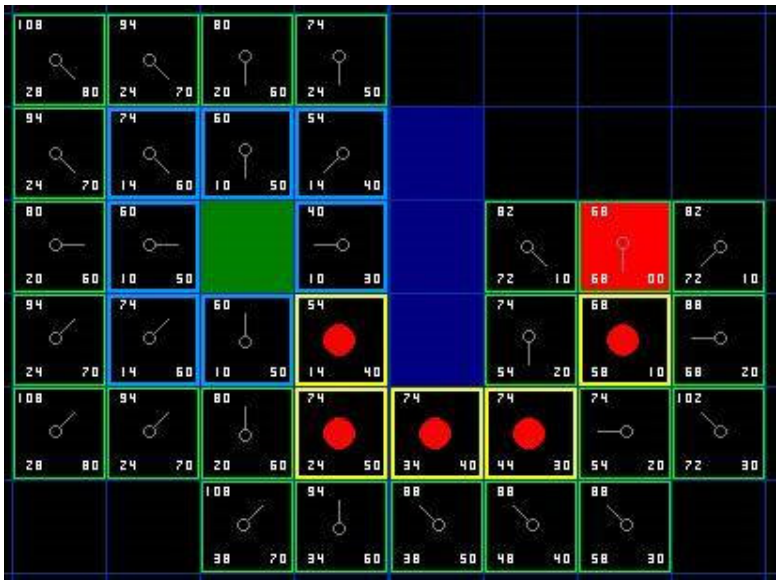


Figura 4.6 Caminho encontrado pelo algoritmo A* (bolas vermelhas)

4.2.2.2 Máquina de Estados

O agente sugerido em [1] possui cinco estados bem definidos. São eles:

- **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 *command center* e vai seguir essa rota até o final, ao chegar ao *command center* o agente deposita todo o recurso que ele esta carregando. E em seguida seu estado é setado como IDLE.

4.2.2 ZigWorker

Este agente foi concebido por Sergio Sette [7] e possui este nome peculiar por apresentar um agente explorador “zig” que destoa da lógica principal de escolha de mina para encontrar possíveis candidatos melhores, já que como veremos a seleção da mina é feita a partir de uma pontuação aproximada.

4.2.2.1 A Seleção do recurso

A seleção da mina neste agente é feita considerando-se vários fatores além da distância. Foi criada então uma função de utilidade que nada mais é que uma função que atribui uma relevância a cada fator para assim poder calcular um escore a cada mina. Citaremos a seguir os fatores que são levados em conta no cálculo desta função.

A distância do *command center* até a mina é iniciada com a distância *manhattan* entre eles, mas sempre que um agente completa o percurso até uma mina esta distância é atualizada seguindo a fórmula: distância = (1 – taxa de aprendizado) * distância antiga + taxa de aprendizado * distância atual. Onde a distância atual se refere à quantidade de turnos que o agente gastou para chegar até a mina (a distância de turnos citada anteriormente), ou seja, a distância real da mina ao *command center* levando em consideração os obstáculos do percurso até lá, logo acrescentando o princípio de acessibilidade ao cálculo. Foi criado também o conceito de região, onde, quando a distância de uma mina é atualizada esta é propagada às minas da mesma região visto que minas adjacentes possuem os mesmos níveis de acessibilidade.

O número de agentes que já se encontram minerando a mina e quantos espaços vagos esta mina possui, são fatores que determinam a disponibilidade da mina e que auxiliam na prevenção de zonas muito congestionadas.

A distância do agente à mina também é levada em consideração caso o agente tenha exaurido uma mina, mas ainda não esteja carregando sua carga máxima, logo está procurando uma mina próxima ao local que se encontra. Para não realizar cálculos adicionais é considerada a distância *manhattan*.

4.2.2.2 Algoritmo A* Colaborativo

O algoritmo A* funciona bem para ambientes monoagentes, mas em ambientes multiagentes a colisão precisa também considerar a presença de outros agentes no ambiente. O ZigWorker utiliza esta versão modificada do algoritmo A* levando então em consideração, também, a colisão temporal com outros agentes.

Sempre que um agente encontra um caminho antes de iniciar seu percurso o agente o reserva numa matriz temporal (que considera o tempo um eixos, possuindo pra cada índice indicador do turno a configuração do mapa naquele instante). Deste modo quando outro agente for procurar por um caminho terá o registro dos caminhos já alocados e terá que contornar as colisões.

Esta solução possui melhorias significativas no deslocamento dos agentes, mas apesar de evitar os problemas de congestionamento aumenta ainda mais a sobrecarga do *pathfinding* que já é a ação mais custosa do agente. Sem contar o alto nível de complexidade para manter esta lista sempre atualizada, sincronizada com o turno atual.

4.2.2.3 Máquina de Estados

Segue abaixo a máquina de estados descrita em [7] para o agente ZigWorker:

- **FIND RESOURCE**: É o estado inicial do agente. Neste estado o agente procura a melhor mina de acordo com a função de utilidade implementada. Esta mina é armazenada na memória do agente. Se não houver mais minas disponíveis, o agente entra no estado IDLE, caso contrário, ele entra no estado MOVE TO MINE.
- **MOVE TO MINE**: O agente executa o algoritmo de *pathfinding* e se move em direção à mina armazenada pelo estado FIND RESOURCE até chegar no local de mineração, onde ele entra no estado GATHERING.
- **GATHERING**: Neste estado o agente executa a ação de coleta na mina armazenada em sua memória. Se a mina se esgotou, e o agente ainda não está cheio, o agente passa para o estado FIND RESOURCE, se o agente já estiver cheio, ele passa para o estado MOVE TO COMMAND CENTER.
- **MOVE TO CC**: O agente executa o algoritmo de *pathfinding* e se move em direção ao *command center*. Quando o agente atinge o *command center*, ele passa para o estado de DELIVER.
- **DELIVER**: Neste estado o agente entrega os recursos no *command center*, e entra novamente no estado FIND RESOURCE.
- **IDLE**: Este estado foi criado para evitar sobrecarga de execuções do pathfinding. Nele, o agente fica parado por cinco turnos, quando entra novamente no estado FIND RESOURCE.

4.3 Conclusões

Podemos perceber que o agente JcmjWorker, apesar de simples é bastante eficaz em mapas pequenos, apresentaria um rendimento baixo em mapas com altos níveis de congestionamento ou em mapas grandes, visto a simplicidade do seu método de seleção de mina e a falta de uma coordenação mais efetiva dos agentes.

O agente ZigWorker propôs diversas novas abordagens que melhoraram significativamente a eficácia do agente, combatendo principalmente estas falhas. Destacam-se as medidas tomadas para aumentar a colaboração entre os agentes, como os critérios de acessibilidade e disponibilidade, na escolha da mina, e no algoritmo A* colaborativo.

Apesar da melhora expressiva, ainda existem alguns fatores que podem ser melhorados nesta abordagem, entre eles utilizar outra heurística para o *pathfinding,* já que como discutido anteriormente, a distância *manhattan* subestima bastante a distância real entre os pontos, por não considerar a existência de obstáculos. Para isto é preciso também levar em consideração a sobrecarga já imposta à etapa de *pathfinding* e encontrar formas mais eficientes de implementar este algoritmo.

Outros fatores são os critérios de seleção de mina que são inicializados com a distância *manhattan* e só são atualizados quando um agente coleta uma mina da região, o que dispararia as atualizações nas distâncias de turno dos agentes daquela região. Apesar da existência do agente “zig” servir para amenizar este fato, já que ele visitaria estas minas não visitadas e dispararia essas atualizações, num jogo onde o ambiente é totalmente observável a razão da existência de tal agente “explorador”, diminui o número de agentes realmente efetivos na coleta e, é apenas um paliativo para o real problema de custo computacional alto de executar cálculos mais elaborados.

5 Proposta

A proposta de estratégia apresentada leva em consideração os diversos fatores discutidos até aqui, em especial os pontos fracos encontrados no agente ZigWorker, visando criar um agente baseado neste porém com modificações nas observações feitas no capítulo anterior.

Neste capitulo serão expostas as idéias desenvolvidas e será dada uma estrutura que viabilize a implementação da mesma com o fim de criar um agente eficiente na tarefa de coleta de recursos.

5.1 Grid Computing

Grid Computing é a técnica de se utilizar de diversos computadores em paralelo, todos resolvendo o mesmo problema. É uma forma de computação distribuída onde um computador mais poderoso e virtual é na verdade composto por pequenas unidades autônomas, mas que atuam juntas para resolver problemas maiores. *Grid Computing* depende de um software para dividir e atribuir as peças do programa a todos os computadores. [16]

Percebemos que o princípio básico de um sistema multiagente se assemelha bastante com este modelo. Ainda mais quando nos lembramos que em competições como *RoboCup* cada agente atua realmente em um *hardware* diferente. É baseando-se nas idéias de *grid computing* que foi concebida a estratégia da proposta que será apresentada.

5.2 GridWorker

Como observamos anteriormente, diversas etapas dos modelos apresentados utilizam-se de cálculos simples para fazer certas tomadas de decisão, como alguns critérios da escolha da mina e a heurística do *pathfiding*. Realizar cálculos mais complexos teria um custo computacional muito alto o que geraria grandes atrasos na execução das tarefas e diminuiria ao invés de aumentar a eficácia da coleta.

O agente proposto foi chamado de GridWorker seu principal objetivo é tirar proveito, de alguns momentos de ócio dos agentes, para realizar cálculos que serão utilizados para melhorar a eficiência da tarefa de coleta de recursos.

Esses momentos são especialmente os turnos de movimentação e de coleta, que constituem a maior parte das ações do agente.

Analisando a máquina de estados dos agentes do capítulo anterior nota-se que os agentes só realizam cálculos em três momentos, para selecionar a mina, para mover-se até a mina e para mover-se até o *command center*. Nos outros momentos apenas ações unitárias são chamadas, "*move*", "*collect*", "*deliver*".

Por que o processo de cálculo é condensado em apenas alguns momentos enquanto na maioria do tempo os agentes não fazem nada? Por que gerar atrasos em alguns turnos para conseguir realizar o *pathfinding* se nos turnos anteriores não estávamos realizando nenhuma operação e este cálculo poderia ter sido adiantado?

É para resolver estas questões que o GridWorker foi idealizado.

5.2.1 Modelo de arquitetura

Para obter tal resultado na implementação do GridWorker é necessário estruturar um modelo que viabilize os agentes a continuarem executando suas tarefas regulares, mas que também lhes permita executar tarefas adicionais após isso.

Para viabilizar a execução destes cálculos pesados estes têm que ser estruturados de forma a serem executados de uma maneira iterativa. Assim, após a execução normal de cada turno o agente realiza uma iteração do algoritmo.

Este cálculo adicional foi chamado de *PosWork* e só é executado se após a execução normal o turno ainda não terminou, ou seja, o agente não pode executar uma ação, o que caracterizaria o tempo ocioso mencionado anteriormente.

A execução do agente não mais se resume a turnos onde ele toma decisões, executa uma ação e espera pelo próximo turno. Agora o agente mantém sua execução continuamente e o controle de turno tem que ser feito internamente.

Para não gerar uma sobrecarga de execução foi definido um número máximo de iterações de *PosWork* que um agente executa num turno. Após cada iteração o agente verifica se o turno já passou pra saber se ele deve voltar à execução normal se não, e ainda não executou o número máximo de iterações, ele executa mais uma iteração e assim sucessivamente.

Fica a cargo de uma estrutura superior que foi chamada de *VitualWorld* a tarefa de designar a cada agente qual será o *PosWork* que este irá executar. E quando este acabar este cálculo a de lhe atribuir uma nova tarefa.

5.2.2 Cálculo de distâncias

Um dos cálculos de extrema importância e que poderia estar sendo realizado, mas que não estava por causa do custo computacional, é o cálculo do *shortest path* das minas e *command center* aos pontos do mapa.

Este cálculo pode ser realizado uma única vez, e armazenado pela estrutura que contiver estas unidades. Por exigir o caráter iterativo da execução e pela simplicidade do algoritmo foi escolhido o algoritmo *BFS* (*Breadth First Search*) [17], busca em largura, já que cada movimentação possui o mesmo custo este o algoritmo mais indicado para estas situações.

A par estes valores estes poderiam ser utilizados no escore das minas, acoplando-os na função de utilidade e descartando a distância aproximada de *manhattan*, a fim de deixar a escolha mais precisa. É preciso porém se ater ao fato de que algumas minas podem não ter seu *shortest path* calculado ainda, já que este é feito por partes, então é necessário haver uma sincronização para que este fator só seja considerado após o termino de todos os cálculos.

Estas distâncias também poderiam ser utilizadas como heurística do *pathfinding* no A*, ou no A* Colaborativo. Neste caso o ganho ainda seria maior já que sendo a função heurística o menor caminho exato o A* passará a reportar sempre o menor caminho até a mina ou até o *command center*.

5.2.3 Máquina de estados

A máquina de estados do GridWorker é similar às anteriores. A cada turno o agente executa uma ação do estado, a grande diferença é que sempre após executar a ação o trabalhador entra num estado de ***GRIDWORK*** onde executa a tarefa que estiver pendente. O agente permanece neste estado, a cada iteração verificando se o turno já terminou, quando isto acontecer ele retorna ao seu estado anterior para executar a próxima ação.

Os estados das ações comuns seguem a mesma linha dos agentes vistos anteriormente. São eles:

- **SEARCH**: É o estado inicial do agente. Neste estado o agente procura a melhor mina de acordo com a função de utilidade implementada semelhante à [7]. Esta mina é armazenada na memória do agente e é executado o algoritmo de *pathfinding* para encontrar o caminho em direção à mina armazenada. Este caminho é reservado, usando o A* Colaborativo explicado anteriormente, e o agente se moverá a cada turno até chegar ao local de mineração, onde ele entra no estado COLLECT.
- **COLLECT**: Neste estado o agente executa a ação de coleta na mina selecionada. Quando o agente estiver cheio ele passa para o estado RETURN. Se a mina se esgotou, e o agente ainda não está cheio, ele volta para o estado SEARCH.
- **RETURN**: O agente executa o algoritmo de *pathfinding* e se move em direção ao *command center*. Quando o agente atinge o *command center*, ele passa para o estado de DELIVER.
- **DELIVER**: Neste estado o agente entrega os recursos no *command center*, e entra novamente no estado SEARCH.

5.3 Problemas Encontrados

O grande problema para a implementação do *GridWorker* foi como realizar o gerenciamento de turnos. Tal artifício deveria se valer da maneira como o simulador *RTSCup* lida com a passagem de turno para agir coerentemente com este fato.

Vários *bugs* de inconsistência aconteceram, alguns fatores foram levantados com Victor Costa [9], o que causou um melhor entendimento da plataforma, mas ainda assim *bugs* foram descobertos no simulador que causaram grande atraso no desenvolvimento, alguns não foram detectados a causa primeira, mas tratam-se de erros de sincronização com o servidor do jogo. O fato dos turnos não terminarem não foi planejado quando o *RTSCup* foi concebido e este comportamento inesperado que ocasionou nestes erros.

O controle de acesso de múltiplas *threads* também foi um problema principalmente para depurar os *bugs* encontrados

Além das dificuldades de compreender a maneira que o simulador funciona também foi muito difícil conceber um modelo que permitisse ao agente não perder seu estado atual e mesmo assim executar tarefas terceiras e ainda retornar a sua tarefa original sem que isto ocasionasse algum atraso na execução. Problemas dessa natureza mostraram-se ainda mais complexos quando se tentou inserir o princípio de reservas de caminhos, foi por causa disso que resultados mais efetivos do trabalho não puderam ser apresentados, já que sem reservas de caminho a eficácia do agente cai gigantescamente pela quantidade de colisões o que impediu uma comparação efetiva com os outros agentes, como era planejado.

6 Conclusão

Os jogos RTS impõem diversos desafios para área de inteligência artificial. Entre estes está o problema da coleta de recursos multiagentes. Para que tais pesquisas tenham sucesso é de fundamental importância a existência de simuladores que sirvam de benchmark para as soluções propostas. Este é exatamente um dos grandes problemas para o desenvolvimento da área.

A Universidade Federal de Pernambuco possui seu próprio simulador criado por alunos do Centro de Informática, que vem sendo utilizado em competições entre equipes da disciplina de Agentes Autônomos. Este simulador, o *RTSCup*, foi o simulador escolhido para a implementação deste projeto.

Apresentamos os problemas comuns ao problema da coleta de recursos e como possíveis soluções endereçaram tais problemas. Propomos então uma solução própria nos baseado nas qualidades destes e focando nas suas falhas e então sugerimos um agente novo que se espera que com os devidos ajustes tenha uma grande eficácia na tarefa da coleta de recursos.

6.1 Trabalhos Futuros

Corrigir os problemas do *GridWorker* com relação ao controle de reservas, finalizar sua implementação e realizar testes para validar sua eficácia. Implementar as idéias de tornar o *pathfinding* iterativo e assim adiantar o seu processamento como um cálculo de *PosWork* e procurar novas idéias que aumentem a eficácia da coleta de recursos.

Criar modelos do *GridWorker* para coleta em ambientes de *Fog of War*. Estender a idéia do *GridWorker* para o *game* 2 e *game* 3.

Implementar um agente baseado no *GridWorker* nos moldes do game 1 para competição do ORTS na AIIDE.

Referências

[1] MOURA, José Carlos. **Uma estratégia eficiente de coleta multiagente para jogos RTS.**

[2] ROBOCUP FEDERATION. **RoboCup Brief Introduction**. Acesso em: 26/08/2008. Disponível em: <http://www.robocup.org/Intro.htm>.

[3] VIEIRA, Vicente; WEBER, Renan; MOURA, José Carlos. **Agentes Autônomos**. Acesso em 26/08/2008. Disponível em: <http://www.cin.ufpe.br/~vvf/rtscup/files/docs/RTSCup.ppt>.

[4] WIKIPEDIA. **Strategy Game**. Acesso em: 25/10/2008. Disponível em: <http://en.wikipedia.org/wiki/Strategy_games>

[5] WIKIPEDIA. **Strategy Vídeo Game**. Acesso em: 25/10/2008. Disponível em: <http://en.wikipedia.org/wiki/Strategy_video_game>

[6] WIKIPEDIA. **Real Time Strategy**. Acesso em: 25/10/2008. Disponível em: <http://en.wikipedia.org/wiki/Real-time_strategy>

[7] SETTE, Sergio Schechtman. **Um estudo de estratégias para coleta de recursos em ambientes multiagente**.

[8] BURO, Michael; AHA, David; STURTEVANT, Nathan; CORRUBLE, Vincent. **Complex Video Game AI Competitions at AIIDE'06**.

[9] CISNEIROS, Victor Costa de Alemão. **APERFEIÇOAMENTO DO SIMULADOR RTSCUP**.

[10] ROBOCUP FEDERATION. **What is RoboCup.** Acesso em: 30/11/2008. Disponível em: <http://www.robocup.org/overview/21.html>

[11] WEBER, Renan. **JARTS – Java Real Time Strategy**. Acesso em 30/11/2008. Disponível em: <http://www.cin.ufpe.br/~tg/2006-1/rtw.pdf>.

[12] FILHO, Vicente Vieira. **RTSCup.** Acesso em 18/10/2008. Disponível em: http://www.rtscup.org/

[13] WIKIPEDIA. **Pathfinding**. Acesso em: 06/12/2008. Disponível em: <http://en.wikipedia.org/wiki/Path_finding>

[14] Lester, Patrick. **Pathfinding for Beginners**. Acesso em: 22/11/2008. Disponível em: <http://www.policyalmanac.org/games/aStarTutorial_port.htm>

[15] WIKIPEDIA. **Taxicab Geometry**. Acesso em: 06/12/2008. Disponível em: http://en.wikipedia.org/wiki/Taxicab_geometry

[16] WIKIPEDIA. **Grid Computing**. Acesso em: 06/12/2008. Disponível em: <http://en.wikipedia.org/wiki/Grid_computing>

[17] WIKIPEDIA. **Breadth-first Search**. Acesso em: 11/12/2008. Disponível em: < http://en.wikipedia.org/wiki/Breadth-first_search>