Universidade Federal de Pernambuco

Graduação em Ciência da Computação

Centro de Informática

APERFEIÇOAMENTO DO SIMULADOR RTSCUP

TRABALHO DE GRADUAÇÃO

Aluno: Victor Costa de Alemão Cisneiros (vcac@cin.ufpe.br)

Orientador: Geber Lisboa Ramalho (glr@cin.ufpe.br)

Co-Orientadora: Patrícia Cabral de Azevedo Restelli Tedesco (pcart@cin.ufpe.br)

Recife, 25 de junho de 2008

“Ni!”

Assinaturas

Este Trabalho de Graduação é resultado dos esforços do aluno Victor Costa de Alemão Cisneiros, sob a orientação dos professores Geber Lisboa Ramalho e Patrícia Cabral de Azevedo Restelli Tedesco, conduzido no Centro de Informática da Universidade Federal de Pernambuco. Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação.

_____

Victor Costa de Alemão Cisneiros (aluno)

_____

Geber Lisboa Ramalho (orientador)

_____

Patrícia Cabral de Azevedo Restelli Tedesco (co-orientadora)

Agradecimentos

Primeiramente, agradeço aos meus pais, por me apoiarem e se esforçarem ao máximo para me dar a educação necessária à minha formação pessoal.

Agradeço ao professor Geber Lisboa Ramalho. que juntamente com Patrícia Cabral de Azevedo Restelli Tedesco me forneceram a orientação necessária para a elaboração deste trabalho

Agradeço à Universidade Federal de Pernambuco pelo apoio institucional e incentivo a minha formação acadêmica.

Por fim, agradeço a Sergio Sette por me ajudar a testar o projeto desenvolvido nesse trabalho.

Índice

Índice de Figuras

1 – Introdução

1.*1 – Motivação*

Competições de Inteligência Artificial são uma das melhores formas de estimular estudantes e pesquisadores da área a desenvolver e testar novas técnicas de IA. Uma das competições mais conhecidas no mundo é a Robocup, que tem como objetivo final desenvolver até 2050, um time de robôs humanóides, completamente autônomos, capazes de derrotar a seleção de futebol campeã do mundo em uma partida. [1]

Muitas dessas competições de Inteligência Artificial utilizam-se de programas para simular o ambiente da competição, a própria RoboCup possui uma liga onde os *softwares* jogadores (agentes) disputam uma partida em um campo virtual simulado dentro do computador. [1] Esses simuladores provêm uma forma eficiente de medir o desempenho das técnicas de IA e comparar resultados entre os diversos competidores.

Um ótimo ambiente de simulação para as competições de IA são os jogos de Estratégia em Tempo Real. O desenvolvimento de uma inteligência artificial para esses jogos apresenta diversos desafios interessantes e que também são relevantes em outras áreas como colaboração entre robôs e simulações de combates militares [2]. Alguns exemplos de problemas de IA presentes nesses jogos são: *path-finding*; lidar com informações incompletas; raciocínio espacial e temporal; planejamento dentro do domínio do jogo; coordenação entre múltiplos agentes, entre outros.

Dentro desse contexto de simuladores de Jogos de Estratégia em Tempo Real existem vários projetos, tais como o ORTS, Stratagus e Glest, que visam fornecer uma *engine* altamente customizável para a criação de jogos de estratégias, podendo-se inclusive programar toda a Inteligência Artificial. Alguns dos problemas dessas *engines* são, no entanto, a falta de documentação e usabilidade, a instabilidade dos programas e a falta de foco na questão da competição entre agentes. [3]

1.*2 – Objetivos*

Este trabalho tem como objetivo propor e implementar melhorias no RTSCup, um ambiente de simulação de jogos de estratégia em tempo real, desenvolvido no Centro de Informática da Universidade Federal de Pernambuco, e utilizado nas competições de Inteligência Artificial da disciplina de Agentes Autônomos.

1.*3 – Estrutura do Documento*

2 – Jogos de Estratégia em Tempo Real

Nesta seção, será dada uma definição do que são os jogos de Estratégia em Tempo Real, e em seguida, será feito um estudo da história desses jogos, dando destaque às inovações que cada jogo trouxe para o gênero. É importante fazer esse estudo, pois muitas das idéias desses jogos podem ser trazidas para o simulador RTSCup, de modo a torná-lo melhor e mais parecido com esses jogos comerciais.

2.1 – Definição

O termo jogo de Estratégia em Tempo Real (*RTS - Real Time Strategy game*) surgiu em 1992, como forma de distinguir um novo gênero de jogos de estratégias que não eram baseados em turnos. Bren Sperry, um dos fundadores da Westwood Studios e pioneiro nesse gênero de *video-games*, decidiu categorizar seu novo jogo em desenvolvimento por esse nome, pois achava que o termo anterior, *real-time wargames*, iria afastar muitos jogadores de experimentar esse novo tipo de jogo dinâmico [4]. Antes de 1992, o gênero de *wargames* era um mercado não muito popular, com a exceção dos jogos de Sid Meyer.

Algumas das características mais comuns nos jogos RTS incluem combates, coleta de recursos, construções de bases, desenvolvimento de tecnologias e o controle de unidades por meio de ordens. Geralmente esses jogos apresentam para o jogador uma visão de cima para baixo do campo de batalha, apesar de que alguns RTS 3D mais modernos permitirem uma liberdade total no controle da câmera. A característica principal desses jogos está no fato de que a ação não para, ao contrário dos jogos de estratégia baseados em turnos, onde os jogadores têm que esperar os turnos dos outros acabarem para poder agir.

É importante não confundir os jogos RTS com os jogos de simulação conhecidos como *god games*. Esses últimos possuem um público alvo bem diferente, apesar de apresentarem muitas características semelhantes, como o fato de serem jogados em tempo real. Alguns dos jogos conhecidos como *god games* são Simcity e Populous. Para evitar essa confusão, Bruce Geryk da GameSpot recomenda considerar jogos RTS

aqueles em que o princípio básico é coletar recursos, construir bases e destruir o inimigo [4].

2.2 – História

2.2.1 - Dune II

Apesar de já existirem alguns jogos de estratégia com elementos em tempo real, foi somente em 1992, com o lançamento de Dune II pela Westwood Studios, que estratégia em tempo real passou a ser considerado um gênero distinto de *video-games*.

Dune II: The Building of a Dinasty, foi um marco na história desse gênero de jogos. Foi ele que definiu as principais características dos jogos RTS modernos, tais como construção de bases, coleta de recursos, treinamento e movimentação das tropas.

Apesar de simples, a interface de Dune II serviu de base para os jogos RTS futuros, nela, o jogador selecionava unidades com o clique do mouse e clicava no local do mapa aonde queria que as unidades se movessem ou atacassem. Uma das características de Dune usadas em praticamente todos os RTS posteriores era o fato de o mapa do jogo começar todo escuro, e as áreas serem reveladas aos poucos a medida que o jogador as explorava com suas unidades.

Figura 1 – Dune II

Outra *feature* interessante de Dune II foi a introdução das árvores de tecnologia. Essas árvores consistem em dependências que cada unidade ou construção tem em relação às outras, por exemplo, para construir um *Barracks* antes é necessário construir um *Outpost*. Essa funcionalidade aumenta a diversidade estratégica, pois o jogador tem que decidir se coleta mais recursos, treina mais tropas ou evolui sua árvore de tecnologia.

Um dos principais problemas na jogabilidade de Dune II estava em não ser possível selecionar mais de unidade de uma só vez, e mandar ordens para todas elas, além disso, a inteligência artificial era bastante simples, os oponentes já começavam com suas bases prontas e apenas coletavam recursos e atacavam o jogador.

2.2.2 - Warcraft

O sucesso de Dune II estimulou o desenvolvimento de um outro jogo chamado Warcraft: Orcs & Humans, que foi lançado em 1994 pela Blizzard Entertainment. Warcraft era ambientado em um mundo medieval fantasioso e contava a história de duas facções inimigas, os Humanos e os Orcs.

Fora as diferenças nas imagens das unidades e algumas magias, as duas facções eram idênticas uma as outras, cada unidade dos Humanos tinha uma equivalente nos Orcs com os mesmos atributos (pontos de vida, ataque, movimentação). Fato esse que foi um retrocesso, visto que em Dune II, cada uma das três facções possuíam algumas unidades exclusivas com poderes diferentes.

Warcraft possuía uma jogabilidade bem parecida com a de Dune II, algumas melhorias estavam no fato de se poder selecionar mais de uma unidade por vez (no máximo quatro), e a de possuir *multiplayer* via conexão serial ou *direct modem link*. Warcraft foi, portanto, o primeiro RTS a suportar partidas *multiplayer*.

Dune II e Warcraft foram bem recepcionados e alcançaram boas notas da crítica, além de conseguirem um bom número de fãs, no entanto, foi somente alguns anos depois, com os próximos jogos da Westwood e Blizzard, que os jogos de estratégia em tempo real passaram se tornar bastante populares. Dentre esses jogos da 2ª geração estão a série Command and Conquer da Westwood Studios, e o jogo Warcraft II, da Blizzard Entertainment; discutiremos aqui apenas o último.

2.2.3 - Warcraft II

Warcraft II foi lançado em dezembro de 1995 e representou um enorme avanço em relação ao seu antecessor, tornando-se rapidamente um dos maiores sucessos da história dos jogos de computador. Entre as várias melhorias de Warcraft II estavam seus gráficos SVGA, uma IA mais avançada, capaz de evoluir suas construções do zero começando apenas com um peão, e diversas melhorias na jogabilidade, como poder selecionar até nove unidades e utilizar o botão direito do mouse tanto para movimentar quanto para atacar.

Um dos conceitos mais interessante introduzido em Warcraft II foi o *Fog of War*, uma espécie de neblina cinza permanente que cobria todo o mapa, escondendo todas as unidades inimigas que estavam foram do campo de visão das unidades do jogador. A presença do *Fog of War* aumentou drasticamente a complexidade estratégica do jogo, pois obrigava o jogador a constantemente ter que explorar o mapa para tentar descobrir o que seu oponente estava fazendo.

Figura 2 – Fog of War em Warcraft II.

Warcraft II foi também o primeiro RTS a se tornar popular nas partidas multiplayer pela Internet, conquistando uma legião de fãs. Tudo isso graças a um programa chamado Kali, que permitia com que jogos que utilizassem o protocolo IPX, como era o caso de Warcraft II, pudessem ser jogados pela Internet.

2.2.4 - Starcraft

Em 31 de março de 1998, a Blizzard lançou seu mais novo jogo de estratégia, chamado Starcraft. Diferente de Warcraft, que se passava em um mundo fictício medieval, Starcraft se passava no século 26 e tratava da guerra entre três raças: os Terran, humanos que foram exilados da Terra; os Zergs, uma raça de insectóides; e os Protoss, uma espécie de humanóides com poderes psíquicos e tecnologias avançadas.
Starcraft foi altamente reverenciado pela crítica, não só pela história do jogo, bem trabalhada, como também pela sua excelente jogabilidade e sua maior inovação: o fato das três raças do jogo serem completamente diferentes entre si. Em Starcraft, cada raça possui unidades distintas, cada uma com atributos e habilidades diferentes das outras; até a forma de se construir e treinar as unidades varia entre as raças de Starcraft,

enquanto que os Terrans constroem tudo como em qualquer outro RTS convencional, na raça dos Zergs as unidades nascem a partir de evoluções das larvas, que são geradas constantemente nos *Hatcheries*, a construção principal dos Zergs.
Outro grande fator que contribuiu para o sucesso de Starcraft foi a utilização da rede Battle.net, uma espécie de *chat* onde os jogadores com acesso a Internet podem se conectar, marcar partidas com outros jogadores facilmente, visualizar a lista de jogos abertos e competir no Ranking. A Battle.net foi originalmente construída para o jogo de RPG Diablo, da mesma produtora, e foi um grande sucesso, tanto em Diablo como em Starcraft, contribuindo para que esses dois jogos batessem recordes de vendas e entrassem na lista dos jogos mais vendidos de computador. A Battle.net foi uma verdadeira revolução para a popularização das partidas *multiplayers* pela Internet.

O sucesso de Starcraft foi tão grande que até hoje, mesmo após 10 anos de seu lançamento, ainda é um dos principais jogos da World Cyber Games (maior competição mundial de jogos), e na Koréia do Sul em especial, o jogo se tornou uma verdadeira febre, com jogadores profissionais ganhando patrocínios e disputando competições televisionadas. [5]

Figura 3 – Batalha em Starcraft, Raças distintas.

2.2.5 - Dawn of War e Company of Heroes

Nos anos de 2004 e 2006, a Relic Entertainment lançou dois novos jogos de RTS, Dawn of War e Company of Heroes, respectivamente. Diferente dos RTS tradicionais anteriores, esses dois jogos abandonam o esquema de treinar trabalhadores para coletar recursos, ao invés disso, os recursos são conseguidos automaticamente através do controle de pontos estratégicos, espalhados pelo mapa. Essa mudança foi feita para simplificar a forma de gerenciamento de recursos desses jogos, e ao mesmo tempo, obrigar os jogadores a constantemente expandir seus territórios, fazendo desse modo, com que eles se preocupem mais com a parte tática e os combates do jogo, do que com a coleta de recursos.

Outros avanços de Company of Heroes foram os cenários dinâmicos, onde construções, obstáculos e até mesmo tanques destruídos podem ser usados como cobertura para as tropas. Esses obstáculos também podem ser destruídos, e tudo é animado de forma extremamente real, graças à engine física que o jogo utiliza.

Figura 4 - Soldados se protegendo atrás de obstáculos em Company of Heroes.

3 - Inteligência Artificial em Jogos RTS

Os jogos de estratégia em tempo real oferecem diversos desafios interessantes aos pesquisadores de Inteligência Artificial. As técnicas desenvolvidas para esses jogos englobam as mais variadas áreas da IA como algoritmos de busca, aprendizagem de máquina, raciocínio lógico e simulações de comportamentos humanos. Além disso, técnicas desenvolvidas para esses jogos também podem ser aplicadas em outras áreas fora dos jogos, como por exemplo, a colaboração de robôs e simulações de combates militares [2].

Neste capítulo, serão descritos os principais desafios que devem ser resolvidos para a construção de um bom sistema de IA em um jogo RTS. Ao final, será feita uma análise sobre o estado atual desses sistemas nos jogos RTS modernos.

3.1 – Principais Desafios

3.1.1 – Tomada de Decisões com informações incertas

Diversos jogos RTS modernos implementam uma funcionalidade chamada *Fog of War*, que esconde do jogador todas as unidades e construções inimigas que estão fora do alcance de visão de suas tropas. Isso obriga o jogador a constantemente ter que enviar exploradores e lidar com informações incertas. Algumas das questões que o jogador ou IA comumente devem descobrir para vencer as partidas são as seguintes:

- Onde estão as bases inimigas?
- Meu oponente está treinando tropas para me atacar ou está evoluindo suas construções e unidades?
- Na última batalha meu oponente me atacou com carroças, monges e catapultas. Será que ele vai continuar com essa tática ou irá usar outra?

3.1.2 – Raciocínio Temporal e Espacial

Raciocínios sobre domínios espaciais e temporais são de extrema importância para qualquer jogador ou inteligência artificial de um RTS. Uma batalha pode ser

perdida em questão de segundos devido ao mau posicionamento das tropas, assim como uma partida pode ser perdida se o jogador não souber o momento certo de atacar, expandir e evoluir suas unidades. Algumas questões relacionadas ao raciocínio espacial e temporal em um jogo RTS são as seguintes:

- Será que estarei em desvantagem se enfrentar o inimigo nessa posição do mapa? Onde devo posicionar minhas tropas?
- Encontrei o inimigo marchando suas tropas em determinada região do mapa. Será que ele está vindo em direção à minha base?
- Em que momento do jogo devo criar uma expansão da minha base para coletar mais recursos?

Figura 5 – Dois tanques bem posicionados em Starcraft.

3.1.3 – Personalidade da IA

Uma das melhores formas de tornar a experiência de um jogo contra a máquina mais agradável é a de atribuir personalidades humanas à ela. Isso pode ser feito através da simulação de sentimentos, raciocínios inteligentes e atitudes desesperadas em situações urgentes. Algumas das formas de se fazer isso em um jogo RTS são:

- Fazer com que o oponente retribua um ataque feito por você, simulando uma espécie de vingança.
- Fazer com que o inimigo realize táticas arriscadas, que podem ou não dar certo, quando ele estiver a ponto de perder o jogo.
- Em jogos com três ou mais facções, fazer com que uma delas faça propostas de alianças ao jogador ou a outra facção, com o objetivo de derrotar um oponente em comum mais poderoso.

Figura 6 – Negociação com facção inimiga em Galactic Civilizations II.

3.1.4 - Aprendizagem

Uma das maiores fraquezas dos sistemas de IA dos jogos RTS atuais está na incapacidade delas de aprenderem com a experiência. Jogadores humanos conseguem facilmente depois de algumas partidas descobrir pontos fracos em suas estratégias ou na de seus oponentes. Os métodos atuais de aprendizagem de máquina são inadequados para o domínio dos jogos RTS e precisam ser adaptados, ou novas técnicas criadas.

3.1.5 – Gerenciamento de Recursos

Saber gerenciar os recursos é essencial para um jogador ou IA de um RTS. Um bom jogador terá uma tática em mente e saberá quantos trabalhadores terá que alocar de

modo que consiga coletar recursos suficientes para aplicar sua tática. Do mesmo modo, uma boa IA para o jogo deve saber fazer o mesmo, e deve ser inteligente o suficiente para conseguir alterar o fluxo de coleta em caso de necessidade de mudança de tática.

3.1.6 - Pathfinding

Pathfinding consiste em encontrar o caminho mais curto de um ponto do mapa a outro. A maioria dos jogos RTS já resolve esse problema automaticamente, bastando para o jogador apenas clicar na posição do mapa em que ele deseja mover suas unidades. Existem diversos algoritmos de busca que garantem encontrar a melhor solução para o *pathfinding*, caso ela exista, entretanto, a implementação desses algoritmos se torna mais complicada no domínio dos jogos RTS devido aos seguintes fatores:

- Ao contrário dos jogos de estratégia baseados em turnos, nos jogos RTS a ação acontece em tempo real, e por isso, o computador não possui muito tempo para calcular a solução do *pathfinding*. É um requisito essencial dos jogos RTS que assim que o jogador ordene uma unidade a se mover, ela o faça imediatamente.
- Nas partidas desses tipos de jogos geralmente existem dezenas ou centenas de unidades se movendo no mapa simultaneamente. Um algoritmo que seja rápido para uma unidade, não necessariamente será rápido quando aplicado a todas elas.
- Como são feitos para rodarem em computadores domésticos, esses jogos possuem restrições de memória baixas, e que não podem ser totalmente destinadas apenas à tarefa de *pathfinding*

Todos esses fatores tornam muito mais complexo a elaboração de algoritmos de busca para esses jogos, exigindo que os desenvolvedores apliquem otimizações tanto no algoritmo quanto na representação dos dados, de forma a tornar a busca mais eficiente.

3.2 – Considerações Finais

Como vimos neste capítulo, 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 [6] 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. Em outro trabalho, ele clama por mais pesquisa nessa área e cita a necessidade de criação de um simulador de RTS de código livre, que possa ser usado como uma plataforma de pesquisa e *benchmarking* para novas técnicas de IA nessa área. [7]

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, consequentemente, 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.

Para comprovar essa tese de que o foco no multiplayer prejudica o desenvolvimento da IA desses jogos basta comparar os jogos de estratégia em tempo real com os de estratégia baseado em turnos. Nesses últimos, as partidas costumam demorar várias horas e até dias, motivo esse que inviabiliza a popularização das partidas multi-jogadores neles. Isso faz com que os desenvolvedores desses jogos de estratégia baseado em turnos invistam muito mais tempo refinando a IA, pois elas são fundamentais para um bom *singleplayer* que possa divertir por muito tempo.

Um dos desenvolvedores de Galactic Civilizations II, famoso jogo de estratégia em turnos reconhecido por sua forte IA, examinou essa questão de como a presença de uma opção multiplayer pode prejudicar algumas funcionalidades da porção *singleplayer* do jogo [8]. Em seu tópico no fórum oficial da Stardock, produtora do jogo, ele demonstra como Civilization IV, que segundo ele foi a melhor implementação multiplayer de um jogo de estratégia em turnos, aparentemente teve várias *features* cortadas do *singleplayer* por causa da presença do multiplayer.

4 - Simuladores

Vimos no capítulo anterior que os jogos RTS fornecem um domínio bastante interessante para a pesquisa de novas técnicas de IA em tempo real, vimos também a necessidade da existência de um simulador de RTS de código livre que pudesse servir como teste e *benchmark* para essas técnicas. Nesse capítulo, fortaleceremos essa justificativa da existência do simulador, logo depois faremos um estudo dos principais simuladores existentes, e em seguida, serão elicitados os motivos que levaram a criação do simulador RTSCup.

4.1 – Necessidade de um Simulador

Michael Buro cita os seguintes motivos para justificar a necessidade de criação de um simulador de RTS para estudo de problemas de IA em tempo real: [9]

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.

4.2 – Estudo dos Simuladores

4.2.1 - ORTS

O Open Real-Time-Strategy (ORTS) é um ambiente de programação para o estudo de problemas de IA em tempo real, como *pathfinding, scheduling*, informação imperfeita e planejamento no domínio dos jogos RTS. Ele foi desenvolvido por Michael Buro, da Universidade de Alberta no Canadá, junto com a contribuição de dezenas de outras pessoas.

O ORTS é um projeto *open-source* e seu protocolo de comunicação é aberto, o que permite com que seus usuários conectem quaisquer clientes que eles queiram ao simulador. Isso é possível pois o ORTS implementa uma arquitetura cliente-servidor em que apenas as partes da simulação que atualmente estão visíveis são mandadas para os jogadores. Isso evita com que clientes maliciosos tentem revelar todo o estado do jogo, e assim obter uma vantagem desleal na disputa.

O ORTS provê uma engine de jogos RTS totalmente customizável. Seus usuários podem descrever o jogo que eles querem simular a partir de arquivos scripts que definem todos os tipos de unidades e construções do jogo. Esses scripts são carregados pelo servidor e executados. A partir dai, os clientes podem se conectar ao servidor e gerar ações para cada objeto do jogo. O servidor manda visões da simulação para esses clientes e recebe as ações de todos os objetos jogadores, sendo esse *loop* executado várias vezes por segundo. Há ainda a possibilidade de se conectar com um cliente 3D, que renderiza o mundo usando OpenGL, e permite ao usuário enviar comandos usando o mouse e o teclado.

Figura 7 – Cliente 3D do ORTS

O ORTS tem sido usado com sucesso em duas competições de IA realizadas em 2006 e 2007. A participação da segunda competição mais que dobrou em relação a primeira, demonstrando o sucesso do simulador. As competições do ORTS são formadas por vários games, que serão descritos a seguir:

- **Game 1:** É uma jogo *singleplayer* onde o objetivo é coletar a maior quantidade de recursos possíveis em 10 minutos. Nesse game existe 1 *Control Center* e 20 *Workers*, e toda a informação do mapa está disponível aos agentes (informação perfeita). Também há a presença no game de alguns obstáculos móveis (os *Sheeps*), que se movimentam randomicamente. Os desafios de IA enfrentados nesse game são o path-finding cooperativo e a esquiva de obstáculos estáticos e móveis.
- **Game 2:** Nesse game, dois times disputam um jogo onde o objetivo é destruir o maior número de construções inimigas dentro de um intervalo de 15 minutos. Cada time recebe 5 *Control Centers*, 10 *Tanks*, e não é possível treinar unidades adicionais. Toda a informação do mapa está disponível aos agentes (informação perfeita). Os desafios de IA enfrentados neste game são o combate em pequena

escala, o gerenciamento de grupos de unidades e o path-finding cooperativo/adversarial.

- **Game 3:** Esse game é uma simulação completa de um jogo RTS, nele, dois times se enfrentam e o objetivo é destruir todas as construções do adversário. Ao contrário dos outros dois games, o game 3 implementa o Fog of War e por isso os agentes só conseguem visualizar os objetos que estão dentro de seu campo de visão (informação perfeita). Nesse game, é possível construir 3 tipos de construções (*Control Centers*, *Barracks* e *Factories*) e treinar 3 tipos de unidades (*Workers*, *Marines* e *Tanks*). Os desafios de IA enfrentados são vários, entre eles combate, path-finding, gerenciar grupos de unidades, exploração do mapa e alocação de recursos.
- **Game 4:** É semelhante ao Game 2, a única diferença é que os dois times recebem apenas 50 marines cada, e o objetivo é destruir a maior quantidade possível de marines adversários.

4.2.2 - Stratagus

O Stratagus é uma engine multi-plataforma para o desenvolvimento de jogos RTS. Ela inclui suporte para partidas multiplayers pela Internet/LAN ou singleplayers contra o computador. O Stratagus já foi utilizado para a criação de diversos jogos entre eles o Bos Wars, Wargus, Stargus e World Domination. Em 10 de junho de 2007, os desenvolvedores responsáveis pelo projeto anunciaram que iam parar permanentemente o projeto, e que estavam se dedicando ao jogo Bos Wars, que utiliza uma versão modificada da engine.

A principal vantagem do Stratagus é o fato dele poder ser facilmente modificado, permitindo assim a configuração dos vários objetos do jogo. Esse motivo o torna viável como uma opção de ambiente de pesquisa para técnicas de IA.

4.2.3 – Warcraft World Editor

Lançado em julho de 2002 pela Blizzard Entertainment, Warcraft III é um dos mais bem sucedidos jogos de estratégia em tempo real, com vendas iniciais de 4,5

milhões de unidades. Apesar de ser um jogo comercial de código fechado, Warcraft III pode ser usado como um simulador para competições e testes de técnicas de Inteligência Artificial graças a seu poderoso World Editor, um programa gráfico que vem junto com o CD do jogo, e possibilita a alteração de vários aspectos do mesmo, tais como as unidades, construções, magias, eventos e até a inteligência artificial dos oponentes.

O World Editor é formado por vários módulos, entre eles:

Terrain Editor: Provê uma interface 3D onde é possível criar ou editar um mapa do jogo. Esse módulo permite editar todo o terreno de um mapa (criando-se florestas, zonas elevadas, oceanos e rios), além de posicionar unidades, construções e outros objetos (*doodads*). O Terrain Module também permite definir regiões (áreas do mapa que irão provocar algum evento, a ser definido no Trigger Editor).

Trigger Editor: É o módulo mais avançado e poderoso do World Editor, ele permite alterar praticamente qualquer aspecto do jogo. É na verdade uma linguagem de programação simples, construída especialmente para ser de fácil acesso à novatos (mesmo aqueles sem experiência em programação), mas poderosa o suficiente para satisfazer até mesmo os usuários mais avançados. Toda a programação é feita visualmente através da criação de *Triggers*, unidades lógicas do mapa compostas por 3 partes: Eventos, Condições e Ações.

- **Eventos:** São ocasiões que podem levar o *Trigger* a ser ativado, o World Editor define vários eventos pré-definidos que podem ser escolhidos para o *Trigger*, alguns deles são: Um tempo X de jogo passou, um tipo X de Unidade entrou em uma Região Y, uma unidade X morreu, etc.
- **Condições:** São condições que devem ser atendidas para que o *Trigger* ocorra, por exemplo, dado que o evento unidade X ocorreu, o *Trigger* só vai realizar a ação caso a unidade X pertença ao *Player* 1.
- **Ações:** São as ações que o *Trigger* deverá executar caso seu Evento ocorra e as condições sejam atendidas. O World Editor possui mais de 300 ações pré-definidas, algumas delas são imprimir um texto na tela, mandar uma unidade atacar outra unidade, alterar o valor de alguma variável, entre outras.

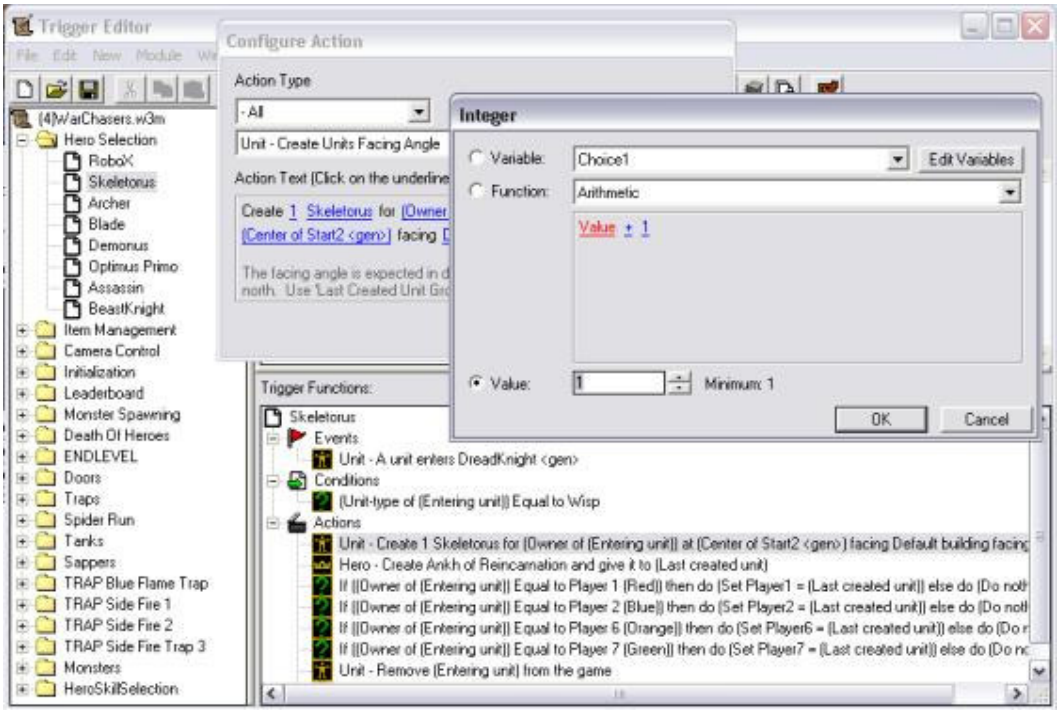

Figura 8 – Programação Visual no Trigger Editor do Warcraft III.

Sound Editor: Permite alterar as músicas e efeitos sonoros do jogo.

Object Editor: Permite criar novos objetos ou alterar os atributos de um já existente. Um objeto do jogo pode ser uma unidade, uma construção, um item, um objeto destrutível (*destructible*), um objeto de enfeite (*doodad*), uma habilidade ou um *upgrade*. É junto com o *Trigger* Editor, um dos módulos mais interessantes e usados do editor, permitindo customizar completamente o jogo, e servindo de base para a maioria dos mapas customizados encontrados na Internet.

Campaign Editor: Permite agrupar vários mapas, telas de *loading*, interfaces e dados customizados para a criação de uma campanha, igual às campanhas originais do jogo.

AI Editor: Provê uma interface visual onde é possível criar uma Inteligência Artificial que irá comandar o desenvolvimento das tropas e estratégias de ataque. O AI Editor permite customizar vários aspectos da Inteligência Artificial do oponente tais como a prioridade na escolha dos heróis, a prioridade das construções e unidades a serem construídas, além de permitir definir grupos de ataques formados por determinadas quantidades de certos tipos de unidades, e outras variáveis. O AI Editor possui também uma área onde é possível configurar um ambiente de teste da IA, definindo o mapa a ser rodado, um multiplicador representando quantas vezes mais

rápido vai rodar que o jogo normal (muito útil para teste), e definindo a quantidade de *players* e seus respectivos scripts de IA.

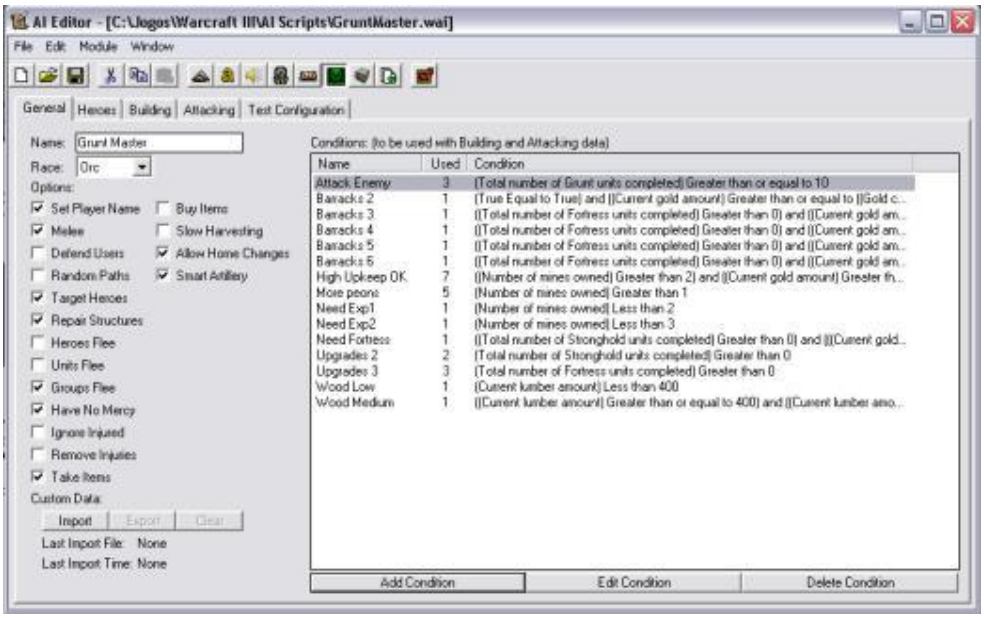

Figura 9 – Exemplo de IA criada no AI Editor do Warcraft III.

As vantagens para se usar o Warcraft III como simulador de RTS para competições de IA são várias, entre elas:

- Por ser um jogo comercial de sucesso, a qualidade gráfica e estabilidade do Warcraft III é superior a de qualquer outro simulador de RTS.
- Editar o terreno e a configuração de um jogo ou mapa é extremamente fácil com o World Editor, pois o mesmo fornece uma interface visual de fácil entendimento.
- Por ser um jogo bastante popular, existem diversos sites e comunidades dedicadas a criação de mapas customizados e modificações (mods) para Warcraft III, além de uma grande quantidade de documentação sobre o uso do World Editor. Alguns dos sites mais famosos sobre modding para Warcraft III são o WC3Campaigns [http://www.wc3campaigns.net/] e o HiveWorkshop [http://www.hiveworkshop.com/].
- Possui uma incrível quantidade de arte presente no jogo e pronta para ser usada no World Editor. São 810 unidades (todas modeladas e animadas), 273 itens, 246 objetos destrutíveis, 469 doodads, 773 habilidades e 83 upgrades.

- Como foi lançado em 2002, Warcraft III não exige um computador muito poderoso para rodar, principalmente se comparado aos computadores e placas de vídeo atuais.

Entre as desvantagens, podemos citar:

- Vários dos desafios de IA como movimentação das tropas, path-finding, entre outros, já estão resolvidos e não podem ser modificados, diminuindo o seu valor como uma plataforma de estudo e teste de técnicas de IA.
- O AI Editor é bastante limitado, não fornecendo muitas opções de customização. Toda lógica mais complicada teria que ser feita através de Triggers. O problema é que algumas das ações dos Triggers são inviáveis para uma competição de IA, por exemplo, pode-se criar um Trigger que mata todas as unidades do adversário. Teria que ser criada uma lista ações permitidas para o desenvolvimento da IA para competição e os Triggers escritos teriam que ser checados para verificar se não usam nenhuma ação proibida antes da realização da competição.
- Os Triggers só podem ser programados visualmente no World Editor, ou por meio de uma linguagem de script conhecida como JASS, usada somente no Warcraft III, fato esse que forçaria o desenvolvedor a aprender uma linguagem nova somente para usar na competição.
- O Warcraft III é um produto comercial de código fechado, qualquer mudança nele que estivesse além do alcance do World Editor não poderia ser feita, há não ser por meios de hacks, o que é bastante complicado e não valeria o esforço.

4.3 – O Simulador RTSCup

O RTSCup surgiu como uma evolução do simulador JARTS (Java Real Time Strategy). 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*.

O JARTS foi criado pois segundo seus autores, os simuladores existentes como o ORTS e o Stratagus não atendiam os dois requisitos principais, que eram a simplicidade e uma boa documentação [10][11]. Segundo eles, a tarefa de compilar e rodar o ORTS se mostrou um processo extremamente complicado. Já o Stratagus não possuía nenhuma documentação exemplificando como criar um jogo, ao invés disso, seus próprios criadores sugeriam que quem estivesse interessado usar baixasse o código-fonte de um jogo já pronto e usasse como *template*.

5 - RTSCup

Este capítulo irá apresentar um breve resumo da arquitetura usada na implementação do RTSCup, o simulador de jogos de estratégia em tempo real criado na Universidade Federal de Pernambuco, e utilizado nas competições de Inteligência Artificial da disciplina de Agentes Autônomos.

5.1 – Estrutura do Simulador

O RTSCup é formado por 3 módulos distintos:

- Kernel: É o servidor central do RTSCup. O Kernel é responsável por realizar toda a simulação do jogo, e enviar mensagens contendo todo o estado atual da simulação aos clientes conectados.
- Agents: São vários programas responsáveis por realizar todas as ações de cada Agente do jogo (as unidades e construções). Os usuários do RTSCup são responsáveis pela elaboração dos agentes.
- Viewer: É um componente gráfico usado para visualizar o ambiente de jogo.

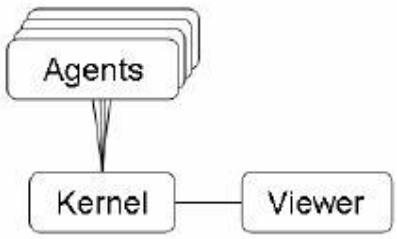

Figura 10 – Arquitetura do RTSCup

5.2 – Progresso de Simulação

A simulação de um jogo no RTSCup ocorre em dois passos: A Inicialização e o Progresso. A fase de Inicialização é o primeiro passo e ocorre somente uma vez, já a fase de Progresso, é repetida constantemente até o final do jogo.

Inicialização: Nessa fase, o Kernel é inicializado e lê as informações da simulação a partir de dois arquivos de configuração XML. O primeiro deles, o arquivo

de configuração do jogo, é responsável por descrever todas as entidades que podem fazer parte do jogo, essas entidades podem ser unidades, construções, recursos ou obstáculos. Já o segundo arquivo, o de configuração do mapa, descreve as posições de cada entidade no mundo virtual da simulação. Após ler esses dois arquivos, o Kernel cria uma representação interna do jogo em sua memória e está pronto para receber conexões dos clientes (Agentes e Viewers). Quando todos os clientes se conectam, o Kernel envia a representação do jogo para cada um desses clientes, designa um Agente para cada unidade e construção do mundo virtual e inicializa a simulação.

Progresso: A fase de Progresso é constantemente repetida logo após a inicialização do jogo. A cada ciclo da fase de progresso, os módulos envolvidos na simulação executam as seguintes tarefas:

1 O Kernel envia informações de visão para cada um dos clientes Agentes.

2 Cada Agente do RTSCup envia uma ação de comando para o Kernel.

3 O Kernel envia as ações de comando para cada um dos seus sub-simuladores.

4 Os sub-simuladores atualizam o estado do mundo virtual do Kernel.

5 O Kernel integra esses estados recebidos e os manda para o Viewer.

6 O Kernel avança o tempo de jogo e volta par ao passo 1.

Cada ciclo desse executa em um intervalo de tempo fixo, pré-definido no arquivo de configuração do jogo. Esse intervalo de tempo pode ser modificado, de modo a permitir aos agentes terem um tempo maior ou menor para calcularem suas ações.

Esse ciclo descrito é repetido constantemente, até que um critério de fim de jogo seja atendido. Esse critério pode variar, a depender do jogo. Em alguns jogos pode ser um limite de tempo, em outros pode ser quando todas as unidades de um jogador forem destruídas.

5.3 – Representação do Mundo

O Kernel modela o mundo da simulação como um conjunto de objetos, podendo estes serem virtuais ou reais. Objetos virtuais são aqueles que não possuem uma

representação no mundo, como por exemplo, o Mapa em si. Já os objetos reais possuem uma representação e participam constantemente da simulação, são representados pelas unidades, construções, recursos e obstáculos. Todos os objetos possuem atributos específicos e os objetos reais possuem um ID para identificá-los.

5.4 – Protocolo de Comunicação

O RTSCup foi construído sobre o protocolo UDP (User Datagram Protocol) e toda a comunicação entre o Kernel, os Agentes e o Viewer é feito através dele. O UDP foi escolhido pois ele é mais rápido que o TCP, já que não precisa garantir que os pacotes não se percam e cheguem na ordem correta

5.5 – Protocolo de Mensagens

O Protocolo de Mensagens do RTSCup define um conjunto bem definido de mensagens, usadas para realizar a comunicação entre o Kernel, os Agentes e o Viewer. A seguinte tabela define todas as mensagens do protocolo:

Valor	Cabeçalho	Uso
	Direcionadas ao Kernel:	
1	AK_CONNECT	Para requisitar uma conexão ao Kernel.
2	AK_ACKNOWLEDGE	Para informar que recebeu a mensagem KA_CONNECT_OK.
3	AK_MOVE	Para se mover a uma outra posição.
4	AK_BUILD	Para construir uma construção.
5	AK_FIX	Para reparar uma construção.
6	AK_COLLECT	Para coletar recursos de uma fonte.
7	AK_DELIVER	Para entregar os recursos à uma construção.
8	AK_ATTACK	Para submeter um ataque ao inimigo.
9	AK_CURE	Para curar uma unidade amiga.
10	AK_CONVERT	Para converter uma unidade inimiga.
11	AK_REST	Para parar a atividade em andamento.
12	AK_TRAIN	Para treinar uma nova unidade de um tipo.
13	AK_RESEARCH	Para pesquisar a tecnologia fornecida.
	Vindas do Kernel:	
14	KA_CONNECT_OK	Para informar o sucesso de uma conexão.
15	KA_CONNECT_ERROR	Para informar a falha de uma conexão.
16	KA_SENSE	Para enviar o estado do jogo.

5.6 - Detecção de Colisão

Em alguns jogos convencionais, o teste de colisão entre dois objetos consiste em simplesmente checar se as posições finais desses dois objetos se sobrepõem, em caso positivo, eles estão em colisão e alguma correção deverá ser feita. Esse teste funciona bem em jogos, porém, pode apresentar problemas no caso em que os objetos estão se movendo muito rapidamente, visto que nesse caso, os objetos podem um ultrapassarem os outros, e ambos terminarem uma posição de não colisão.

Uma solução para esse problema seria subdividir o movimento do objeto em vários movimentos menores, e fazer um teste de colisão para cada um desses movimentos, essa solução, no entanto, pode acabar gastando muito tempo de processamento, a depender do número de subdivisões.

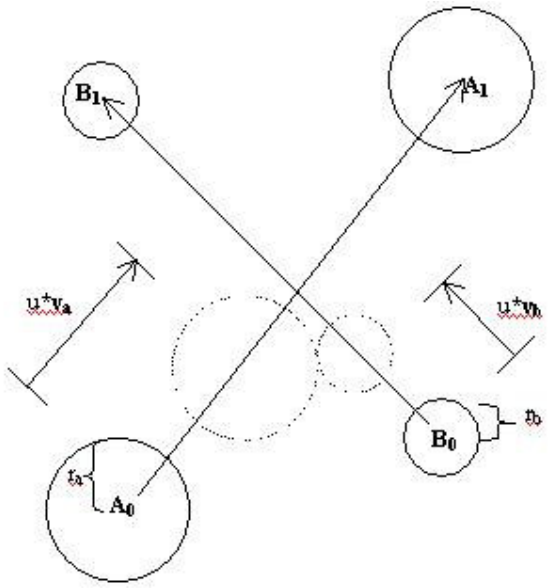

Figura 11 – Colisão entre as trajetórias de duas esferas.

Ao contrário da maioria dos jogos, que rodam a 30 ou 60 frames por segundo, o RTSCup permite definir intervalos de tempo entre os frames bem maiores, com o objetivo de dar um tempo maior para os agentes desenvolvidos fazerem seus cálculos; como conseqüência disso, o movimento dos agentes em um loop é bem maior, tornando inviável um teste de colisão onde se leva em conta apenas as posições finais. Por esse motivo, o RTSCup utiliza o método de sweep tests, ou teste de varredura, como descrito

no artigo *Simple Intersection Test for Games* por Miguel Gomez [12]. Esse teste é capaz de detectar uma colisão entre as duas trajetórias de dois objetos.

6 – Propostas de Melhorias

A versão atual do RTSCup apresentava alguns problemas, tanto técnicos quanto de usabilidade, além disso, existem várias funcionalidades que podem ser adicionadas de forma a melhorar a qualidade do simulador. Os problemas encontrados e as propostas para melhoria do RTSCup serão descritos nesta seção:

Detecção de Colisão: Vários desenvolvedores de Agentes do RTSCup relataram problemas com o sistema de colisão. Muitas vezes dois objetos se colidiam, mesmo quando não deviam, e ficavam presos um no outro, tornando-se necessário o uso de gambiarras para evitar ou sair desse estado. A implementação do sistema de colisão deve ser revista de forma a consertar esses bugs encontrados, livrando os desenvolvedores de agentes dessas preocupações.

Problemas no UDP: O RTSCup vinha sofrendo alguns problemas devido ao fato de usar o protocolo UDP para a comunicação. Um dos problemas ocorria quando se mandava mensagens muito grandes, o que acabava resultando em erros, uma solução tentada, foi a de partir uma mensagem grande em várias outras pequenas, porém, os erros persistiram o que acabou causando o adiamento da competição do Game 2 duas vezes. Outro problema era que o código de comunicação do RTSCup estava se tornando extremamente grande e complexo devido ao uso do UDP, além disso, cada aplicativo do RTSCup (um tipo de agente, servidor ou viewer) tinha que especificar 2 portas diferentes uns dos outros, o que aumentava o número de configurações e podia ser incômodo para os desenvolvedores.

Uma solução para o problema do UDP seria trocar o protocolo pelo TCP. O TCP é um protocolo de comunicação de mais alto nível, e que, apesar de mais lento, possui uma série de vantagens comparada ao UDP, como por exemplo, garantir que os pacotes não se percam e cheguem na ordem em que foram mandados. O TCP tem sido usado em diversos jogos RTS de sucesso como Warcraft III [13], o que garante sua viabilidade. Além disso, a API do TCP para Java é muito mais fácil de usar do que a do UDP e como o RTSCup foi feito para rodar localmente ou em LAN, latência não deve ser uma preocupação.

Usabilidade: Um dos maiores problemas de usabilidade no RTSCup é o processo para testar os Agentes desenvolvidos. Para realizar tal tarefa, é necessário

rodar um processo para ligar o Servidor, outro processo para ligar o Viewer, e mais um processo para cada tipo de agente em cada time do jogo. Isso não é um grave problema quando se está testando um game simples como o Game1, onde só existe um Time e somente um tipo de Agente (Worker), porém, em Games mais complexos, como por exemplo o Game3, haveria no mínimo 2 times, cada um com um Agente para os Workers, os Tanks e os Command Centers, isso resultaria em um total de 6 agentes para os 2 times, obrigando o desenvolvedor a ter que ligar e desligar 8 processos toda vez que qualquer mudança fosse realizada em um dos agentes.

Desempenho: Devido a algum problema de implementação, a versão atual do RTSCup rodava de uma forma extremamente lenta. Para a realização dos testes de desempenho, foi utilizada a seguinte configuração:

- Computador: Athlon 64 3700, 2Gb de RAM, placa de vídeo Radeon X1800XT 256MB.
- Simulador: Servidor rodando com um grande intervalo de tempo entre os frames da simulação (500 ms).
- Mapa: Mapa simples com apenas 1 unidade, 1 recurso e 120 obstáculos.
- Agentes: O agente que não faz nada, somente fica parado.

Mesmo com essas configurações de jogo, as mais simples possíveis, o RTSCup apresentou um desempenho inaceitável, pois assim que o visualizador do mapa era ligado todo o computador ficava lento, com a CPU gastando 100% do processamento, fato esse que evidenciava que o maior problema estava na implementação do visualizador.

Outro problema de desempenho encontrado foi que o servidor constantemente enviava a cada um dos clientes todas as informações da simulação, incluindo dados que nunca mudavam como o poder de ataque de cada unidade do mapa, isso fazia com que grandes capacidades de processamento fossem gastas transferindo informações duplicadas via socket.

Editor de Mapas: Editar manualmente os arquivos de configuração XML dos mapas do RTSCup é uma tarefa extremamente trabalhosa, uma ferramenta visual que pudesse automatizar esse processo seria extremamente útil. Uma solução para esse problema seria a criação de um Editor de Mapas onde o desenvolvedor pudesse

visualizar o mapa, clicar com um mouse na posição onde ele deseja inserir ou remover um objeto e ver o resultado imediatamente. Um Editor de Mapas agilizaria a criação de mapas, possibilitando ao desenvolvedor de Agentes a criar diversos cenários específicos onde seria possível testar seus agentes.

Fog of War: A adição do Fog of War aumenta drasticamente a complexidade tática dos jogos de estratégia pois obriga o jogador a ter que explorar o mapa e a ter que constantemente tentar adivinhar o que o oponente está fazendo baseado nessas explorações.

No RTSCup, todo o mapa e as unidades são revelados aos agentes, o que limita a utilidade do simulador como um ambiente de teste de técnicas de IA como exploração. Por esse motivo, é muito importante que a próxima versão do RTSCup ofereça suporta à Fog of War, escondendo dos Agentes todos os objetos que estão fora da visão de alcance dele.

Game 3: Um dos objetivos do RTSCup é o de permitir a realização de uma competição semelhante à do ORTS. Para isso, é necessário a implementação do Game 3, que ainda não foi feito. Mais do que uma simples configuração de jogo, o Game 3 vai exigir novas funcionalidades no servidor como por exemplo, quando uma unidade for treinada ela deve nascer junto da construção que a treinou; essa posição entretanto, pode estar ocupada e o servidor terá que varrer outras posições até encontrar uma que esteja livre e o mais perto possível da construção. Uma outra funcionalidade parecida seria a de checar se a posição escolhida pelo trabalhador para construir uma construção está livre.

Além dessas funcionalidades, também seria necessário implementar várias das mensagens restantes que ainda não foram implementadas totalmente no RTSCup, como o AK_BUILD, AK_TRAIN, AK_RESEARCH e AK_FIX.

7 - Resultados

Foram realizadas neste trabalho, as seguintes melhorias no código do RTSCup.

Detecção de Colisão: O algoritmo de colisão foi revisto e os agentes do mapa, não mais ficam presos uns aos outros.

Problemas no UDP: O código de comunicação do RTSCup foi totalmente refeito, e agora está usando o protocolo TCP ao invés do UDP, o que trouxe algumas vantagens, como a eliminação dos bugs, e um código muito mais limpo, com uma redução drástica no número de classes e linhas de código.

Usabilidade: Para resolver os problemas de usabilidade, foi criada uma GUI chamada RTSCup Launcher, capaz de lançar todos os programas envolvidos na simulação de uma só vez. Mais detalhes sobre o RTSCup Launcher se encontram no Apêndice deste trabalho.

Desempenho: O Viewer foi implementado e agora ele está rodando mais que 10 vezes mais rápido. Enquanto que antes, o simulador gastava 100% do processamento da CPU do computador de teste com um intervalo de loop de 500 ms, agora, mesmo rodando com loops de 50 ms, o uso da CPU não passa de 60%.

Outra otimização realizada foi a de evitar que o Servidor constantemente envie informações duplicadas para os clientes. Na nova versão, o servidor envia todos os dados somente na inicialização da simulação, e a partir dai, no loop de execução, só envia os dados das unidades que mudam como os pontos de vida e a posição x-y. Os obstáculos do mapa nem são enviados, pois eles estão sempre na mesma posição. Os recursos só são enviados quando acontece alguma coleta nele, o que modifica sua carga atual.

Editor de Mapas: Foi criado um Editor de Mapas para permitir a fácil edição dos arquivos de configurações XML referentes aos mapas. Mais detalhes sobre o Editor de Mapas podem ser encontrados no Apêndice deste trabalho.

Fog of War: O Fog of War foi implementado no Servidor, fazendo com que os agentes só enxerguem os objetos dentro do seu campo de visão. O Fog of War também é exibido no Viewer para melhor exibição do jogo.

Game 3: O Game 3, inicialmente proposto como um dos objetivos deste trabalho, não pode ser implementado devido ao estado de imaturidade que o simulador se encontrava. Não era possível implementar um game complexo como o Game 3 sem antes resolver os problemas de comunicação, de desempenho e de detecção de colisões.

8 – Trabalhos Futuros

Há diversas melhorias que ainda podem ser aplicadas no RTSCup, algumas das idéias para uma futura versão são as seguintes:

- A implementação do Game 3, que não pôde ser concluído nesse trabalho.
- Estender o Editor de Mapas para poder também editar os arquivos de configuração do jogo.
- Melhorias no visual do Viewer, incluindo a criação de um Viewer 3D.
- Desenvolver uma ferramenta que gere *reports* automáticos ao final de cada partida, de modo que eles possam ser utilizados como benchmarks para avaliação das soluções de Agentes desenvolvidas.

Referências

[1] ROBOCUP FEDERATION. **RoboCup Brief Introduction**. Acesso em: 18/03/2008. Disponível em: <http://www.robocup.org/Intro.htm>.

[2] BURO, Michael; AHA, David; STURTEVANT, Nathan; CORRUBLE, Vincent. **Complex Video Game AI Competitions at AIIDE'06**. Acesso em: 18/03/2008. Disponível em: <http://www.cs.ualberta.ca/~mburo/orts/rts-aiide.pdf>.

[3] VIEIRA, Vicente; WEBER, Renan; MOURA, José. **Agentes Autônomos**. Acesso em 18/03/2008. Disponível em: <http://www.cin.ufpe.br/~vvf/rtscup/files/docs/RTSCup.ppt>.

[4] GERYK, Bruce. **A History of Real-Time Strategy Games**. Acesso em 25/06/2008. Disponível em: <http://www.gamespot.com/gamespot/features/all/real_time/>.

[5] BLOOMBERG NEWS. **Samsung, SK Telecom, Shinhan Sponsor South Korean Alien Killers**. Acesso em 25/06/2008. Disponível em: <http://www.bloomberg.com/apps/news?pid=email_us&refer=asia&sid=a2JvzciDnpB4>.

[6] BURO, Michael. **Call for AI Research in RTS Games**. Acesso em 25/06/2008. Disponível em: <http://www.cs.ualberta.ca/~mburo/ps/RTS-AAAI04.pdf>.

[7] BURO, Michael; FURTAK, Timothy. **ON THE DEVELOPMENT OF A FREE RTS GAME ENGINE**. Acesso em 25/06/2008. Disponível em: <http://www.cs.ualberta.ca/~mburo/ps/orts05.pdf>.

[8] STARDOCK CORPORATION. **Galactic Civilizations: The case for no multiplayer**. Acesso em 25/06/2008. Disponível em: <http://forums.stardock.com/98074>.

[9] BURO, Michael; FURTAK, Timothy. **RTS Games and Real–Time AI Research**. Acesso em 25/06/2008. Disponível em: <http://www.cs.ualberta.ca/~mburo/ps/BRIMS-04.pdf>.

[10] WEBER, Renan. **JARTS – Java Real Time Strategy**. Acesso em 25/06/2008. Disponível em: <http://www.cin.ufpe.br/~tg/2006-1/rtw.pdf>.

[11] MOURA, José. **UMA ESTRATEGIA EFICIENTE DE COLETA MULTIAGENTE PARA JOGOS RTS**. Acesso em 25/06/2008. Disponível em: <http://www.cin.ufpe.br/~tg/2006-1/jcmj.pdf>.

[12] GOMEZ, Miguel. **Simple Intersection Tests For Games**. Acesso em 25/06/2008. Disponível em: <http://www.gamasutra.com/features/19991018/Gomez_1.htm>.

[13] BLIZZARD ENTERTAINMENT. **Blizzard Support**. Acesso em 25/06/2008. Disponível em: <http://us.blizzard.com/support/article.xml?articleId=21109&rhtml=true>.

Apêndice

Guia do Usuário: RTSCup Editor

O RTSCup Editor é um programa simples, criado para permitir a fácil edição dos arquivos xml de configuração do RTSCup. A versão atual do programa permite somente a edição do arquivo de configuração do mapa, pois este é o arquivo mais difícil de editar manualmente. Versões futuras do RTSCup Editor também permitirão a edição dos arquivos de configuração do jogo.

Para rodar o RTSCup Editor é necessário possuir uma versão 1.5 ou superior do Java. A forma mais simples de rodá-lo é clicando no seguinte link e o executando via Java Web-Start: http://www.cin.ufpe.br/~vcac/rtscup/rtscup-editor.jnlp

Depois de rodar o programa, é necessário carregar o arquivo de configuração do jogo (pois ele contém as unidades e construções que são possíveis de se inserir no mapa), e carregar ou criar um novo mapa. Para fazer isso, basta clicar em File, e logo em seguida clicar em New/Open Map, isto irá abrir uma janela popup onde será possível carregar os dois arquivos. Uma foto da janela será mostrada logo abaixo.

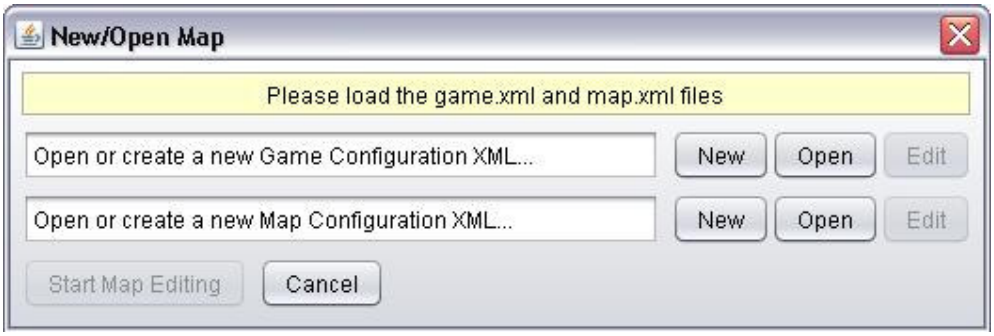

Figura 12 – Janela para carregar o mapa no RTSCup Editor

Depois de carregado os dois arquivos, o programa irá para a tela principal, onde é possível adicionar ou remover objetos do jogo no mapa. A tela principal é dividida em quatro partes, que serão explicadas logo em seguida.

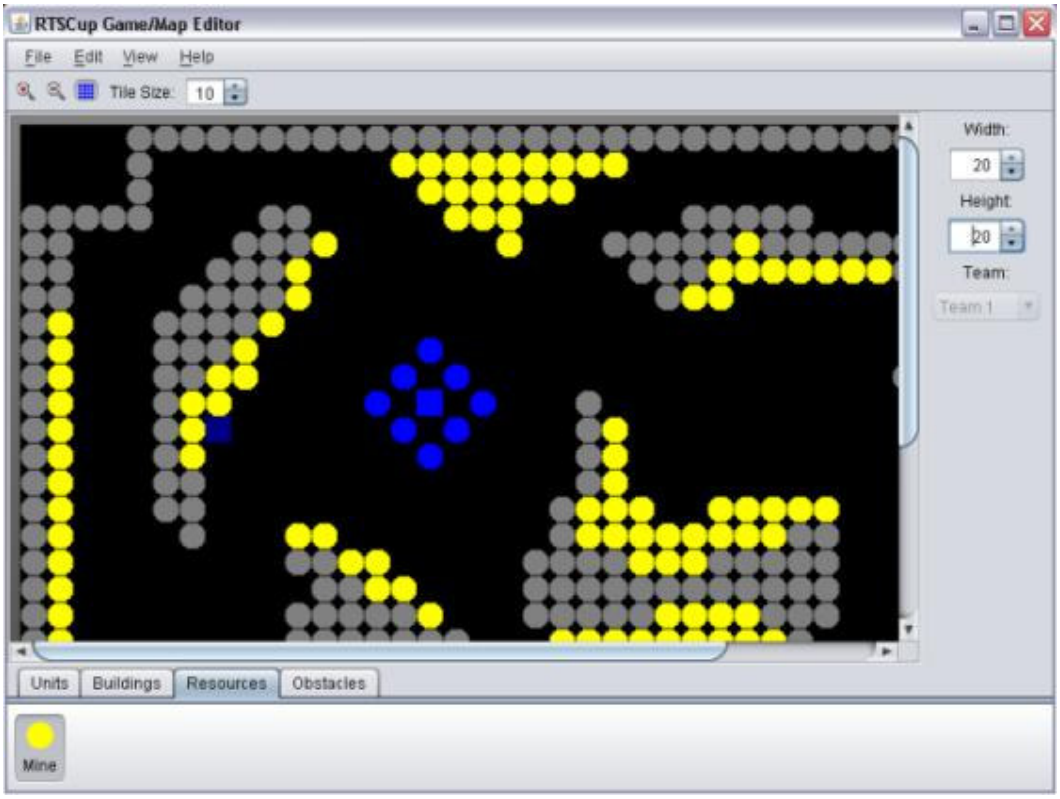

Figura 13 – Tela principal do RTSCup Editor

Toolbar: É a barra de ferramentas na parte de cima da tela, abaixo do menu principal. Possui 3 botões. O primeiro botão (Zoom In, ícone de uma lupa com +) permite aumentar o zoom no mapa. O segundo botão (Zoom Out, ícone de uma lupa com -) permite diminuir o zoom no mapa. O terceiro botão (Snap to Grid, símbolo de uma grade azul) permite habilitar ou desabilitar o Snap to Grid, uma funcionalidade que quando habilitada faz com que todos os objetos sejam inseridos no mapa em posições ordenadas como uma grade, o tamanho de cada posição da grade é definido no campo ao lado com o nome de Tile Size.

Objects Panel: É o painel na parte de baixo da tela. Possui quatro abas, uma para cada tipo de objeto do jogo (Unidades, Construções, Recursos e Obstáculos). Ao clicar em uma aba, será listado logo abaixo todos os objetos do tipo correspondente, podendo um item ser selecionado para inserir no mapa.

Properties Panel: É o painel à direita. Permite modificar todos os atributos de um objeto antes de embirá-lo no mapa. Os atributos possíveis de serem modificados são a largura e altura do objeto, e o time ao qual ele pertence (válido somente para unidades e construções).

Map Component: É o componente central da tela, onde é mostrado o mapa e todos os seus objetos. A inserção de objetos no mapa é feita através do clique com o botão esquerdo do mouse e a remoção, feita através do clique com o botão direito. Um retângulo semi-transparente é exibido no mapa para mostrar onde o objeto será inserido ou removido, de acordo com a posição do mouse e o tamanho do objeto. A cor desse retângulo é azul quando se pode inserir o objeto, e vermelho quando não se pode, pois nesse caso já existe um outro objeto no mapa que vai colidir com ele naquela posição.

Guia do Usuário: RTSCup Launcher

O RTSCup Launcher é uma programa simples, criado para facilitar a inicialização do servidor, do visualizador e dos agentes. Ele provê uma interface onde é possível carregar as configurações do servidor, do jogo, do mapa, e dos agentes, permitindo executar todos eles através de um simples clique em um botão. As configurações usadas são persistidas em um arquivo temporário no computador, de modo que não é necessário configurar o programa novamente da próxima vez que ele for executado, isso facilita bastante o desenvolvimento de agentes, pois para cada mudança que o desenvolvedor implementar, ele não vai precisar desligar e iniciar vários processos, somente o Launcher.

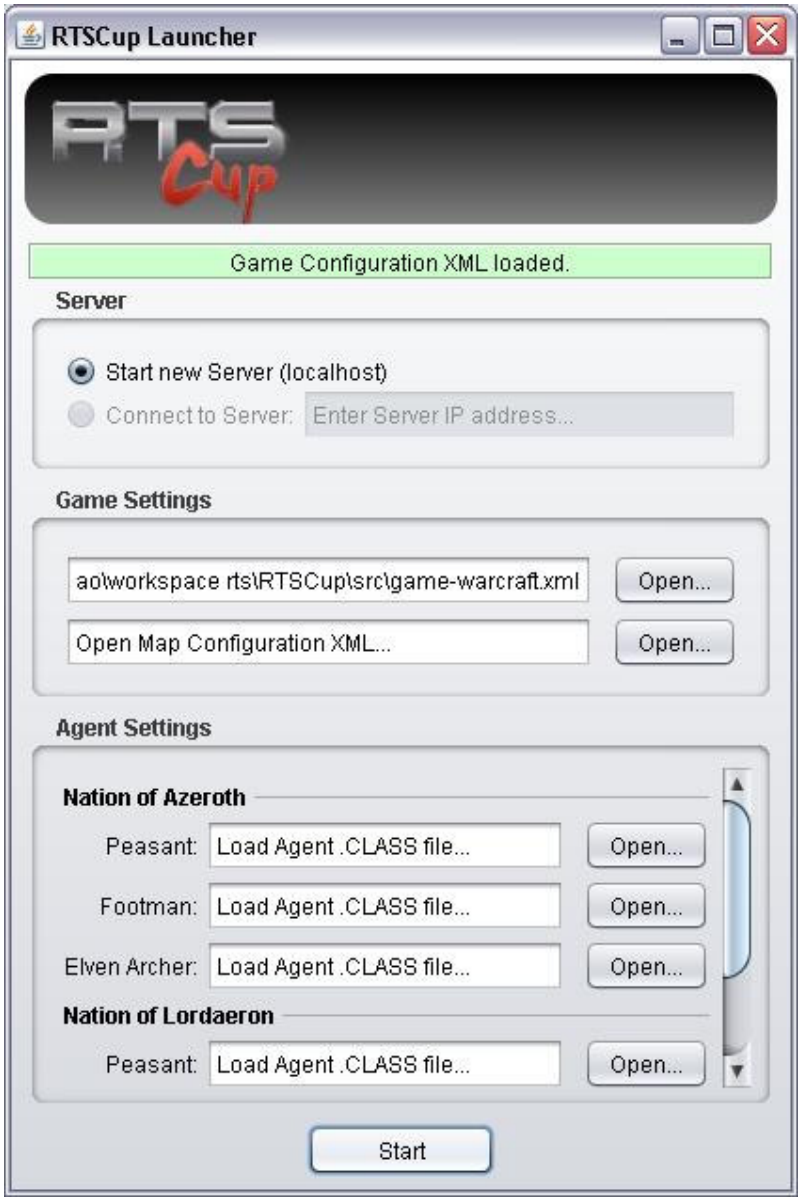

Figura 14 – Tela principal do RTSCup Launcher