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

JARTS – JAVA REAL TIME STRATEGY

TRABALHO DE GRADUAÇÃO

Renan Távora Weber <rtw@cin.ufpe.br>

Orientador: Geber Lisboa Ramalho <glr@cin.ufpe.br>
Co-orientadora: Patrícia Cabral de Azevedo Restelli Tedesco <pcart@cin.ufpe.br>

Recife, outubro de 2006

"Nothing in this world is to be feared...

only understood.

— Marie Curie

À memória do saudoso “Chapeleiro”.

AGRADECIMENTOS

“Gratitude is the memory of the heart”

Aos meus pais, minha irmã, minha madrinha e minhas avós de forma especial, pelo apoio desde o início e a cada momento.

A Geber e Patrícia pela oportunidade, e por todo apoio dado durante este período, mesmo nos momentos mais críticos na construção deste trabalho.

A Jynx Playware e todos os seus integrantes, especialmente Scylla, pelo apoio e incentivo, e ainda pela compreensão nas folgas concedidas, sem as quais elas o trabalho não seria concluído.

A Manifesto Game Studios pela cessão da sala que foi fundamental para o desenvolvimento desta ferramenta. Em especial a Túlio, por colaborar com as discussões referentes à modelagem. Principalmente por evitar o uso do padrão de projeto *Adapter*, além de ser um excelente *bot* de *Call of Duty 2*.

A André, Zé Carlos, Zobby, Vilmar e Túlio, “secadores oficiais”, e suas “macumbas” infalíveis durantes os jogos do Sport.

A Livar, Rafael e Allan (este menos, por ficar me atrapalhando pra falar do Santa) pelas dicas dadas e pelas dúvidas tiradas durante a elaboração deste documento. Também por servirem como referencial de progresso do trabalho.

A todos os amigos e colegas de faculdades, principalmente aos integrantes do “grupo default” porque sem eles, para ajudar a desenvolver os projetos de todas as disciplinas, não conseguiria chegar até aqui.

Principalmente a Vicente e José Carlos que foram parceiros no desenvolvimento deste trabalho, e no apoio à disciplina de Agentes Autônomos.

E por fim ao Centro de Informática e todos os professores, pela estrutura fornecida, pelo conhecimento transmitido e exemplo de sucesso e dedicação.

RESUMO

Este trabalho tem o intuito de propor uma nova solução para simuladores de Inteligência Artificial, em particular para jogos RTS e com ênfase em combate. Apresenta uma visão geral sobre este estilo de jogo e o papel da IA neles. Também são mostrados os problemas que envolvem o combate multiagente. Analisando as soluções já existentes. Por fim é descrita em detalhes a proposta do novo simulador, o JARTS.

Palavras-chave: Inteligência Artificial, *Real-time Strategy*, Estratégia em tempo real, Combate, Simulador

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

"Everything that has a beginning …"
— Matrix

Os jogos de eletrônicos (computador e videogame) têm evoluído bastante ao longo dos últimos anos e a cada dia vêm conseguindo mais adeptos. Do seu início árduo e desacreditado, da época do "*Tennis For Two*" [Pong] 1958, onde era necessário ocupar uma sala inteira para instalar um computador, aos tempos de "*Halo 2"* [Halo2], 2004, um dos maiores sucessos do Xbox[1], houve uma explosão de aficionados por todo o mundo. Este jogo foi responsável por uma falsa epidemia de gripe no Japão, pois milhares de pessoas faltaram aos seus empregos alegando doença para poderem comprar o jogo no dia do seu lançamento, 9 de novembro de 2004.

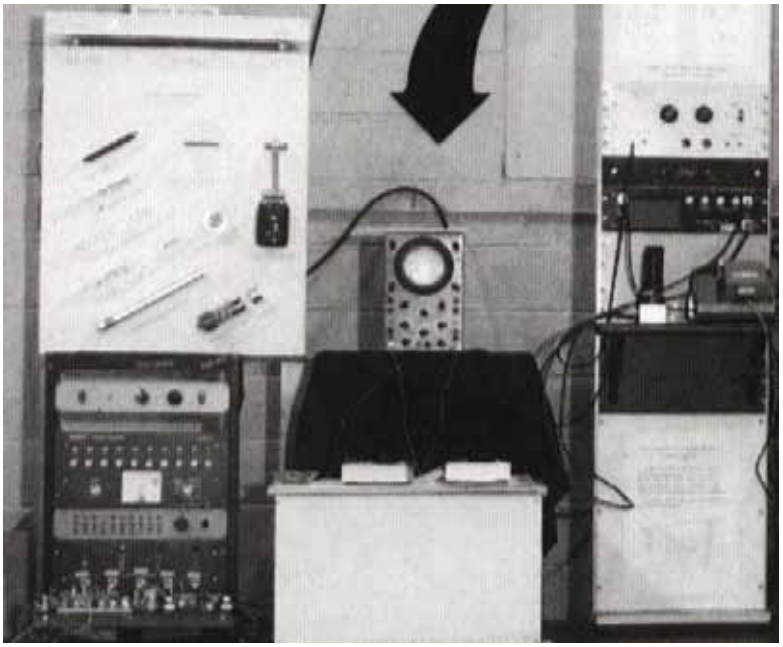


Ilustração 1.1 - Computador onde era jogado o "*Tennis for Two*"

[1] Videogame, também conhecido como console de segunda geração da Microsoft.

Esse enorme crescimento do mercado de jogos foi devido a diversas causas, e o mais comumente apontado, é a incrível evolução dos gráficos, como pode ser visto logo a seguir na comparação do “*Tennis for Two”* e do “*Halo 2”*.

Ilustração 1.2 - Comparação do *"Tennis for Two"* e *"Halo 2"*

Um dos grandes responsáveis por este “boom” no mercado de jogos, são os jogos de estratégia em tempo real, mais conhecidos como RTS (do inglês *Real Time Strategy*). RTS é um gênero de jogos de estratégia normalmente focado em combates militares. Em jogos como *Warcraft* [Warcraft] e *Empire Earth* [EE], dá-se ao jogador o controle de uma civilização, onde é possível usar suas forças militares para derrotar os oponentes. Ações comuns desses jogos são a construção de prédios, pesquisas de novas tecnologias, e, principalmente, o controle das unidades durante o combate.

Desde o início, jogos deste tipo tiveram milhares de fãs, mas com a popularização de internet e velocidades de conexão cada vez mais rápidas, estes número não param de crescer. Além disso, o modo multiplayer dos jogos permite que as pessoas conectadas joguem entre si, trazendo um atrativo adicional para o jogo.

Com o aprimoramento dos hardwares 3D e as qualidades gráficas chegando perto do seu limite, onde não constituem mais um diferencial, mais uma revolução vem acontecendo. A Inteligência Artificial vem ganhando cada vez mais importância, tornando-se um dos fatores críticos de sucesso de um jogo, determinando quando serão *bestsellers* e quando serão fracassos [Rabin 2002].

Esse crescente aumento da importância da Inteligência Artificial no sucesso dos jogos, faz com que seja também cada vez maior a necessidade de ferramentas que dêem suporte ao seu desenvolvimento. Por exemplo, a parte gráfica dos jogos já é bem apoiada por

diversos motores 3D, etc. Na IA para jogos este movimento ainda está se iniciando. Estão começando as pesquisas de motores de IA, equivalentes aos motores gráficos, e também pesquisas de simuladores, que são ferramentas bastante úteis para a indústria de jogos, pois auxiliam o desenvolvimento e validação dos comportamentos projetados para os agentes.

Os jogos RTS necessitam de um suporte de IA dos mais complexos dentre todos os estilos de jogos, pois envolvem inúmeras unidades, inúmeras técnicas de IA aplicadas, e inúmeros fatores a serem tratados. Por exemplo, vai do planejamento estratégicos das ações ao controle detalhado de cada uma das unidades. Por isto é importantíssimo um simulador, focado neste estilo, que auxilie o desenvolvimento deste tipo de jogo. E também que possa criar/recriar diversas situações que sirvam como forma de testar e validar, de forma mais automatizada, se o que foi projetado está realmente sendo executado.

1.1 OBJETIVOS

O trabalho desenvolvido nesta monografia está relacionado às ferramentas de auxilio ao desenvolvimento e estudo de Inteligência Artificial, em particular simuladores voltados ao problema do combate multiagentes. Então o objetivo deste trabalho pode ser subdividido em:

- Discutir os desafios relacionados ao problema de combate multiagentes em jogos RTS, analisando as soluções existentes.
- Analisar os principais problemas a serem tratados, e como eles foram abordados;
- Desenvolver um simulador que seja capaz atender aos requisitos necessários para apoiar a construção da IA em jogos RTS.

1.2 ESTRUTURA DO TRABALHO

Este documento está dividido em cinco capítulos. São eles:

- **IA e RTS:** apresenta uma visão geral da Inteligência Artificial, em particular a aplicada a jogos, bem como sua importância nos jogos RTS.
- **Combate:** apresenta os problemas enfrentados pela IA no âmbito do combate em jogos RTS.
- **Simuladores:** apresenta as soluções existentes na área de simuladores de jogos RTS e combate.

- **JARTS:** apresenta a nova solução proposta para simuladores de IA focados em combate multiagente.
- **Conclusão e Trabalhos Futuros:** conclui o trabalho, apresenta as contribuições, explicita as dificuldades encontradas, e apresentas pontos de continuidade.

CAPÍTULO 2

IA E RTS

“Ready to work!”
— Peasant, Warcraft III

A IA aplicada a jogos é complexa, pois tem que dividir o processamento com outras rotinas do jogo, como a parte gráfica. Nos jogos RTS este problema é ainda mais crítico, visto que este tipo de jogo possui necessita de um apoio, de IA, dos difícies. Isto ocorre por haver diversos desafios a serem enfrentados: o mundo (ambiente do jogo) mudar constantemente (simulação em tempo real), o mundo ser parcialmente observável, o número de estados de cada unidade é quase infinito, assim como a quantidade de ações que podem ser tomadas. E, a tomada de decisão, baseada nestes fatores, tem que ser muito rápida de modo a não prejudicar o andamento do jogo. Tudo isto faz com que, diferentemente de outros jogos de estratégia, como jogos de tabuleiro (Xadrez, Dama e Gamão), onde o computador já consegue jogar tão bem quanto um jogador humano, o desempenho da IA em jogos RTS é relativamente ruim, pois projetar uma IA em ambientes tão complexos é uma tarefa bastante difícil.

Além destes aspectos, um dos maiores desafios para a IA de um jogo de estratégia em tempo real é a criação de um “bom” oponente, ou seja, que consiga ser desafiante para o jogador humano. Para a Inteligência Artificial ser “bom”, é, na grande maioria dos casos, fazer o que um jogador humano faria: analisar o mapa, definir as tarefas, e executar a estratégia [Davis 1999].

2.1 INTELIGÊNCIA ARTIFICIAL E JOGOS

A questão “O que é Inteligência Artificial?” não é simples de ser respondida. Em um dicionário o termo “inteligência artificial” será definido como: “A habilidade do computador ou outra máquina de executar atividades que normalmente são tidas como as que requerem inteligência” [Bourg e Seemann 2004]. Uma das definições mais aceitas e usadas, dentre inúmeras existentes, é uma definição acadêmica proposta por Russel e Norvig [Russell e Norvig 2003], que diz:

IA é a criação de programas de computador que simulam ações e pensamentos como um humano, assim como agir e pensar racionalmente.

2.1.1 IA Aplicada a Jogos

Dando continuidade a definição de Inteligência Artificial citada anteriormente, *Game AI* (IA aplicadas a jogos) é o trecho de código do jogo responsável por fazer com que elementos controlados pelo computador "pareçam" tomar decisões inteligentes. O termo "parecer" é de fundamental importância, porque representa a principal diferença entre a IA tradicional e a aplicada a jogos. Nesta última objetiva-se exclusivamente o modo de ação do sistema, e não o modo de pensamento [Schwab 2004]. Para um jogador o grande atrativo é o jogo em si, ele não quer saber como o computador controla seus inimigos, ou quais as técnicas que são utilizadas, o importante é que seja realista e desafiador.

Outra diferença entre a IA tradicional e a aplicada a jogos, é que nesta última são sempre considerados tempo e custo computacional. Já a segunda preza por estes fatores para não comprometer o desempenho do jogo como um todo. Além disto, o nível de dificuldade deve ser aceitável (que permita ao jogador progredir sem facilidades nem dificuldades extremas), para não se ter um inimigo "perfeito". Por exemplo, se Quake, um jogo estilo FPS (*First Person Shooter*), tivesse sido desenvolvido visando obter o comportamento ideal (sempre buscando a melhor solução) ao entrar no jogo o jogador se depararia com a seguinte situação: um inimigo entraria correndo no ambiente, se agacharia de forma perfeita, miraria perfeitamente, além de "conhecer" exatamente a posição onde o jogador se encontraria. Dessa forma seria impossível derrotá-lo, e o jogo não seria muito divertido. A *Game AI* visa obter do computador uma performance "humana". É importante que o agente erre em alguns momentos, fique sem munição, perca tiros, e tudo mais que o faça parecer um oponente humano.

Ilustração 2.1 - Screenshot de Quake 2

Um exemplo de "técnica" exclusiva, e talvez a mais usada [Bourg e Seemann 2004], pela *Game AI* é o *cheating*, (do inglês trapaça). Como o próprio nome diz é uma artimanha,

usada para dar vantagem ao computador, dando a idéia de que o computador está tendo um comportamento mais inteligente, e obtendo um desempenho melhor. Por exemplo, em um jogo de simulação de guerra o time controlado pelo computador pode ter acesso a todas as informações do oponente humano – localização de suas bases, tipos, números e localização de suas unidades, etc. – isso tudo sem ter que mandar exploradores para coletar estas informações, como o jogador humano teria que fazer.

O uso do *cheating* é muito comum e ajuda o computador a ter uma vantagem sobre jogadores humanos. Embora útil, pois na maioria das vezes consegue dar a impressão de inteligência (principalmente para distinguir níveis de dificuldades diferentes), é preciso tomar cuidado com esta técnica, pois se ficar óbvio que o jogador está sendo "trapaceado", ele provavelmente perderá interesse no jogo por achar que seus esforços serão inúteis. Outra forma de causar desinteresse é dar poder demais ao computador, tornando-o imbatível. O *cheating* é uma técnica bastante útil, mas tem que ser muito bem balanceado para criar um desafio superável e manter o jogo interessante e divertido.

Acredita-se que o futuro da IA aplicada a jogos é a utilização do aprendizado [Schwab 2004]. Ao invés dos NPCs (*Non Player Characters*, ou seja, unidades controladas pelo computador) terem comportamentos predefinidos quando o jogo é desenvolvido, eles poderão aprender e se adaptar ao estilo de cada jogador. Isso resulta num jogo que evolui juntamente com o jogador, aumentando bastante seu "tempo de vida". Já existem jogos se aventurando nesta área, *Black & White* [B&W], por exemplo, utiliza conceitos de aprendizado. Nele o jogador desempenha o papel de um Deus, que pode ser representado no mundo como uma criatura (um animal: vaca, macaco, tigre ou lobo), e ela precisa ser ensinada a fazer diversas coisas, desde procurar comida até elementos mais complexos magias bem elaboradas.

2.2 RTS

Real Time Strategy (RTS) ou Estratégia em Tempo Real, estão entre os pioneiros da IA aplicada a jogos. Isto não é surpreendente visto que tais jogos não precisam ir muito longe na computação gráfica, e requerem uma boa IA para serem jogáveis. A Inteligência Artificial nesta categoria é particularmente desafiante, pois necessita de uma sofisticada abordagem para as unidades, e uma ainda mais complexa para as decisões estratégicas e táticas do computador. Tais jogos são, atualmente, resultado de um longo período de evolução e refinamento, com enormes avanços em todas as áreas [Laursen e Nielsen 2005].

Neste gênero todos, ou pelo menos a grande maioria dos jogos, envolve guerra e disputa de territórios. Existem também alguns jogos que podem ser configurados para considerar também conquistas econômicas e tecnológicas, mas este recurso não é muito popular entre os jogadores. Nos jogos RTS, tanto o jogador como o computador tem que controlar uma civilização (sociedade ou povoado) e lidar com diversas tarefas como a coleta e gerenciamento de recursos, a construção e a manutenção da sua base ou cidade, a criação e a manutenção de seu exército, a evolução de sua tecnologia, além do gerenciamento das decisões estratégicas, táticas, etc. Estes jogos oferecem uma enorme variedade de problemas fundamentais que são abordados nas pesquisas de Inteligência Artificial (*pathfinding*, coordenação multiagentes, planejamento, etc.), diferentemente de outros estilos de jogos [Buro 2003].

Coleta de recursos juntamente com reconhecimento do território, são os primeiros problemas a serem enfrentados em um jogo de estratégia. No começo o mapa não é conhecido, as fontes de recursos não foram localizadas, por isso é fundamental explorar o ambiente em busca de informações. Ao serem encontrados (os recursos) começa a coleta propriamente dita. É preciso também coodernar esta atividade para dizer quem (qual unidade) irá coletar e onde, além de tentar reduzir a ociosidade das unidades ao mínimo possível. Caso duas unidades cheguem ao mesmo ponto é preciso fazer uma negociação visando definir quem tem a prioridade para executar a tarefa,

Ilustração 2.2 - *Fog of War* em *Civilization*[Civilization], necessidade de exploração do mapa

Gerenciamento de recursos é outra dificuldade imposta logo no início, e permanece durante toda a partida. É um dos elementos, senão o mais, importante deste gênero. Ele é responsável pela tomada de decisão de quando e qual a quantidade de recursos que precisa se coletada, quanto pode ser gasto, em que gastar, etc. É fundamental para um planejamento bem elaborado, para garantir que as ações projetadas serão realizadas com

sucesso no momento certo. Por exemplo, em *Age of Empires II* [AoK] para se evoluir para a Idade Imperial é preciso ter, em alguns casos, 1500 pontos de comida (um dos tipos de recursos do jogo).

Se as unidades, ou seja, os agentes, que são controlados no jogo (trabalhadores e militares, como arqueiros), também forem considerados recursos, o gerenciamento de recursos fica ainda mais complicado. Será preciso decidir quando é necessário criar mais unidades, qual unidade deve ser criada (civil ou militar) e ainda qual subtipo, no caso de serem militares, de unidade: arqueiros, cavaleiros, ou infantaria.

Outro problema a ser levado em conta é a análise de terreno. À medida que o mapa é explorado e as informações são obtidas, é importante analisar esses dados para saber são os melhores caminhos, aqueles com menos ameaças, isto é importante tanto para a criação de novas construções (prédios como quartéis, fortes, fazendas, etc.), como para posicionamento de tropas para ataque e defesa.

Por fim um dos problemas mais desafiadores, que será discutido em mais detalhes mais a frente, é a Tomada de Decisão Estratégico-Tática. Esta consiste em definir a postura das unidades ao longo da partida, desde o nível macro, da civilização como um todo, ao micro, descrevendo ações bem detalhadas de cada unidade. Por exemplo, decidir quem irá atacar e em qual momento isto será feito, assim como a forma como isto será feito (caminho a ser seguido até o objetivo, o que cada unidade irá atacar, etc.).

Ilustração 2.3 - *Battle for Middle Earth II*[BFME II], exército a caminho do ataque.

Jogar um RTS já é desafiante. Ainda mais desafiante é a criação de sistemas autônomos, em tempo real, capazes de ter um desempenho semelhante ao de humanos nesse domínio. Por tudo isso que a Inteligência Artificial vem galgando cada vez mais espaço, e mostrando uma grande importância nos jogos, especialmente nos deste tipo.

2.3 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os conceitos de Inteligência Artificial aplicada a jogos, e foram mostradas algumas de suas diferenças para a IA tradicional. Foram também exemplificadas algumas situações de como as técnicas de IA são utilizadas nos jogos.

Apresentou-se também o gênero de jogos RTS, mostrando algumas de suas características, como a mecânica do jogo, e em particular, o combate, que será abordado em mais detalhes no próximo capítulo. Foi também explicitada a relação da IA com estes jogos, sua importância, e alguns exemplos de como ela está sendo usada neste estilo.

CAPÍTULO 3

COMBATE

"Time for killing!"
— Grunt, WarCraft III

RTS muitas vezes são definidos da seguinte forma, *"Jogos Real Time Strategy são simuladores de guerra onde diversas facções batalham em um mundo virtual"* [Laursen e Nielsen 2005]. A partir dessa definição é fácil perceber a importância do combate neste estilo de jogo. Na realidade, grande parte dos jogos explora elementos de confronto como um dos principais componentes da jogabilidade da imersão. Mas nos RTS, assim como os FPS, o confronto como forma de batalha e luta fica bem mais evidente que em outras categorias. O combate consiste de inúmeras unidades brigando umas com as outras.

O combate, por ser um dos fatores mais desafiantes para os jogadores de RTS, pois a vitória no jogo depende disto, passa a merecer um tratamento especial do ponto de vista em sua Inteligência Artificial. Uma excelente IA, para a coleta e gerenciamento de recursos, pode passar despercebida caso o desempenho do computador no combate não seja bem executado, entretanto uma extraordinária IA de combate pode "esconder", se necessário, falhas em outros elementos do jogo mantendo um alto nível de jogabilidade.

3.1 FATORES ENVOLVIDOS

Ao se implementar um conjunto de técnicas, para tratar o problema do combate em um RTS, é necessário levar em conta vários fatores na hora da tomada de decisão (qual tática deve ser usada para contra-atacar, por exemplo) que acontece em tempo real. Para jogadores humanos essa tomada de decisão é feita de forma muita simples e rápida, bastante intuitiva. No entanto para o computador simular este tipo de tomada de decisões, com restrições de tempo e processamento, é um problema bastante complexo.

3.1.1 Variedade de Unidades

Em jogos RTS cada time (civilização ou facção) possui uma enorme variedade de unidades. Esses atributos têm que ser levados em consideração na hora do combate, para,

por exemplo: decidir qual a ordem de ataque à unidade inimiga? Qual das unidades é prioritária? Qual deve ser atacada antes? Os principais atributos a ser considerados são:

- **Poder de Ataque** – é a quantidade de dano que a unidade pode causar a outra cada vez que executa o seu ataque. Para exemplificar, considerou-se a situação: a unidade está sendo atacada por duas outras, um arqueiro e um elefante (elementos do jogo *Age of Empires II*), baseado em seus poderes de ataque é possível calcular o que será mais vantajoso, atacar o arqueiro e derrotá-lo com um golpe ou focar no elefante, pois tem um poder de ataque bastante alto, enquanto é atacado pelo arqueiro.

- **Tempo de Retardo** – é quantidade de tempo que a unidade leva para executar um segundo ataque, ou seja, o tempo entre um ataque e outro. Pode significar a vantagem para unidades mais fracas. Suponha que o arqueiro demore 0,1 segundo entre seus ataques, e tenha um Poder de Ataque de 10. Seu oponente, uma trebuchet (outro elemento de *Age of Empires II*), possua um ataque de 150, entretanto demore 2 segundos entre um ataque e outro. Num confronto direto o arqueiro levaria certa vantagem (desconsiderando seus pontos de vida, que será tratado a seguir), mesmo sendo mais “fraco”, pois seu ataque total ao longo dos 2 segundos seria de 200, 50 a mais que seu oponente.

- **Pontos de Vida** – quantidade de pontos de danos que a unidade pode receber até ser destruída, como as unidades são diferentes (suas armaduras, veículos, etc.) elas não podem morrer recebendo a mesma quantidade de ataques. No exemplo anterior viu-se que o arqueiro levaria certa vantagem contra a trebuchet, mas isto no jogo (*Age of Empires*) não acontece de fato. Embora a trebuchet demore bastante para atacar, ela é bastante resistente e possui um Poder de Ataque altíssimo. Por mais que o arqueiro a atacasse ela não seria derrotada durante o Tempo de Retardo, e ao executar seu ataque o derrotaria com um único golpe.

Poder de Ataque, Tempo de Retardo, e Pontos de Vida são valores fixos e predefinidos desde a concepção do jogo, durante um processo chamado balanceamento, que tem como objetivo deixá-lo equilibrado, sem grandes disparidades. Associados a estes fatores há outros que são específicos de cada unidade, independente do seu tipo, e eles variam ao longo do tempo de acordo com ações dentro do jogo. São eles:

- **Temperatura** – quantidade de tempo que deve passar para que a unidade possa atacar novamente [Demyen 2004]. Bastante semelhante ao Tempo de Retardo, a Temperatura representa o tempo para o próximo ataque (é variável – terá um valor caso um ataque tenha acabado de ser executado, e outro caso não tenha executado nenhum). Voltando à trebuchet, seu Tempo de Retardo seria de 2 segundos, mas caso ela tenha feito um ataque 1 segundo atrás ela estará a apenas 1 do próximo, ou seja, seu valor de Temperatura é de 1 segundo.

- **Pontos de Vida Restantes** – é a quantidade de "vida" restante para uma determinada unidade, ou seja, quantos ataques ela ainda pode sofrer até que seja derrotada [Demyen 2004]. Por exemplo, uma unidade recém criada pode ter 1000 pontos de vida restantes, e uma outra do mesmo tipo, pode possuir apenas 10, caso já tenha sofrido alguns ataques.

3.1.2 Variedade de Danos

Os ataques de cada unidade podem causar danos com valores diferentes, baseado em várias circunstâncias, ou até de forma aleatória dependendo do jogo. Por exemplo, cada unidade responde de forma diferente a cada tipo de ataque sofrido, sofrendo mais danos um determinado tipo de oponente. Em *Age of Empires II* [AoK] isto é bastante evidente, por exemplo, é impraticável destruir um castelo – que possui uma forte defesa – sem utilizar armas de cerco, como a trebuchet. Estes tipos de armas existem exatamente para demolição de construções, pois têm um Poder de Ataque bastante alto. Entretanto, possuem um Tempo de Retardo grande, e sua locomoção é lenta, o que as torna extremamente fracas contra unidades móveis, como as de infantaria.

Outro fator que algumas vezes é levado em conta no momento de calcular o dano, causado por um ataque, é o ambiente. Ataques mais próximos, causam mais danos que os distantes. E já existem jogos como *Sins of a Solar Empire* [Sins], que pretendem tratar o clima como um fator de grande influência nas batalhas. Por exemplo, em um cenário de deserto, ao se iniciar uma tempestade de areia, ficaria inviável ataques de longa distância, forçando tanto o jogador como o computador a assimilar este outro elemento e modificar as decisões tomadas. Isto tudo visando tornar os RTS simuladores de guerra cada vez mais realistas, e com isso aumentando ainda mais os problemas a serem tratados pela *Game AI*.

3.1.3 Movimentação de Unidades

O problema da movimentação é complexo. Ele afeta diversos outros fatores, como os anteriormente citados, e é crucial para a decisão de quem ataca saber se é possível chegar ao ponto do ataque, quanto tempo levaria a movimentação, a velocidade das unidades, além de qual caminho é preciso seguir. Este último é um dos problemas clássicos em IA, conhecido como *pathfinding*, tem por objetivo encontrar o melhor caminho entre dois pontos. Atualmente no RTS, ele é feito considerando-se apenas elementos do ambiente, como árvores, pedras e outras unidades, das quais se deve desviar. Entretanto já há uma nova tendência, iniciada nos jogos FPS como Halo 2 [Halo 2], de se acrescentar ainda mais "inteligência" a este tipo de solução, não apenas observando elementos fixos do ambiente. Neste jogo, os inimigos não apenas calculam suas rotas baseados no ambiente, como também levam em consideração o posicionamento do seu oponente, no caso o jogador, buscando além do melhor caminho, o caminho mais seguro. Isto torna o *pathfinding* um problema bem mais complexo, pois ele terá que ser feito a cada momento, fazendo um acompanhamento do caminho, já que dependendo das alterações no mundo a unidade pode acabar saindo de uma rota segura e caindo em uma armadilha.

3.1.4 Grupos

Grupos são vistos como pelotões, onde várias unidades são agrupadas, e na grande maioria dos RTS comercias há a possibilidade de criá-los. Há também uma nova forma de encarar isso, em *Battle for the Middle Earth* [BFME II] por exemplo, só existe o conceito de pelotões não havendo a possibilidade de criar unidades isoladas. Neste jogo sempre que se manda criar infantaria, são criadas cinco novas unidades. Isto acontece também em alguns momentos em *Age of Empire III*[AoE3].

A idéia de pelotões além de tentar reproduzir a realidade das batalhas militares, é bastante útil para a solução de problemas, pois a tomada de decisão é feita para o grupo como um todo e não para cada unidade, com isso a coordenação dos agentes (unidades) pode ser melhorada, principalmente do ponto de vista de custo computacional. Analisando-se alguns jogos desta categoria é possível perceber que as unidades realmente se comportam em grupos, ao se selecionar várias unidades (do mesmo tipo) e mandá-las a outro ponto do mapa, é fácil ver que elas se reorganizam. Na maioria das vezes elas tentam reduzir a "área de contato" do grupo (ficando em fila, por exemplo), o que facilita o *pathfinding*, pois gera menos colisões. Seguindo a mesma idéia, quando são selecionadas unidades de tipos diferentes para se movimentarem no mapa, em alguns jogos como *Age of Empires*, elas se organizam de forma a deixar as mais fracas no centro, ficando mais

protegidas, e todas passam a andar com a velocidade da mais lenta para não se dispersarem (ver Anexo B, Formações).

3.2 ESTRATÉGIAS E TÁTICAS

Outro grande problema, da Inteligência Artificial de jogos RTS é necessidade de se coordenar as unidades para elas sigam determinadas estratégias e táticas ao longo da partida, e não fiquem “perdidas”, com movimentos aleatórios. Esta decisão é tomada baseada em inúmeras variáveis, como tipo de mapa, time ao qual pertence, time do oponente, etc. Além disto, tem-se que identificar as estratégias/táticas que estão sendo utilizadas pelo oponente para tentar se defender delas, ou até analisá-las para descobrir suas fraquezas e contra-atacar.

3.2.1 Estratégia

A escolha de uma boa estratégia é fundamental para a vitória em jogos RTS, tanto para jogadores humanos como os controlados pelo computador. Existem várias definições para o termo “estratégia”, entre elas as que mais se adequam aos jogos RTS são: “Arte militar de escolher onde, quando e com que travar um combate ou uma batalha”, e principalmente “Planejamento total, de como a campanha conseguirá os objetivos” [Ferreira 1999] . No contexto de jogos RTS, “estratégia” envolve uma visão mais abrangente da guerra, onde têm que ser analisados todos os componentes e variáveis envolvidos, para a partir daí, fazer o planejamento e estruturar as ações futuras de cada uma de suas unidades, visando o objetivo final, que é a vitória da partida.

Do ponto de vista da Inteligência Artificial a estratégia é um fator crucial para a tomada de decisão. A sua escolha irá influenciar todas as tomadas de decisões futuras, desde a coleta de recursos, até o local de construção dos prédios, passando pelo posicionamento dos exércitos. Por exemplo, imagine que o computador adotar uma estratégia de tentar evoluir ao máximo suas unidades, para que elas sejam mais fortes e resistentes que as do oponente, ele terá que planejar toda a coleta de recursos de modo a garantir que o preço de cada evolução poderá ser pago. Além disso terá que analisar a árvore de tecnologias do seu time (exemplo este baseado em *Age of Empires II*) para decidir quais evoluções são prioritárias. Agora suponha que durante a execução deste planejamento as unidades do computador passe a ser atacado. Ele está indefeso, pois de acordo com a estratégia os recursos coletados serviriam para os avanços tecnológicos e não para a criação de um exército. Neste momento (na verdade, durante toda a partida) o

computador tem reavaliar as variáveis, levando em consideração o ataque sofrido, para reestruturar sua estratégia e tentar sobreviver ao ataque. E para isso, além dos diversos fatores do seu time e do ambiente, é preciso levar em conta as decisões tomadas pelo oponente. Muitas vezes é preciso prever o futuro, tentar estar um passo a frente do inimigo para ter alguma vantagem. Implementar isso é bastante complexo. Normalmente é colocado no computador um conjunto de regras, que define sua "base de conhecimento de estratégias", para que ele possa tentar prevê-las.

Tudo isto faz dos RTS um grande desafio para a Inteligência Artificial, pois implica na combinação de inúmeras técnicas (grande parte delas pode ser adaptada a algum destes problemas), além de ter que lidar com certa quantidade de restrições, como processamento, tempo, etc. Um exemplo da IA sendo usada na análise estratégica e na tomada de decisão é *Dark Reign,* cujo modelo foi feito para permitir a criação de oponentes que respondam à mudanças em certas circunstâncias [Dark Reign]. Nele acredita-se que se o computador for capaz de recuar e reagrupar quando se sentir ameaçado pelo jogador, ou reconhecer uma fraqueza do inimigo e decidir atacá-lo, isto pode causar um enorme avanço na jogabilidade e na imersão do jogo. Para tanto é preciso identificar os momentos no jogo onde uma decisão estratégica inicial (de alto nível) deve ser seguida, reformulada ou até abandonada, devido a ameaças ou oportunidades surgidas no decorrer da partida.

3.2.2 Tática

O termo tática possui várias definições, dentre elas as mais adequadas ao âmbito dos jogos de computador são: "Parte da arte da guerra que trata de como travar um combate ou uma batalha"; "Meios usados para alcançar um objetivo", e principalmente destacando sua relação com "estratégia": "Táticas são as maneiras como as estratégias são executadas" [Ferreira 1999]. Diferentemente da estratégia, a tática representa uma visão de um confronto específico. Existe uma expressão popular bastante conhecida que diz "ganhou a batalha, mas ainda não ganhou a guerra", que representa de forma clara a distinção entre estratégia e tática. Uma guerra é composta de várias batalhas, estas representam os confrontos, as lutas entre times/facções/exércitos. Em um jogo RTS seria considerado como guerra a partida como um todo e os combates seriam as batalhas. A estratégia é o planejamento que visa vencer a guerra, e para tal ela faz uso de inúmeras táticas, estas que são usadas separadamente em cada batalha.

A tática em um combate define coisas mais concretas, como as formações dos pelotões (discutidas em detalhes no Anexo B), a movimentação, a postura e o

posicionamento das unidades, entre outras coisas. Do ponto de vista de Inteligência Artificial as decisões táticas são mais simples que as estratégicas, pois não requerem uma análise completa de todo o mundo, nem planejamento em longo prazo. Porém também não representam uma solução trivial. Por exemplo, ao ter sua base atacada por vários cavaleiros do exército inimigo, qual a melhor solução a ser tomada? É mais vantagem atacar com os arqueiros que estão próximos, ou mandar voltar os lanceiros mais distantes? No momento do ataque todas as unidades devem ficar próximas ou espalhadas? O computador se defronta com circunstâncias como essa diversas vezes ao longo da partida, e tem que fazer uso de IA para encontrar a melhor resposta.

3.3 CONSIDERAÇÕES FINAIS

Foram analisados alguns elementos que devem ser considerados, pela Inteligência Artificial de um jogo RTS, ao se tratar o problema do combate multiagentes. Cada um deles foi descrito, e foi apresentado um cenário de uso em jogos deste estilo, utilizando *Age of Empires II* [Age2] como referência.

Um dos principais problemas da IA em jogos, a tomada de decisões, foi descrito neste capítulo. Abordou-se estes problemas em dois níveis, o estratégico e o tático, para tal foi preciso, antes, definir estes conceitos.

Através de alguns exemplos, apresentados, tentou-se deixar mais clara a complexidade destes problemas do ponto de vista da IA, bem como sua vital importância para tais jogos. E devido a isto, a necessidade de ferramentas que dêem um melhor suporte ao desenvolvimento de técnicas para esta destinação.

CAPÍTULO 4

SIMULADORES

“I do have work to do.”
— Footman, Warcraft II

Simulação é uma tentativa de modelar situações da vida real em um computador, para que possa ser estudado e visto como o sistema funciona. Alterando variáveis, é possível fazer predições sobre o comportamento do sistema. Simulações de computadores têm sido extremamente utilizadas em áreas como física, biologia, química, e também em informática [Simulation].

Na Inteligência Artificial os simuladores são usados em enorme escala. Isto se deve à necessidade de análise e validação das soluções projetadas. Nos jogos isto também é fundamental. Um simulador pode ser usado de diversas formas: para analisar um aspecto específico, isolando-o dos demais para que estes não interfiram; para analisar um aspecto em novas situações; para repetir condições de simulações prévias; etc. Por exemplo, estudar apenas a movimentação dos agentes, ignorando a exploração do mapa.

Os jogos RTS não fogem à regra, também utilizam simuladores para o desenvolvimento de técnicas e estudos. São largamente usados porque os jogos deste estilo (como visto no Capítulo 2) são dos mais complexos do ponto de vista da Inteligência Artificial. E ter a possibilidade de isolar elementos para estudá-los em separado pode ser de grande ajuda. Por exemplo, se o objeto do estudo for a implementação de um mecanismo de raciocínio para um tanque, não é interessante colocar elementos, desnecessários neste ponto, como coleta de recurso, e evolução através da árvore de tecnologias. Desta forma, com o uso destas ferramentas, é possível analisar “módulos” específicos da Inteligência Artificial aplicada no jogo, sendo mais fácil corrigi-los e melhorá-los.

4.1 REQUISITOS

O uso de simuladores em jogos *Real Time Strategy* é de enorme importância, principalmente para a IA. Entretanto eles devem atender a alguns aspectos fundamentais

para que seu uso não se torne um empecilho. A seguir serão considerados alguns fatores para a análise deste tipo de ferramenta, apenas considerando o foco em simulações no âmbito de Inteligência Artificial, em particular o problema do combate, levando-se em conta o que foi abordado nos capítulos anteriores, e também as dificuldades encontradas durante o desenvolvimento deste trabalho.

4.1.1 Aspectos Gerais

A seguir são descritos três itens que são críticos na implementação da grande maioria dos softwares, e que com os simuladores não é diferente. São eles:

- **Simplicidade**: Simuladores complexos são difíceis de entender e têm uma curva de aprendizado bastante baixa. Muitas vezes tenta-se colocar inúmeras funcionalidades no simulador, que não acrescentam muito à sua utilidade. Em um jogo RTS, o simulador deve servir para validar a IA projetada e desenvolvida, isto visa descobrir se os agentes estão se comportando como desejado, por exemplo. Eles devem ser desenvolvidos da forma mais simples possível, pois o jogo por si só apresenta problemas demais para que se preocupe com coisas fora de seu escopo.

- **Documentação**: Por não serem ferramentas triviais de se usar, é necessário um mínimo de documentação para que sua utilização seja feita da forma mais fácil possível. Como visto antes, simplicidade é importante, mas nem sempre é possível atender as todas as demandas de forma simples e objetiva. Para contornar isto é crucial que um simulador possua uma excelente documentação, mostrando sua estrutura, ensinando seu funcionamento e sua utilização.

4.1.2 Eficiência

Este é também um item bastante genérico e poderia estar junto aos da seção 4.1.1, mas foi destacado devido a sua relação mais direta com o problema de Inteligência Artificial. Como visto no Capítulo 2, jogos *Real Time Strategy* são dos mais custosos computacionalmente. Normalmente acaba restando para a IA cerca de 10% de todo o processamento utilizado em um jogo [Schwab 2004].

Pelo fato de ter que tratar problemas como compartilhamento de CPU além dos próprios problemas da IA, é que a eficiência é fundamental a um simulador. Para os estudos realizados nele, o tempo de resposta não é tão crítico, mas de nada adianta implementar as técnicas num cenário completamente distinto do cenário real. Por exemplo, um *pathfinding*

implementado no simulador demora X milissegundos para calcular uma rota, entretanto no jogo este tempo passar a ser 3X. Todo o estudo feito baseado no valor X acaba sendo perdido.

4.1.3 Diversidade de Unidades

Jogos RTS sempre contam com uma enorme variedade de civilizações, cada uma possuindo suas unidades próprias, umas diferentes das outras. Isto acrescenta bastante à jogabilidade, e também ao “tempo de vida” dos jogos, pois faz com que as pessoas tentem atingir excelência com cada um dos elementos.

Em simuladores de combate também é importante que exista uma gama de unidades para serem controladas. Além de ser mais condizente com os jogos existentes, é fundamental para a elaboração de uma boa IA. A variedade torna possível a criação de estratégias e táticas bem mais complexas. Por exemplo, o uso de Formações para a movimentação e posicionamento de tropas faz muito mais sentido quando se possuem unidades diferenciadas. A formação em caixa, uma das mais comuns nos jogos, perde sua essência, que é de colocar as unidades mais fracas na parte interna (ver Anexo B para mais detalhes sobre técnicas de IA usadas em combate), se todas as unidades forem idênticas.

A ausência de unidades variadas restringe muito o desenvolvimento de uma IA mais completa. Grande parte das tomadas de decisão é abandonada ao se igualar todos os elementos. Como visto no Capítulo 3, decisões como: “quem deve ser atacado antes?”, “qual unidade é mais importante para ser protegida?”, entre outras, que poderiam tornar tanto o jogo, como o estudo da Inteligência Artificial mais interessantes, acabam sendo deixadas de lado.

4.1.4 Visibilidade

A visibilidade, na realidade a falta dela (os agentes não conhecem o todo o mundo, mas apenas o que conseguem “enxergar”), é bastante explorada nos jogos RTS, portanto deve estar presente em seus simuladores. É preciso explorar o mapa inicialmente para conhecê-lo, é preciso fazer patrulhas rotineiramente em busca de alterações no mundo, além de aumentar muito as possibilidades do ponto de vista estratégico/tático.

Um simulador que seja focado no estudo do combate em RTS deve ter uma forma de abordar esta questão. Caso isto seja negligenciado, perde-se muito do ponto de vista da IA, pois uma gama de possibilidades acaba não sendo mais aplicáveis.

4.1.5 Possibilidade de configuração

Todos os elementos analisados até agora são de suma importância, mas parte deles deve ser configurável para que seja possível estudar esses aspectos sob várias perspectivas. A alteração dessas informações deve ser feita de forma simples e rápida, como um arquivo de configurações, por exemplo. Dessa forma bastaria alterá-lo e executar a simulação novamente com as alterações feitas. No caso da visibilidade, por exemplo, é bastante interessante a possibilidade de tratar as duas alternativas, estudar os comportamentos em um mundo inexplorado, e em um mundo com informação perfeita, onde tudo já está à mostra. No momento em que o foco principal não for a questão da visibilidade em si, seria interessante poder “desligá-la” para que não interferisse em outros aspectos, por exemplo. Outro item para o uso dos arquivos de configurações seria na definição das unidades, o que possibilitaria alterar as características de cada uma delas, sendo bastante útil, pois permitiria modificar elementos do jogo e do simulador sem precisar conhecer detalhes da implementação do simulador.

4.1.6 Comunicação

Para qualquer ferramenta voltada para o estudo de Inteligência Artificial, onde existam vários agentes que possam interagir, é fundamental que haja um sistema de comunicação entre eles, ou que sua implementação, por parte do usuário, não seja restringida devido à estrutura da ferramenta.

Em grande parte dos jogos modernos a comunicação multiagentes tem um papel importante, desde jogos de esporte, como FIFA a jogos FPS. Nos RTS esta necessidade é ainda mais evidenciada, pois sempre existirão pelotões se comunicando tanto para movimentação, como para o combate, além das outras unidades. Por isso é fundamental que um simulador de combate de jogos RTS, ao menos, permita a implementação de comunicação multiagentes, caso contrário acabaria restringindo bastante o campo de estudos da IA.

4.2 ESTADO DA ARTE

Atualmente existem alguns trabalhos nesta área, tanto de simuladores de RTS, estes bem abrangentes quase uma reconstrução do jogo, quanto para combate, mais específicos e simplificados. Para a elaboração deste trabalho foram estudados diversos simuladores, jogos e *engines* buscando o que mais se adequasse à proposta inicial, que era criar uma estratégia de combate para jogos RTS. A seguir serão apresentados mais detalhadamente

os que se adequaram mais aos requisitos propostos, se aproximando mais do conceito de simulador de combate em jogos RTS.

4.2.1 Stratagus

O *Stratagus* é uma *engine*, multi-plataforma e *opensource*, para desenvolvimento de jogos RTS [Stratagus]. Diversos jogos foram produzidos utilizando-o, e os que mais se destacam são: Wargus [Wargus] (uma cópia de *WarCraft II* [Warcraft2]), Battle of Survival [BoS] e Magnant [Magnant]. Todos estes jogos compartilham a API provida por esta *engine* [Ponsen 2005].

O Stratagus possui algumas características bastante úteis para a pesquisa de Inteligência Artificial, entre elas destaca-se a possibilidade fácil configuração dos elementos do jogo. O fato de ser facilmente configurável é fundamental, como visto antes. Nesta engine é possível até alterar o visual, por exemplo, desligando-se parcialmente os gráficos, resultando em jogos mais eficientes. Há também neste aspecto um editor de mapa que é de grande utilidade para o estudo da IA, pois é possível usá-lo para testar diversas técnicas em inúmeros cenários.

Outro ponto forte do *Stratagus* é a capacidade gravar partidas, fazendo com que elas possam ser jogadas repetidamente com o mesmo cenário. Isto facilita bastante o uso de aprendizado no desenvolvimento de táticas. Por exemplo, cria-se um banco de jogos, cada um executando uma mesma estratégia de forma diferente. Daí confronta-se o computador com este banco de estratégias, e ele acaba aprendendo a desempenhar tarefas como reconhecer qual foi a que o seu oponente adotou, e dessa forma é possível adotar a contra estratégia para enfrentá-lo.

Ilustração 4.1 - Jogos desenvolvidos usando o *Stratagus*

4.2.2 Robocode

Robocode [Robocode] não é um simulador de RTS, entretanto por ter seu foco em combate, inclusive envolvendo times, foi considerado durante o estudo. O objetivo deste simulador é que os tanques evitem ser atingidos pelo oponente e também evitem colisões com as paredes [Eisenstein 2003]. Além disso, é preciso localizar o adversário e atirar nele.

Neste simulador a questão da visibilidade é bem abordada. Os tanques dispõem de um radar com alcance limitado, e só são capazes de “enxergar” coisas dentro do campo de visão de seu radar. Diferentemente dos jogos RTS onde a área visível é um circulo em volta da unidade, o *Robocode* é diferente, ele leva em conta o local que a unidade está apontando para saber o que é visível para aquele elemento.

Esta ferramenta foi desenvolvida pela IBM para incentivar o estudo tanto da linguagem de programação Java, quanto da Inteligência Artificial. Baseado nele existe uma competição, onde são programados os tanques que competem entre si por diversos rounds – o que morrer menos é o vencedor.

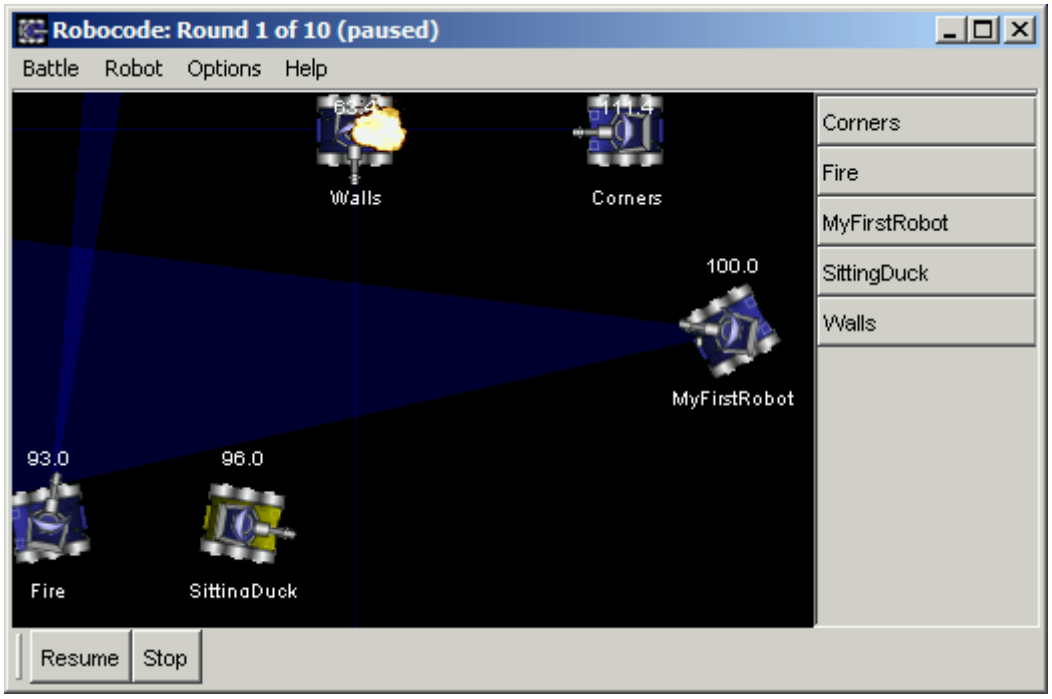


Ilustração 4.2 - *Robocode*, tanques competindo num campo de batalha

4.2.3 Open Real Time Strategy

O ORTS, acrônimo de *Open Real Time Strategy*, é um projeto, ainda em desenvolvimento, de um ambiente para estudo de problemas de IA como *pathfinding*, informação imperfeita e planejamento, no âmbito dos jogos RTS [ORTS]. Ele foi criado para facilitar o estudo da Inteligência Artificial em jogos de estratégia, visto que todos os jogos comerciais deste tipo são softwares fechados, tanto para estudo das técnicas usadas, quanto para a sua modificação.

Ilustração 4.3 - Visões distintas do ORTS, a visão do servidor (*Server*) e a do cliente (*Client*)

O ORTS é um ambiente de programação genérico de um RTS [ORTS], provendo uma infra-estrutura completa para estes jogos, incluindo um servidor e uma versão cliente, com interface gráfica mais elaborada, mostrada na figura anterior. Seus jogos são configuráveis através de scripts, e é possível alterar tamanho do mapa, velocidade das unidades, visibilidade, campo de visão, e também definir detalhadamente os atributos de cada unidade.

Nesta ferramenta não há nenhum comportamento definido para as unidades, como ocorre nos RTS comerciais. Dessa forma o desenvolvedor fica completamente livre para estudá-los e programá-los da maneira que achar correta, não tendo que se preocupar em adaptar comportamentos predefinidos.

O ORTS vem sendo desenvolvido na Universidade de Alberta, no Canadá [ORTS], há alguns anos, e semelhante ao *Robocode*, visto na seção 4.2.2, existe uma competição recente baseada, para estimular o seu uso. Nela existem três tipos de jogos a serem disputados, que são:

- **Game1 –** É uma modalidade *single-player*, ou seja, apenas um jogador. Seu objetivo é coletar a maior quantidade de recursos em um determinado período de tempo (10 minutos normalmente). Acontece em um mapa 32x32, com diversos obstáculos, inclusive móveis, com informação perfeita (os agentes têm conhecimento total do mapa). Dispõe-se de 20 trabalhadores para coletar os recursos e uma base onde eles devem ser entregues.

- **Game2 –** Nesta modalidade existem dois times que se confrontam. Seu objetivo é destruir a maior quantidade de bases inimigas num tempo determinado (15 minutos). Dispõe de um mapa 64x64, também com os mesmos obstáculos, e também com informação perfeita. Cada time possui cinco bases, para serem defendidas, e cada uma delas possui 10 tanques (totalizando 50 tanques). Não há a possibilidade de serem criadas novas unidades.

- **Game3 –** É a modalidade mais completa, chamada de "*a real RTS game*" (um jogo RTS real), pois engloba as duas outras modalidades, e ainda acrescenta novos problemas, como: gerenciamento de recursos, árvore de tecnologias e informação imperfeita. Seu objetivo é destruir todos os

prédios (bases ou quartéis) dos oponentes em determinado limite de tempo (20 minutos). Cada time começa com apenas seis trabalhadores para coletar os recursos, e a partir daí, criar seus exércitos.

4.3 ANÁLISE COMPARATIVA

Nesta seção serão analisadas as ferramentas apresentadas anteriormente. Esta análise é baseada nas facilidades e dificuldades encontradas ao se tentar implementar uma IA para agentes em um combate. Levou-se em conta os fatores propostos na seção 4.1.

4.3.1 Glest, Boson e Crytal Space

O *Glest* e o *Boson* são jogos RTS de código aberto, que poderiam ser adaptados para tornar possível o acoplamento de módulos de IA. Entretanto esta solução não pareceu adequada, pois iria requerer um extenso estudo dos códigos fonte destes jogos para, se descobrir a viabilidade desta solução, e só após este processo tentar fazer tal alteração. Devido a isto o risco de adotar alguma dessas soluções seria bastante alto, por isso elas foram descartadas de início.

Já o *Crystal Space* apresentou uma dificuldade oposta, pois não existiam jogos RTS feitos utilizando tal engine. Dessa forma ter-se-ia que criar um jogo, e programá-lo como um todo, desde o início, o que seria inviável levando-se em consideração principalmente o tempo, disponível.

4.3.2 Stratagus

O *Stratatgus* parecia uma boa opção por ser uma engine para desenvolvimento de, RTS o que traria algumas vantagens, e por possuir diversos jogos já implementados baseados em sua plataforma. Destes destacou-se *Wargus* por ser o mais similar a jogos comerciais deste estilo, diferentemente do *Magnant*.

Mesmo com estes pontos positivos o *Stratagus* acabou sendo descartado por alguns fatores. O primeiro deles foi a necessidade de entender todo o código existente do *Wargus* para adaptá-lo, pois ele foi desenvolvido “apenas” como um jogo, sem foco em estudo de Inteligência Artificial (como o *Glest* e o *Boson*, citados anteriormente), o que acarretaria um enorme trabalho de reestruturação e adaptação para transformá-lo num simulador.

4.3.3 Robocode

Por ter sido desenvolvido com foco na IA e no comportamento dos agentes, é uma excelente opção para estudo e implementação de táticas de combate multiagentes. Facilita bastante o uso por ter sido desenvolvido utilizando Java como linguagem de programação, o que permite uma grande portabilidade. Além disso, há diversas IDE (*Integrated Development Environment*, ferramentas de auxilio ao desenvolvimento de softwares) que auxiliam o desenvolvimento de programas nessa linguagem.

Por ser um projeto criado e sustentado pela IBM, possui extensa documentação, e é constantemente atualizado para correção de bugs. Há diversos fóruns de discussão, e ainda, sua competição é um sucesso, com diversos participantes em todo o mundo. Desta forma é fácil encontrar inúmeros trabalhos e materiais sobre o *Robocode*, bem como também sua utilização como base dos estudos.

O ponto que o fez ser descartado foi o fato de não ser baseado em jogos RTS. Nele são programados tanques de guerra para combates individuais, um contra um, ou em time de no máximo cinco agentes. Com isso os combates são de pequena escala, bastante diferentes dos ocorridos em jogos de estratégia, onde são centenas contra centenas.

4.3.4 ORTS

O ORTS apresentou-se como o mais forte candidato, e a priori foi escolhido como a ferramenta para o desenvolvimento da IA dos agentes. Também dispõe de uma competição onde a técnica desenvolvida poderia confrontar as que haviam disputado o campeonato, servindo, portanto como uma boa forma de avaliação do trabalho, pois seria possível comparar com os resultados dos competidores.

Ele atendia a maioria dos requisitos anteriormente propostos. Era facilmente configurável através de scripts, abordava o problema da visibilidade, permitia a comunicação entre os agentes, etc. Havia inclusive uma categoria na competição focada exclusivamente no combate, o Game2.

Entretanto, apesar de tantas vantagens, os problemas apresentados pelo ORTS foram decisivos para o seu abandono. É uma ferramenta bastante complexa de ser usada e compreendida. A simples ação de compilá-lo mostrou-se uma tarefa bastante árdua, necessitando simular um ambiente Linux dentro do Windows. Executá-lo também era uma atividade complexa, principalmente pela falta de documentação adequada.

Outro fator crítico foi a grande instabilidade do sistema. O ORTS é um projeto *opensource,* sediado no *SourceForge* [SF], e o acesso aos seus arquivos é feito através de CVS, uma ferramenta de controle de versões. Aparentemente esta ferramenta não era usada adequadamente, pois a partir de um determinado momento quando se tentava atualizar o sistema, acabava-se por baixar códigos quebrados, que não permitiam o funcionamento correto do programa. Por exemplo, ao final da competição as soluções dos competidores foram disponibilizadas, mas não era possível compilá-las e executá-las, pois a versão do sistema havia sido alterada, e não era mais compatível com elas.

Diversos problemas como os citados acima acabaram tirando parte do brilho do ORTS. Ele parecia ser a melhor solução para o problema de simuladores de jogos RTS, inclusive pela boa abordagem do problema do combate, mas a enorme quantidade de problemas encontrados, completamente fora do âmbito da Inteligência Artificial acabou impossibilitando sua utilização.

4.4 CONSIDERAÇÕES FINAIS

Foram propostos alguns requisitos (cada com sua importância devidamente explicitada), que acredita-se, serem de suma importância para uma boa abordagem de simuladores de combate multiagente.

Analisou-se também as soluções existentes, levando em conta até algumas com foco diferente, mas que poderiam ser adaptadas. E, baseado, nos resultados obtidos decidiu-se, então, criar uma nova ferramenta para tratar o problema de falta de simuladores para combate multiagente. Solução esta que será apresentada em detalhes no capítulo seguinte.

CAPÍTULO 5

JARTS

“More work?”
— Peasant, Warcraft II

Neste capítulo propomos uma nova solução, baseada em outras já existentes, para simuladores RTS. O propósito é contribuir para o desenvolvimento do melhor ambiente para estudo de Inteligência Artificial, baseado em jogos RTS, o JARTS, acrônimo de *Java Real-Time Strategy*. Este é focado somente no estudo de Inteligência Artificial, e para tal escolheu-se como foco os jogos de estratégia em tempo real, por ser um dos gêneros mais complexos nesta área.

Como visto no capítulo anterior os simuladores são de extrema importância para estudos de diversas áreas, inclusive jogos e IA. Entretanto alguns nichos ainda são muito pouco explorados. De forma geral, não existem muitos simuladores para estudos de jogos, ao menos não com fácil acesso. Supõe-se que este tipo de normalmente seja usada de forma fechada pelas próprias empresas que desenvolvem os jogos, pois isso pode ser considerado um diferencial competitivo.

Os jogos RTS são extremamente completos para diversas linhas de estudo. É possível ir, de problemas simples, como evitar colisões, a problemas complexos como, evitar colisões, à tomadas de decisões extremamente complexas, em virtude da quantidade de variáveis e dinâmica do ambiente. Por este motivo decidiu-se então estudar tais jogos e seus respectivos simuladores. A partir daí surgiu a idéia de implementar um novo simulador dês combates em jogos RTS.

Para tal, foram estudadas diversas ferramentas com o mesmo propósito analisando seus pontos positivos e negativos, buscando extrair o melhor de cada uma delas, e procurar contornar os defeitos encontrados.

5.1 REQUISITOS

Baseado em uma análise comparativa, duas soluções que se destacaram, o *Robocode* e o *ORTS*. A seguir serão apresentados os pontos que inspiraram parte do desenvolvimento do novo simulador.

5.1.1 Simplicidade

O *Robocode* tem grande destaque neste aspecto, devido a dois pontos principais: o mecanismo usado para os agentes e colocados no simulador, e a forma como eles são implementados.

Para se carregar um agente nele, basta selecionar o arquivo que define seus comportamentos e depois adicioná-lo ao simulador. Este sistema de “importação” de arquivos foi um dos pontos que se adaptou para o JARTS, por ser uma alternativa simples e extremamente útil, pois simplificava bastante a utilização do simulador.

A forma como os agentes são implementados no *Robocode* é muito simples, nele existe uma classe *Robot*, que é a representação abstrata de um robô, e através do seu método *run*, são descritas todas as ações que agente deverá tomar durante a batalha. Esta abordagem é muito interessante, pois isola a implementação do simulador, do comportamento dos agentes, tornando muito mais fácil o aprendizado desta ferramenta, com pouco ou nenhum *overhead*. Na verdade, para o usuário não importa como serão implementadas as ações, e sim que sejam feitas de forma correta, por isto o simulador deve ser transparente ao usuário, para que ele foque exclusivamente na sua solução.

Ainda levando-se em conta a questão da simplicidade decidiu-se implementar o JARTS totalmente em Java, pois acredita-se que é uma linguagem de programação mais “amigável”, mais simples de ser entendida, e com um excelente suporte (documentação), como APIs e IDEs.

5.1.2 Possibilidade de Configuração

O *ORTS* é uma excelente solução neste aspecto, por isso o JARTS teve seu mecanismo de configuração, das unidades e do mundo, baseado nele. Sua implementação foi feita utilizando as *properties* de Java, que simplificam muito o processo de leitura dessas informações.

5.1.3 Visibilidade

Tanto o *ORTS* quanto *Robocode* abordam o problema da visibilidade, entretanto de formas distintas. O primeiro trata este problema de forma diferente, levando em conta o local

para o agente está olhando. No entanto, o segundo aproxima-se mais dos RTS comerciais, considerando uma campo de visão, circular, em volta de cada unidade. Este deverá ser a abordagem adota no JARTS.

5.1.4 Outros aspectos

A questão da comunicação proposta, no capítulo 4, é tratada de forma aceitável pelas duas soluções (*ORTS* e *Robocode*), pois em ambos é possível implementar técnicas de comunicação multiagente. No JARTS este tipo de abordagem também adotado

5.1.5 Competição

Seguindo o exemplo tanto do *Robocode* como do *ORTS*, decidiu-se fazer uma competição utilizando o JARTS para incentivar o estudo de Inteligência Artificial, de uma forma divertida, através dos jogos eletrônicos.

Esta competição já está sendo disputada por sessenta e cinco alunos da disciplina de Agentes Autônomos, do Centro de Informática da UFPE. Eles foram divididos em equipes, e cada uma delas irá implementar uma solução para um dos jogos existentes, que no dia da competição final, irão se confrontar.

Na disciplina serão avaliadas as soluções apresentadas por cada uma das equipes, e a solução vencedora receberá uma premiação especial. No dia 28 de setembro de 2006 houve a primeira prévia desta competição.

Além de uma forma de avaliação diferenciada visando despertar o interesse do aluno através de atividades lúdicas, a competição será de extrema importância para a validação inicial da ferramenta, o JARTS.

5.2 JARTS

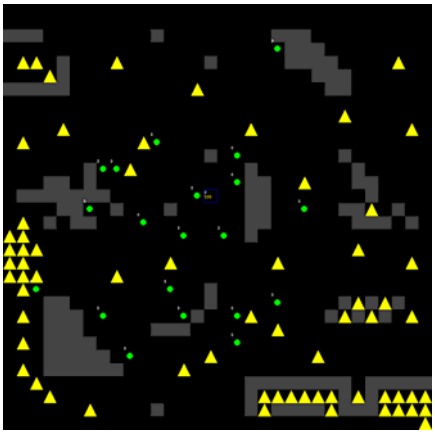
O JARTS é um simulador de jogos RTS, que visa incentivar o estudo da Inteligência Artificial de forma lúdica. O mercado de jogos é dos que mais cresce atualmente, como conseqüência disto, o número de jogadores também vem aumentando cada vez mais. Estudar uma disciplina através de alguns pode ser uma experiência muito interessante e proveitosa, que desperta a curiosidade do aluno.

No JARTS existem três tipos de unidade *Worker*, *Tank* e *ControlCenter*, que representam respectivamente um trabalhador, um tanque e a base de civilização. Há também três elementos de terreno, que compõem o mapa juntamente com as unidades, que são *PlainTerrain*, *Obstacle*, e *Resource*. O primeiro representa um terreno livre (onde as unidades podem ir), o segundo é um obstáculo (deve ser desviado), e o terceiro é uma fonte de recursos. Cada um destes elementos, unidades e terrenos, possuem uma representação diferente na interface da simulação, para facilitar o seu entendimento, e possibilitar o acompanhamento visual das ações dos agentes (a cor das unidades pode variar a cada partida).

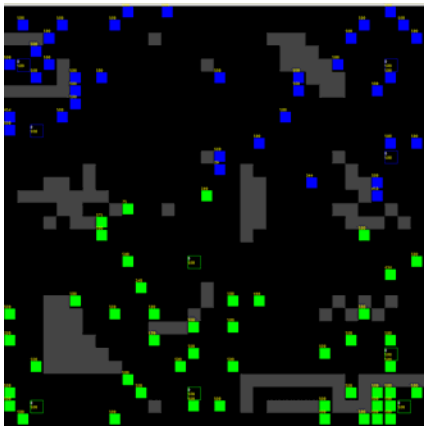
UNIDADES	TERRENOS
Worker	Obstacle
Tank	Resource
ControlCenter	PlainTerrain

Tabela 5.1 – Representação gráfica de cada tipo de *Element*

Os mapas são baseados em *tiles*, ou seja, toda a área do mapa é divida em pequenos quadrados (os *tiles*), formando uma grande malha quadriculada. Cada um dos quadrados só pode ser ocupado por uma única unidade, ou, caso estejam vagos, irá ser representado por algum tipo de terreno.



Game1



Game2

Ilustração 5.1 - Tipos de mapa, Game1 (com recursos) e Game2 (apenas tanques)

Semelhante ao ORTS, o JARTS possui dois tipos de jogos, que seguem as mesmas regras. No Game1, onde foco é a coleta de recursos, como descrito no Capítulo 4, só

existem trabalhadores, pois os tanques não são necessários para este problema. Já no Game2, cujo foco é o combate, existem apenas tanques.

5.2.1 Implementação

O JARTS foi todo desenvolvido em Java 15. A estrutura do sistema foi baseada no *Robocode*, assim como a forma de utilização. Optou-se por fazer um simulador que rodasse localmente, sem a necessidade de um cliente e um servidor, por ser uma abordagem mais simples.

Procurou-se também isolar a implementação do simulador, como no *Robocode*, da dos agentes. Estes podem ser feitos independente dele, contando apenas com sua biblioteca, e no momento de colocá-los no jogo basta escolher o arquivo (.class), que define seu comportamento, e carregá-lo no programa. A figura 5.2, abaixo, mostraa interface gráfica do JARTS, a tela onde são carregados todos os arquivos de implementação dos agentes.

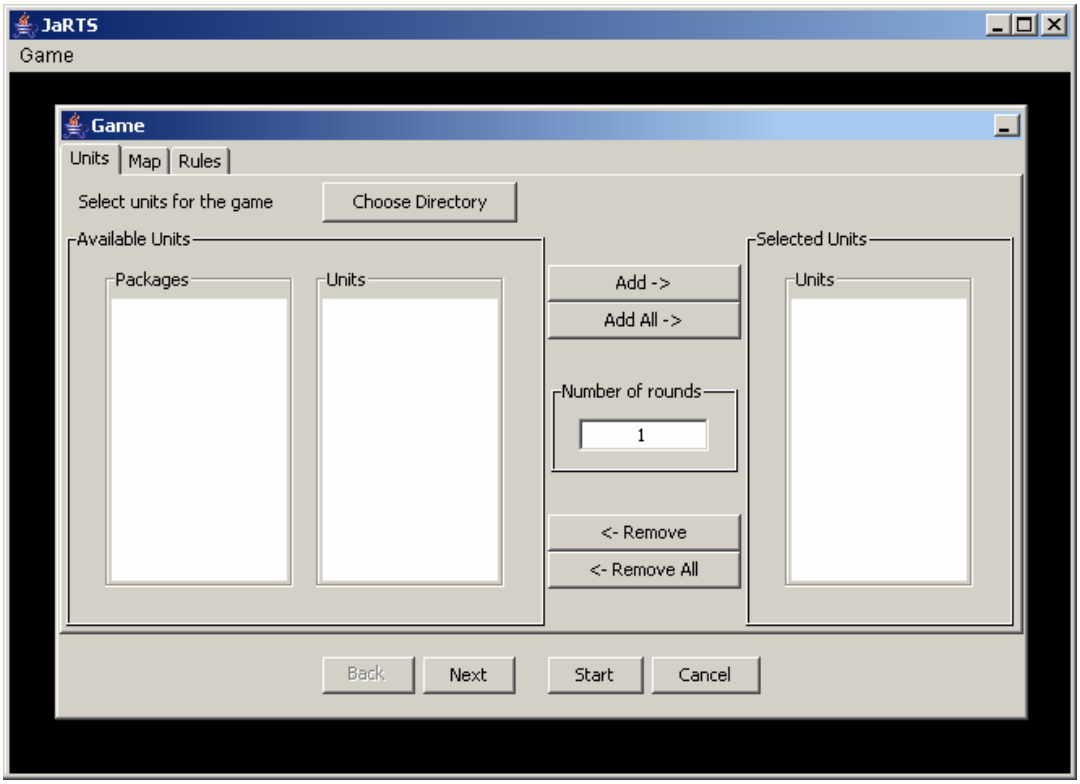


Ilustração 5.2 - Tela Inicial do JARTS, semelhante ao *Robocode*

Outro ponto importante tratado do desenvolvimento do JARTS foi a simplicidade. Era fundamental que a ferramenta fosse fácil de usar, sem a necessidade de conhecimento prévio, ou uma grande quantidade de tempo para estudá-la.

No JARTS, é possível definir os comportamentos de duas unidades, são elas: os *Workers* e os *Tanks*. Para isto basta estender uma destas classes, e implementar o método *updateAction*. Dessa forma, a cada ciclo do jogo ele é chamado e a ação definida nele é realizada.

Existem alguns métodos predefinidos no simulador, que devem ser chamados para que as requisições (as ações) possam sem realizadas, o simulador só “entende” algumas determinadas ações. Cada uma destas ações está associada a uma instrução, que é o que o simulador irá considerar na hora de executá-las. Para execução de tarefas mais complexas, estas devem ser compostas de várias ações diferentes. São elas:

- **idle** – é o estado inicial de todas as unidades, nada é feito.

- **move** – é chamado, passando-se o destino, quando se deseja mover uma unidade. A unidade só irá executar esta ação se duas pré-condicoes forem atendidas, o destino deve estar livre (ser um *PlainTerrain*) e o destino deve estar a 1 *tile* de distância da origem.

- **shoot** – ação que só pode ser executada por tanques, chamando-a irá atirar no oponente. As condições para esta ação são: o destino (alvo) deve ser alguma unidade (não é possível atirar em *tiles* vazios), e deve estar a pelo menos 5 *tiles* de distância do tanque que está atirando.

- **mine** – realizada somente por *Workers*, é a ação de minerar para coletar recursos. Para executá-la deve-se estar a 1 *tile* do destino, e este deve ser do tipo *Resource*.

- **deliver** – ação que complementa o *mine*, é o ato de entregar os recursos coletados de volta na base. Precisa-se ter recursos coletados, estar a 1 *tile*, do destino deve, obrigatoriamente, ser um *ControlCenter*.

A realização destas ações está associada à definição de uma instrução. Para mandar que o agente ande pelo mapa deve-se chamar seu método *move*, este faz apenas o seguinte: cadastrar naquela unidade que a instrução a ser executada é mover-se. Daí, a cada ciclo, quando o mundo é atualizado, o simulador varre todos os *tiles* perguntando

unidade a unidade, qual instrução deve ser executada. Caso as condições sejam satisfeitas, executa-se a ação, caso contrário a unidade nada é feito.

5.2.2 Diagrama de Classe

A seguir será descrita em mais detalhes, a estrutura do JARTS (mostrado na Ilustração 5.3), analisando-se as principais classes do sistema. Para organizar melhor sua implementação, o JARTS, foi dividido em pacotes (*Timer, World, Control, Util, Element, Unit* e *Terrain*), de acordo com as funcionalidades de cada classe.

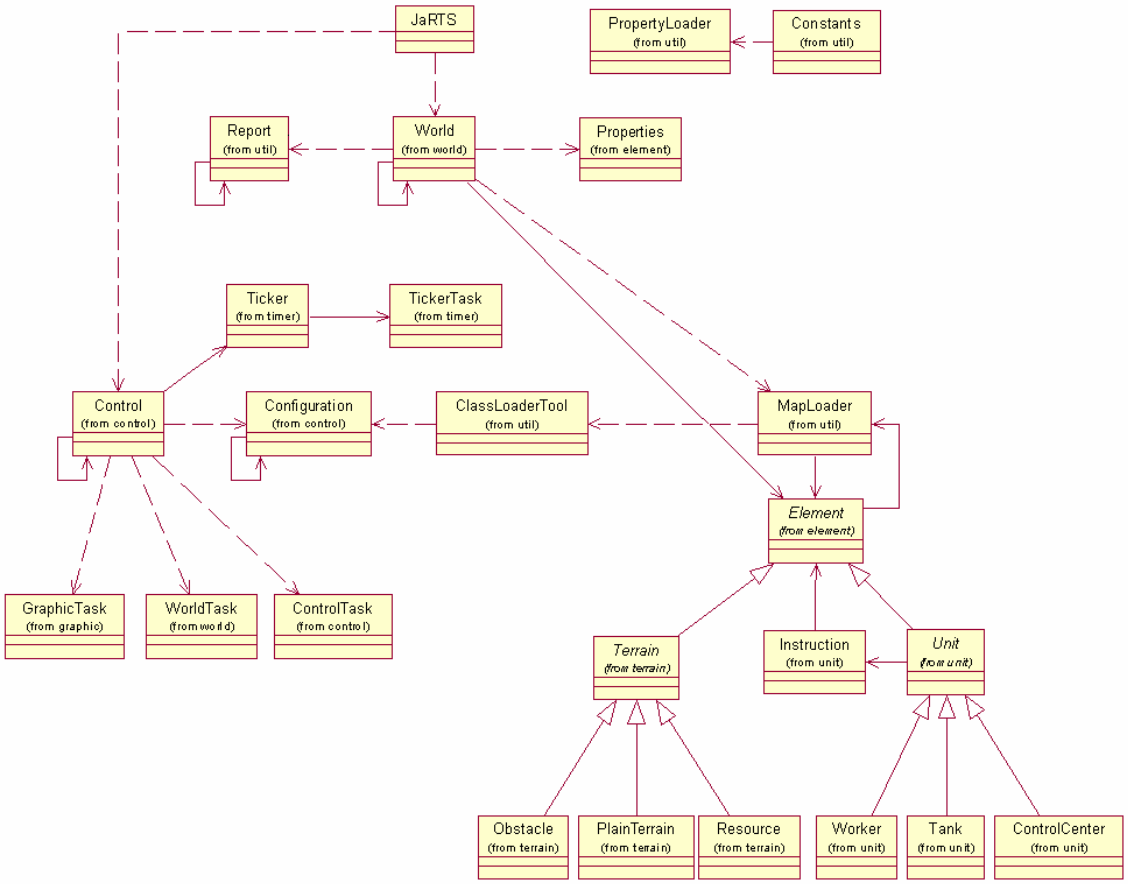


Ilustração 5.3 - JARTS classes principais

TIMER

Pacote que encapsula o controle de ciclos do jogo, a cada determinado período de tempo quando um evento que é disparado o que caracterizará um novo ciclo. Contém as seguintes classes:

- **Ticker** – é o encarregado de disparar os ciclos da simulação. É definido um tempo de duração de cada ciclo, e a cada disparo são executadas as tarefas cadastradas nele, que são representadas por objetos do tipo *TickerTask*

- **TickerTask** – é uma representação, abstrata, de uma tarefa a ser executada pelo *Ticker*. Ela deve ser estendida para a definição de outras tarefas, como a *GraphicsTask que* é a responsável por repintar os gráficos da simulação a cada novo ciclo.

WORLD

Este pacote encapsula as abstrações relacionadas ao ambiente do jogo. É composto pelas seguintes classes:

- **World** – feita usando o padrão de projeto Singleton para garantir que existirá uma única instância do mundo, e que todos podem ter acesso a ela. Um mundo é formado por um array de *Element* e seu tamanho irá definir o tamanho do mapa. A priori, só estão sendo utilizados mundos quadrados, de 32 x 32 ou 64 x 64 *tiles*.
- **WorldTask** – é a responsável pela chamada da atualização do mundo. A cada turno que chama os métodos *updateAction* de cada um dos agentes, para assim executar a ação de cada um deles para aquele turno

CONTROL

É responsável por todo o controle e configuração da simulação. É constituído das seguintes classes:

- **Control** – é a classe responsável por controlar a simulação como um todo, como, por exemplo, pausar uma simulação e recomeçá-la.
- **ControlTask** – é uma das tarefas atribuídas ao *Ticker*, responsável por verificar se já foi decorrido todo o tempo da partida ou do *round.*
- **Configuration** – possui as configurações de cada partida, como o número de rounds, a duração de cada um deles, qual arquivo será carregado, entre outras coisas.

UTIL

Contém as classes auxiliares do sistema, ou seja, classes que não tem relação direta com o funcionamento da ferramenta. São elas:

- **Report** – é responsável por gerar um *log* dos eventos ocorridos durante uma partida. Por exemplo, sempre que um tanque mata o inimigo esta classe guarda esta informação e decrementa a quantidade de unidades do time.
- **Constants** – contém as constantes usadas pelo simulador, é usada apenas para evitar que este tipo de informação fique espalhada pelas outras classes.

- **PropertyLoader** – carrega as propriedades da partida, como os atributos da unidades, duração da partida, do ciclo, etc.
- **MapLoader** – responsável pelo carregamento dos mapas da partida. No momento apenas escolhe entre alguns pré-definidos, mas deve, no futuro, ser capaz de gerar mapas aleatoriamente.
- **ClassLoaderTool** – responsável por processo de carregar os arquivos externos, que definem o comportamento de cada agente.

ELEMENT, UNIT E TERRAIN

O pacote *Element* contém todas as classes relacionadas aos elementos que compõe o mundo. E elas ainda são dividas em outros dois subpacotes: *Unit* (relacionado a todas as unidades do jogo, ou seja, elementos que podem ter seu comportamento definido) e *Terrain* (relacionado aos elementos naturais do mundo). Esses três pacotes são compostos pelas seguintes classes:

- **Element** – é uma representação abstrata do que pode ocupar um *tile*. Não possui implementação própria, é apenas uma classe mãe a qual é estendida por outras duas: *Terrain* e *Unit.*
- **Properties** – foi implementada devido a uma necessidade do sistema, pois de acordo com a modelagem inicial cada unidade poderia ter acesso e alterar tanto seus atributos, quanto o das unidades oponentes. Por exemplo, era possível acessar o tanque inimigo e alterar seus pontos de vida para zero, o destruindo. Devido a isso esta classe foi criada.

  Ela contém uma *Hashtable* (estrutura de dados que permite associação chave-valor de forma bastante eficiente), que faz um mapeamento de cada unidade e seus atributos. Nela são controladas as posições das unidades, o time ao qual pertencem, os pontos de vida que possuem, e a quantidade de recursos que estão carregando (usado no Game1). Há ainda a informação da cor da unidade usada para a diferenciação na interface gráfica do JARTS.
- **Unit** – é a representação abstrata de todas as unidades do simulador, e também uma classe abstrata sendo sua implementação feita através de suas classes filhas.
- **Worker** – é uma extensão de *Unit* e representa um trabalhador, utilizado no Game1.
- **Tank** – representação do tanque de combate utilizado no Game2.
- **ControlCenter** – representação da base de um time, utilizada no Game1 como deposito dos recursos coletados, e no Game2, como alvo principal da batalha.

- **Terrain** – é a representação de um “terreno”, todos os *tiles* que não estão ocupados com alguma unidade são obrigatoriamente algum tipo de terreno.
- **Obstacle** – é um obstáculo no mapa, um lugar por onde o agente não pode passar e terá que desviá-lo.
- **PlainTerrain** – representa um local por onde as unidades podem transitar. Caso o destino da unidade não seja um *tile* deste tipo ela não conseguirá mover-se.
- **Resource** – representa uma mina de ouro, por exemplo, algum lugar onde o trabalhador pode ir coletar recursos. Ela possui uma quantidade determinada de recursos, e ao se esgotar ela é automaticamente transformada em um *PlainTerrain*, podendo, então, ser ocupada.
- **Instruction** – está associada à classe *Unit*, e é ela que informa ao simulador qual ação será executada por aquela unidade. Ela possui um código que informa qual instrução vai ser executada, e um *Element*, o alvo ou destino, que é quem receberá a ação.

5.2.4 Exemplo

Para testar o JARTS, antes da realização da competição, foram implementadas algumas soluções, uma em cada tipo de jogo. As duas apresentam comportamentos semelhantes, se movem aleatoriamente até encontrarem seu objetivo (um inimigo ou recurso). Ao encontrarem executam a ação correspondente.

No Game1 foram criadas duas soluções de teste, a primeira delas um *DummyWorker* que apenas se move pelo mapa sem executar nenhuma outra ação. A segunda foi o *BlackBoardWorker*, que é um trabalhador que utiliza um *BlackBoard* (ver a técnica *Messaging* no Anexo B) para comunicar-se com os outros agentes. Esta solução foi criada para garantir que era possível atender a um dos requisitos estabelecidos, a comunicação multiagentes.

RandomTank é um tanque, do Game2, que anda pelo mapa de forma aleatória. Enquanto se move ele fica varrendo uma área em torno dele, com raio 3 *tiles*, buscando algum inimigo em seu raio de ataque. Quando um oponente é encontrado ele pára de se mover e começa a atirar, até que o inimigo morra, em seguida volta a se mover aleatoriamente.

5.3 CONSIDERAÇÕES FINAIS

Neste capítulo foram avaliados e descritos os pontos, de outras soluções existentes, que seriam adaptados ao JARTS, explicando o motivo de suas escolhas. Em seguida a nova solução foi apresentada em detalhes, mostrando do seu funcionamento ao diagrama de classe (simplificado) do sistema.

Por fim são mostrados três exemplos de agentes, explicando seus comportamentos e parte de sua implementação, que foram implementados para a validação do JARTS. No próximo capítulo serão avaliadas as contribuições trazidas pela ferramenta, bem como possíveis extensões.

CAPÍTULO 6

CONCLUSÃO E TRABALHOS FUTUROS

"… has an end"
— Matrix

Este capítulo discute as contribuições deste trabalho, apresenta as dificuldades encontradas e sugere possíveis melhorias futuras para tornar este trabalho melhor e mais completo.

6.1 CONTRIBUIÇÕES

A principal contribuição deste trabalho foi a criação de uma nova ferramenta para dar suporte ao estudo da Inteligência Artificial. Como sua implementação foi baseada em uma análise mais detalhado de diversas outras soluções existentes, acredita-se que esta nova, o JARTS, irá ser melhor que as estudadas nos aspectos propostos.

A aposta na simplicidade foi fundamental. Já existiam alternativas semelhantes, como *Stratagus* e *ORTS*, mas todas bastante complexas. Talvez por isso elas não tenham tido tanto sucesso quanto o *Robocode*, uma solução parecida, mas com um foco diferente.

O JARTS está sendo utilizado na disciplina de Agentes Autônomos, do Centro de Informática da UFPE, como forma de competição entre os grupos de alunos. Já foi feita uma disputa prévia e o "feedback" foi fundamental para apontar *bugs* ainda existentes, e possíveis melhorias.

6.2 DIFICULADES ENCONTRADAS

Ao longo do desenvolvimento deste trabalho diversas dificuldades foram encontradas. A princípio o objetivo do trabalho não era criar um simulador, mas desenvolver uma estratégia de combate multiagentes para jogos RTS. Esta solução seria implementada utilizando o ORTS, e os resultados obtidos seriam comparados com os submetidos para a sua competição. Para isto foram estudadas diversas técnicas de IA aplicadas ao combate (Ver Anexo B), e também foram analisados diversos jogos para estudar seu comportamento.

A quantidade de problemas encontrados com o ORTS foi enorme, tendo sido gastos cerca de dois meses tentou-se utilizá-lo sem sucesso. Os problemas iam desde a compilação do código disponibilizado, à execução de um simples exemplo, passando por problemas de implementação da solução.

Por isso surgiu a idéia de criar uma nova solução, visto que a equivalente, , apresentava muitas dificuldades. Neste contexto surgiu o JARTS, inspirado no ORTS, e que visa melhorá-lo, corrigindo os problemas encontrados. Por isto optou-se por fazer uma versão mais simples e de fácil utilização, seguindo o modelo adotado pelo *Robocode*, o que levou a um resultado satisfatório, atingindo os requisitos de simplicidade inicialmente proposto.

6.3 TRABALHOS FUTUROS

Devido a grande quantidade de adversidades encontradas ao longo do trabalho, o JARTS ainda está encontra concluído. O primeiro passo seria a implementação de uma *Thread* para cada unidade, de modo que a simulação não fosse retardada por operações demoradas. Esta seria a mudança mais crítica a ser feita.

Outra possível melhoria para o problema do combate seria a adição do conceito de visibilidade, proposto nos requisitos do simulador. No momento atual todos têm acesso a todas as informações disponíveis no mundo, o que deve ser restringido a um campo de visão em torno de cada unidade.

Ainda de acordo com os requisitos propostos, pode-se atuar na adição de novos tipos de unidades, o que acarretaria um grande aumento nas possibilidades de estratégia e táticas a serem usadas, como as formações dos pelotões.

Também é importante trabalhar a divulgação deste projeto, associado à competição, para fazê-la crescer. Pode-se pensar na criação de um campeonato nacional ou internacional, rivalizando com o próprio ORTS, de jogos RTS controlados pelo computador.

6.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou as principais contribuições e dificuldades encontradas no decorrer do trabalho. O trabalho mostrou a viabilidade do desenvolvimento de uma ferramenta, simples e ao mesmo tempo útil tanto para o estudo de IA quanto para o desenvolvimento de jogos eletrônicos.

REFERÊNCIAS

[Schwab 2004] Schwab, Brian. *AI Game Engine Programming*. Charles River Media, Setembro, 2004.

[Rabin 2002] Rabin, Steve. *Ai Game Programming Wisdom*. Vol 1. Charles River Media, 2002.

[GameAI] Game artificial intelligence
http://en.wikipedia.org/wiki/Game_ai

[Vieira 2005] V. Vieira. *Revolution Engine – Desenvolvimento de um motor de Inteligência Artificial para a Criação de Jogos Eletrônicos*. Trabalho de Graduação, Graduação em Ciência da Computação, Universidade Federal de Pernambuco. Ano: 2005.

[Russell e Norvig 2003] S. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach*.
Prentice Hall, 2003

[Bourg e Seemann 2004] D. Bourg, G. Seemann. *AI for Game Developers*. O'Reilly, 2004

[Ferreira 1999] A. FERREIRA. *Novo Dicionário Aurélio – Século XXI*. Nova Fronteira, 1999

[Pong] PONG-Story
http://www.pong-story.com/1958.htm
Visitado em: 2/10/2006

[RTS] Real-time Strategy
http://en.wikipedia.org/wiki/Real-time_strategy
Visitado em: 2/10/2006

[AoE series] Age of Empires series
http://en.wikipedia.org/wiki/Age_of_Empires_series
Visitado em: 2/10/2006

[AoE] Age of Empires
http://en.wikipedia.org/wiki/Age_of_Empires
Visitado em: 2/10/2006

[AoE Rome]	Age of Empires: The Rise of Rome http://en.wikipedia.org/wiki/Age_of_Empires:_The_Rise_of_Rome_Expansion Visitado em: 2/10/2006
[AoK]	Age of Empires II: The Age of Kings http://en.wikipedia.org/wiki/Age_of_Empires_II:_The_Age_of_Kings Visitado em: 2/10/2006
[AoK Conquerors]	Age of Empires II: The Conquerors http://en.wikipedia.org/wiki/Age_of_Empires_II:_The_Conquerors_Expansion Visitado em: 2/10/2006
[AoE3]	Age of Empires III http://en.wikipedia.org/wiki/Age_of_Empires_III Visitado em: 2/10/2006
[C&C]	Command & Conquer: Red Alert http://en.wikipedia.org/wiki/Command_%26_Conquer:_Red_Alert Visitado em: 2/10/2006
[EE]	Empire Earth http://en.wikipedia.org/wiki/Empire_Earth Visitado em: 2/10/2006
[StarCraft]	StarCraft http://en.wikipedia.org/wiki/StarCraft Visitado em: 2/10/2006
[StarCraft GP]	Gameplay of StarCraft http://en.wikipedia.org/wiki/Gameplay_of_StarCraft Visitado em: 2/10/2006
[Warcraft]	Warcraft: Orcs & Humans http://en.wikipedia.org/wiki/Warcraft:_Orcs_%26_Humans Visitado em: 2/10/2006
[Warcraft2]	Warcraft II http://en.wikipedia.org/wiki/Warcraft_II:_Tides_of_Darkness Visitado em: 2/10/2006
[War3]	Warcraft III: Reign of Chaos http://en.wikipedia.org/wiki/Warcraft_III:_Reign_of_Chaos

Visitado em: 2/10/2006

[War3 FT] Warcraft III: The Frozen Throne

http://en.wikipedia.org/wiki/Warcraft_III:_The_Frozen_Throne

Visitado em: 2/10/2006

[B&W] Black & White

http://www.lionhead.com/bw/about.html

Visitado em: 2/10/2006

[Buro 2003] M. Buro. *Real-Time Strategy Games A New AI Research Challenge*. Proceedings of the International Joint Conference on AI 2003, Acapulco, Mexico.

[Civilization] Civilization

http://en.wikipedia.org/wiki/Civilization_%28game%29

Visitado em: 2/10/2006

[BFME II] The Lord of the Rings: The Battle for Middle-earth II

http://en.wikipedia.org/wiki/Battle_for_Middle-earth_2

Visitado em: 2/10/2006

[Laursen e Nielsen 2005] R. Laursen, D. Nielsen. *Investigating small scale combat situations in real time strategy computer games*. Masters Thesis, University of Aarhus, Department of computer science. Ano: 2005

[Demyen 2004] D. Demyen. *Selecting Targets for Abstract Battles in Real Time*. Department of Computing Science, University of Alberta, 2004.

[Sins] Sins of a Solar Empire

http://en.wikipedia.org/wiki/Sins_of_a_Solar_Empire

Visitado em: 2/10/2006

[Halo 2] Halo 2

http://en.wikipedia.org/wiki/Halo_2

Visitado em: 2/10/2006

[Dark Reign] Dark Reign

http://en.wikipedia.org/wiki/Dark_Reign

[Gamasutra] Gamasutra - Features - "Game AI: The State of the Industry”

http://www.gamasutra.com/features/20001108/laird_02.htm

Visitado em: 2/10/2006

[Simulation] Simulation

http://en.wikipedia.org/wiki/Simulator#Computer_simulation

Visitado em: 2/10/2006

[Glest] Glest, A free 3d RTS (real time strategy) project http://www.glest.org/en/index.html

Visitado em: 2/10/2006

[Boson] Boson

http://boson.eu.org/

Visitado em: 2/10/2006

[Crystal] Crystal Space 3D

http://www.crystalspace3d.org/tikiwiki/tiki-view_articles.php

Visitado em: 2/10/2006

[CEL] Crystal Entity Layer

http://www.crystalspace3d.org/tikiwiki/tiki-index.php?page=Cel

Visitado em: 2/10/2006

[Stratagus] Stratagus

http://stratagus.sourceforge.net/

Visitado em: 2/10/2006

[Ponsen 2005] M. Ponsen et al. *Stratagus An Open-Source Game Engine for Research in Real-Time Strategy Games*. Workshop for International Joint Conference on Artificial Intelligence 2005.

[Davis 1999] I. Davis, "Strategies for Strategy Game AI,"AAAI 1999 Spring Symposium on AI & Computer Games Proceedings. March, 1999.

[Wargus] Wargus

http://wargus.sourceforge.net/

Visitado em: 2/10/2006

[BoS] Invasion - Battle of Survival

http://bos.seul.org/

Visitado em: 2/10/2006

[Magnant] Magnant – Insect war

http://insectwar.free.fr/

Visitado em: 2/10/2006

[Eisenstein 2003] J. Eisenstein. *Evolving Robocode Tank. Fighters*. AI Memo 2003-023. October 2003

[Robocode] Robocode Central

http://robocode.sourceforge.net/
Visitado em: 2/10/2006

[ORTS] ORTS - A Free Software RTS Game Engine
http://www.cs.ualberta.ca/~mburo/orts/
Visitado em: 2/10/2006

[SF] SourceForge.net
http://sourceforge.net/index.php
Visitado em: 2/10/2006

APÊNDICE A

JOGOS RTS

A.1 Age of Empires

A série de *Age of Empires*, comumente abreviada para AoE, é uma das mais populares do gênero RTS, teve início em 1997, foi desenvolvido pela *Ensemble Studios* e publicado pela *Microsoft Game Studios* [AoE series]. Tem uma temática histórica, simulando a evolução das civilizações. Existem cinco títulos na série (dois são expansões). Para garantir a vitória, o jogador deve coletar recursos com o objetivo de pagar por novas unidades, construções e avanços tecnológicos. É possível configurar o jogo para a vitória ser baseada em conquistas econômicas e tecnológicas, mas na prática a maioria dos jogadores foca em criar exércitos para derrotar seus oponentes.

AGE OF EMPIRES I E AGE OF EMPIRES I: RISE OF ROME

Também conhecido como AoE1, neste jogo o jogador tem que desenvolver uma nação, a partir da Idade da Pedra, com suas unidades trajando peles, até uma civilização do final da Idade do Ferro, com suas armas e tecnologias. Existem, no jogo, quatro Idades para se evoluir, são elas: Idade da Pedra (Paleolítico, Era inicial dos jogadores), Idade da Ferramenta (Neolítico), Idade do Bronze e a Idade do Ferro. Há doze civilizações a serem escolhidas para jogar (Assírios, Babilônicos, Coreanos - *Choson*, Egípcios, Gregos, Hititas, Minoanos, Persas, Fenícios, Sumérios, Japoneses - *Yamato*), cada uma possui características próprias, com suas vantagens e desvantagens.

Ilustração A.1 - *Age of Empires I [AoE]* , cidade (Idade do Ferro) sofrendo um ataque

A sua expansão, *Rise of Rome*, acrescenta vários novos elementos ao jogo original. Há novas unidades, como a Galera, um barco de defesa contra outras unidades navais. Novas tecnologias como, por exemplo, Medicina, que permite que as unidades sejam curadas de forma mais rápida. E há também a adição de quatro novas civilizações, os Cartagos, os Macedônios, os Sírios (Palmyra), e por fim os Romanos, que dão título à expansão [AoE Rome].

AGE OF EMPIRES II: AGE OF KINGS E AGE OF EMPIRES II: THE CONQUERORS

Continuação do AeE1, é o segundo título da série Age of Empires lançado em 1999, se passa na Idade Média. Como em seu antecessor, o jogador controla uma sociedade e a guia por quatro Eras. O jogo se inicia na *Dark Age*, o início sombrio da Idade Média, evolui para a Idade Feudal. A próxima etapa é a Idade dos Castelos, e por fim chegando a Idade Imperial, que é equivalente aos primeiros anos do Renascença [AoK]. As civilizações disponíveis neste jogo são: Bretões, Bizantinos, Celtas, Chineses, Francos, Godos, Japoneses, Mongóis, Persas, Sarracenos, Teutões, Turcos e Vikings. O jogo segue as características históricas de cada uma das sociedades, por exemplos, os Vikings são excelentes em batalhas marítimas.

The Conquerors é a expansão do Age 2, nela são adicionadas um total de vinte e seis novas tecnologias [AoK Conquerors], melhoramentos gráficos, e melhoraria também na Inteligência Artificial. Nas versões anteriores os aldeões, quando controlados por humanos, não apresentavam nenhuma autonomia. Já nessa versão, por exemplo, ao ser concluir a construção de uma “madeireira”, eles automaticamente começam a colher recursos do ponto mais próximo. Nesta expansão também há novas civilizações, Astecas, Hunos, Coreanos, Maias e Espanhóis.

Ilustração A.2 - *Age of Empires II: The Conquerors*[AoK Conquerors], civilização Asteca e seu castelo

AGE OF EMPIRES III

Terceiro título e mais recente da seqüência *Age of Empires*, lançado em outubro de 2005, é ambientado no Novo Mundo, e cobre a época da colonização européia nas Américas entre os anos 1500 e 1850 aproximadamente [AoE3]. É da série o que apresenta a maior evolução gráfica, se comparado a versão anterior.

Ilustração A.3 - *Age of Empires III*, um forte e um navio da civilização Britânica [AoE3]

No Age 3, assim como nos outros jogos da série, o jogador deve desenvolver uma nação a partir de um simples estabelecimento, com alguns aldeões, até uma civilização poderosa, progredindo através de Eras ou tecnologias que dão mais poder para o seu império. Há duas linhas principais que regem o jogo, o exército e a economia da nação. A vitória dependerá do domínio destes fatores.

Uma das grandes novidades deste jogo é o sistema de "cidade natal", que representa a relação Colônia–Metrópole da época, onde elas dão apoio tanto militar como econômico.

Existem oito civilizações à escolha do jogador (Espanhóis, Britânicos, Franceses, Portugueses, Holandeses, Russos, Alemães e Otomanos), cada uma com suas peculiaridades, forças e fraquezas, além de unidades únicas. Há no jogo também outros povos (Astecas, Caribenhos, Cherokees, Comanches, Crees, Incas, Iroquois, Maias, Nootkas, Seminoles, Lakotas e Tupis) que representam os povos nativos dos locais, e eles podem ser "colonizados" pelas civilizações.

A.2 Age of Mythology e Age of Mythology: The Titans

Age of Mythology, conhecido como AoM, é um jogo RTS dos mesmos criadores da série *Age of Empires*, *Ensemble Studios* e *Microsoft Game Studios*. Foi considerado por muitos a continuação dos AoE, até o lançamento do "herdeiro" oficial, Age 3. Ao contrário dos seus antecessores tem menos preocupação com a precisão histórica, seu foco gira em torno de mitos e lendas da Escandinávia, Grécia e Egito antigo. O jogo introduz também o conceito de unidades "míticas" que são a representação de criaturas da mitologia. Por exemplo, só existiu uma Hidra na mitologia, mas no jogo várias podem ser "invocadas" (criadas). São unidades extremamente mortíferas contra unidades comuns, mas fracas diante dos Heróis. Este foi outro conceito acrescentado ao jogo, eles são representações de figuras épicas da mitologia. Possui uma extensão chamada *Age of Mythology: The Titans*, que segue a mesma linha da versão original, apenas acrescentando algumas unidades.

A.3 Command & Conquer

É também uma série de bastante sucesso no gênero de jogos RTS, contendo um total de seis versões. Diferentemente dos Ages, *Command & Coquer* tem uma ambientação mais moderna, mas não segue nenhuma ordem histórica/cronológica, nem possui ligação entre suas versões. Os jogos existentes dessa série são:

- **Tiberian Dawn** – acontece em 1995, numa disputa pelo *Tiberium*, uma nova substância que chega a Terra em um meteoro.

- **Red Alert** – sua história se passa após o final da Segunda Guerra Mundial, começando em 1946, numa realidade alternativa onde Albert Einstein inventa uma máquina do tempo. Aliados e Soviéticos batalham pelo controle da Europa [C&C].

- **Tiberian Sun** – este jogo ignora o antecessor, o *Red Alert*, e resgata a história do primeiro, seu enredo se desenrola trinta anos após o final do *Tiberian Dawn*.

- **Red Alert 2** – continuação do *Red Alert*, sem nenhuma menção a história dos *Tiberians*, resgata a cisão entre Aliados e Soviéticos. Apresenta também um conceito de "herói" (Tânia, dos aliados, e Yuri, dos soviéticos) como unidades superiores, entretanto únicas, não podendo ser recriadas.

- **Generals** – sua linha de história não possui relação alguma com qualquer dos outros jogos da seqüência. O jogo é ambientado em 2010, onde Estados Unidos e China

são superpotências mundiais e são alvos de uma organização terrorista com poderio nuclear.

- **Tiberium Wars** – é o jogo mais recente da linha *Command & Conquer*. Novamente resgata a história do primeiro e terceiro da série, seu desenrolar se inicia em 2047.

A.4 Empire Earth

Jogo baseado na historia do mundo, semelhante aos AoE, passando por 14 épocas (mais de 500.000 anos) começando na Era Pré-histórica e terminando na Era da Nanotecnologia [EE]. Contém várias inovações únicas, incluindo elementos como um sistema de "moral", que afeta diretamente unidades individuais. Também incorpora o conceito de "heróis", unidades especiais inspiradas em personalidades históricas com extraordinárias habilidades de combate, cura, e influência na moral dos exércitos.

As épocas são Eras que o jogador atravessa ao jogar *Empire Earth*, também conhecido como EE, cada uma representando um pedaço da história e trazendo as inovações correspondentes. São elas:

- **Pré-história** – primeira época do jogo tem início com a descoberta do fogo. Nesse momento o jogo é bastante limitado, com unidades armadas de pau e pedras.

- **Idade da Pedra** – segunda época tem-se a oportunidade de construir barcos, arqueiros e aríetes. É um pouco mais avançada que a pré-história.

- **Idade do Cobre** – começam as fazendas, hospitais, o que permite o crescimento populacional, universidades, muralhas e fortalezas. Heróis ficam disponíveis nesse momento.

- **Idade do Bronze** – ocorre a diversificação dos exércitos e navios, com a introdução de fragatas, galeras, e armas de cerco.

- **Idade das Trevas** – fracamente refletida na historia mundial. Mais armas "bárbaras" são disponibilizadas, como o *crossbowman*, uma cavalaria armada com bestas.

- **Idade Média** – torres e muralhas ficam poderosas, sendo dificilmente derrotadas, pois apenas armas de cerco e navios de guerras furam seus bloqueios.

- **Renascença** – início da pólvora e armas de cerco extremamente poderosas com a adição de trebuchets e balistas.

- **Idade Imperial** – grande salto na evolução do poder naval e invenção dos remédios.
- **Idade Industrial** – início de uma grande revolução com enormes avanços tecnológicos e militares.
- **Primeira Guerra Mundial** – ficam disponíveis tanques e aviões, e acontece a invenção da artilharia. Surgem os mercados, que permitem compra e venda de recursos.
- **Segunda Guerra Mundial** – grande evolução militar, como pára-quedistas e bomba nuclear.
- **Idade Moderna** – surgimento de submarinos nucleares, e helicópteros.
- **Idade Digital** – criação de robôs como máquinas de combate, introdução do LASER na infantaria, engenharia genética e satélites espiões.
- **Idade Nano** – incrível evolução dos robôs, como o Hades, que tem a habilidade de tele transporte, e disseminação de vírus nos robôs oponentes.

Ilustração A.4 - Empire Earth, civilização evoluída com lasers

São disponibilizadas vinte e uma civilizações à escolha do jogador. Entretanto neste jogo há uma pequena diferença, as sociedades são agrupadas por épocas, não sendo possível, por exemplo, levar os assírios à Idade Digital.

- **Pré-história a Idade das Trevas**: Grécia Antiga, Assírios, Babilônia, Romanos Bizantinos, Cartagos, Reino de Israel.

- **Idade Média a Idade Industrial**: Áustria, Inglaterra, Francos, Reino da Itália, Império Otomano, Espanha.

- **Idade Moderna**: França, Alemanha, Grã Bretanha, Itália, Rússia, Estados Unidos.

- **Idade Digital**: China, Nova Rússia, Rebeldes.

Em *Empire Earth II*, a continuação do EE, houve algumas mudanças, foram acrescentadas coisas como, sistema de diplomacia que permite ao jogador fazer alianças, o "War Planner" onde o jogador pode combinar as coordenadas dos ataques com os aliados, clima que influencia o desempenho das unidades, além de novas unidades, campanhas, e uma reformulação na divisão das épocas [EE].

A.5 WarCraft

WarCraft envolve toda uma série de jogos que teve início com RTS, e atualmente seu maior sucesso é o *World of WarCraft*, um RPG (Role Playing Game). Dentre os de estratégia três jogos foram lançados, e mais ainda algumas expansões.

- **WarCraft: Orcs & Humans [Warcraft]** – primeiro jogo da série, lançado em 1994 pela *Blizzard Entertainment*, desenvolvido originalmente para o DOS. Em sua história Orcs demoníacos descobrem um portal permitindo seu acesso a um reino humano, Azeroth. O jogador pode escolher entre estas duas raças, Orcs e Humanos.

- **WarCraft II: Tides of Darkness [Warcraft2]** – normalmente chamado apenas de *War2* foi uma conseqüência do grande sucesso e popularidade do antecessor. Novamente fez-se uso de facções opostas, ainda os Orcs e Humanos. Cada raça tinha sempre uma unidade equivalente no inimigo, uma forma de equilibrar o jogo não dando vantagens a nenhum dos lados. Este jogo teve duas expansões, o **Beyond the Dark Portal** e o **Battle.Net Edition**. Este último não é exatamente uma expansão, é mais uma versão revisada para dar suporte a *Battle.Net*, rede de servidor da *Blizzard* onde milhares de pessoas jogam umas contra as outras pela internet.

Ilustração A.5 - WarCraft 2 Battle.Net Edition

- **WarCraft III: Reign of Chaos[War3]** – versão mais recente dos RTS da linhagem *WarCraft*, também chamado de *War3*, lançado em 2002. Ainda mantém a luta entre Orcs e Humanos, mas são acrescentadas novas raças os *Night Elves* (Elfos) e os *Undeads* (Mortos-vivos). Apresenta gráficos muito melhores que os anteriores. Acrescentou o conceito de Herói que, neste jogo, são unidades que acompanham o jogador ao longo da sua campanha e vão evoluindo ao longo das partidas, ganhando experiência e adquirindo novos poderes. Em abril de 2003 foi lançada sua expansão, o **Frozen Throne[War3 FT]**, que trouxe novas unidades para as raças, assim como novos heróis, além de algumas mudanças na jogabilidade.

Ilustração A.6 - WarCraft 3 [War3]

A.6 StarCraft

É um jogo RTS da *Blizzard Entertainment*, lançado em 1998. Sua linha de história é composta por guerras entre três espécies galácticas: os Terrans, os Zergs e os Protoss. Sua jogabilidade é focada na aquisição e controle de dois recursos, minerais e gás Vespene, que são necessários para construir unidades e prédios.

Ilustração A.7 - StarCraft, Zerg vs Terran

StarCraft é uma evolução do seu predecessor, o *War2*, que possuía elementos avançados para seu tempo, embora muitos dos jogadores consideravam fraquezas [StarCraft]. Este problema era o fato de, que, descontando algumas pequenas diferenças em feitiços e custos, jogar com as duas raças de *WarCraft II* tinha exatamente a mesma mecânica apresentando somente mudanças gráficas, problema que foi corrigido no *War3*. *StarCraft* evoluiu neste sentido ao incorporar três raças ao invés de duas, e ao atribuir elementos e tecnologias únicas para cada uma delas. A escolha da civilização é crucial, pois cada uma irá influenciar definitivamente no estilo de jogo. Este jogo é bastante elogiado quanto ao balanceamento das raças e unidades: não importa que seja escolhida, o jogador terá chances iguais de vencer uma partidas, mesmo se for contra uma unidade muito diferente [StarCraft GP].

StarCraft tem uma expansão, o *StarCraft: Brood War*, que desde seu lançamento vendeu nove milhões de cópias [StarCraft]. Nela foram acrescentados novos elementos, duas novas unidades para cada raça, novos mapas e novas tecnologias. Além de novas campanhas, que como é chamada o modo single-player de um RTS.

APÊNDICE B

TÉCNICAS DE IA EM COMBATE

Como visto anteriormente, *Game AI*, Inteligência Artificial (IA) aplicada a Jogos, refere-se a técnicas usadas em jogos de computador/videogames para produzir a ilusão de inteligência no comportamento de NPCs (*Non-Player Characters*) [GameAI]. Ela foca em resolver diversos problemas relacionados ao comportamento e as decisões dos agentes (personagens). Cada um tem problemas inerentes ao estilo, num jogo de luta o oponente não precisa fazer *pathfinding*, como em vários outros, visto que o cenário é sempre conhecido. Com relação a estas peculiaridades, os jogos de estratégia são os mais complexos, pois envolve inúmeros problemas a serem tratados, talvez, com mais possibilidades de aplicação de técnicas de IA distintas. Tais jogos são dos mais complicados, por serem bastante abrangentes, focam no planejamento cuidadoso e na habilidade de gerenciar recursos, comumente lidando com inúmeras unidades, detecção de colisões e árvores tecnológicas [Schwab 2004].

B.1 Técnicas

Nos capítulos 2 e 3 foram analisados alguns dos principais problemas que são enfrentados ao se tentar implementar a IA de um RTS. Neste tópico serão apresentadas algumas das técnicas mais utilizadas, e como podem ser adaptadas para a resolução destes problemas.

B.1.1 FINITE-STATE MACHINES (FSMS)

Máquina de Estados Finita, FSM como são chamadas, são as estruturas de dados mais usadas no desenvolvimento de jogos. É extremamente útil quando se deseja quebrar grandes problemas em menores, tornando-os mais fáceis de gerenciar, e permitindo a implementação de sistemas inteligentes com mais flexibilidade. Uma máquina de estado é uma estrutura de dados que contem três coisas: todos os estados inerentes à máquina, várias condições de entrada, e a função de transição (funciona como as linhas de conexão entre os estados) [Schwab 2004], ver figura a seguir. Ela é uma máquina abstrata que pode existir em, apenas, um de vários estados, e o estado atual define como ela deve se comportar [Bourg e Seemann 2004].

Máquina de estados datam dos primórdios da programação de jogos de computador. Por exemplo, os fantasmas de *Pac Man* são máquinas de estados finitas. Em cada estado eles agem de forma diferente, e suas transições são determinadas pelas ações do jogador [Bourg e Seemann 2004]. Veja abaixo a máquina de estado do Blinky, um dos fantasmas de Pac Man. Na figura os retângulos representam os estados únicos, ou seja, não é possível que o agente esteja em dois no mesmo instante, por exemplo: “Chase Player”, e “Move Randomly”. As setas representam as funções de transição, a condição que deve ser satisfeita para a mudança de estado. Se o agente está no “Chase Player” e o jogador morre, é disparado um evento que o faz mudar de estado para “Move Randomly”. Esta FSM será mais detalhada a seguir.

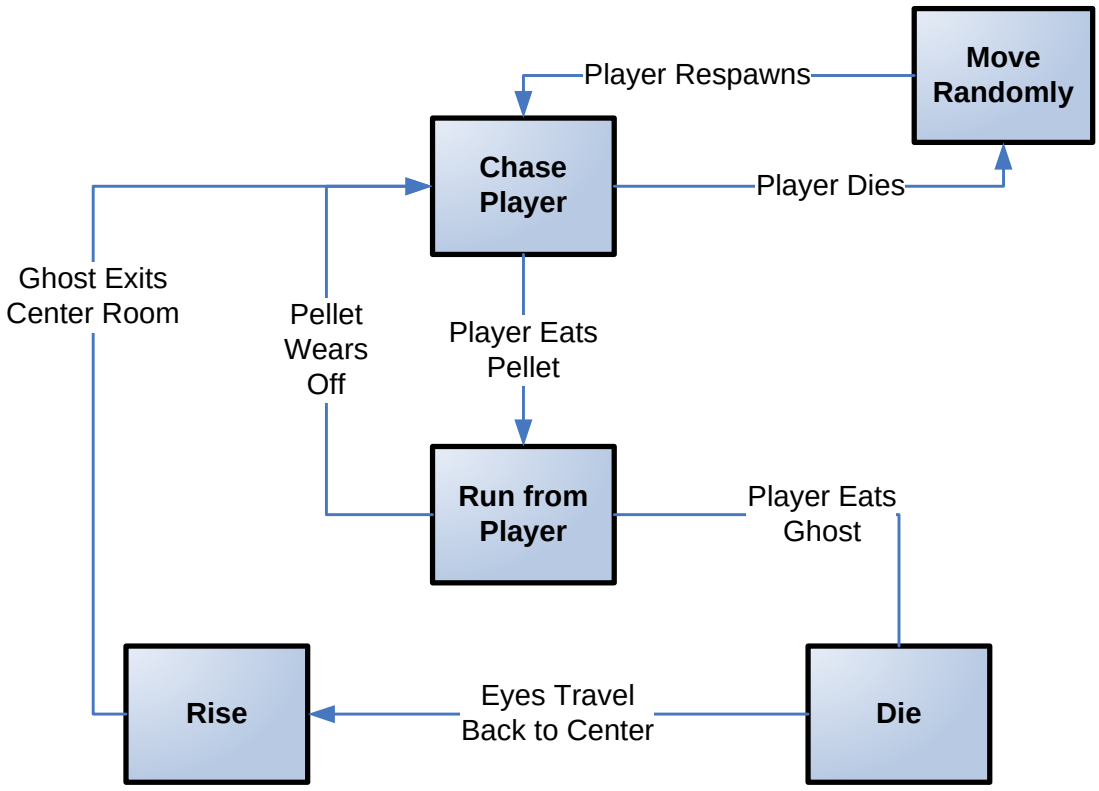


Ilustração B.1 - FSM do Blinky, PacMan

Esta máquina de estados é composta por cinco estados, são eles:

- **Rise** – que significa “nascimento” / ”ascensão”, representa o estado inicial do Blinky, enquanto está nele o fantasma busca sair do seu ponto inicial, o *Center Room*. Ao conseguir um evento é disparado “*Ghost Exits Center Room*”, que é a transição que irá fazê-lo mudar de estado.

- **Chase Player** – que significa “perseguir jogador”, este estado pode ser acionado de três formas, logo quando o fantasma sai do ponto incial, quando o jogador reaparece no mapa, ou quando o efeito do “*pellet*” (um bônus que faz com que o jogador fique invencível por um instante de tempo) acaba. Enquanto está nesse estado o fantasma fica seguindo o *Pac Man* ininterruptamente, até que uma das duas condições transição seja satisfeita, o próximo estado dependerá de qual evento será disparado (“*Player Eats Pellet”* ou “*Player Dies”*).

- **Run from Player** – significa “fugir do jogador”, entra-se neste estado quando o jogador come o “pellet”. Neste momento o fantasma pára de perseguir e começa a fugir do jogador. Há duas formas, duas condições, de sair desse estado, passar o efeito do “pellet” (o fantasma retorna ao estado “Chase Player”), ou ser comido pelo *Pac Man*, que dispara o evento que levará o para o estado “*Die*”.

- **Move Randomly** – estado onde o Blinky move-se aleatoriamente, pois a condição de entrada deste estado é que o jogador tenha morrido, logo o “Chase Player” não é mais válido, visto que não há mais nada a perseguir. Sai-se desse estado quando o jogador reaparece no mapa, disparando o evento “*Player Respawns*”.

- **Die** – significa “morrer”, para se atingir esse estado o Blinky deve que ter sido devorado pelo *Pac Man*, após isso o fantasma passa a ser representado por dois olhos que voltam ao ponto inicial, e fazendo-o voltar ao primeiro estado, recomeça todo o processo.

Como se pode ver é um técnica simples, entretanto bastante útil e poderosa. Por isso é amplamente utilizada em jogos, inclusive os de estratégia. O comportamento das unidades pode ser facilmente mapeado de forma semelhante ao Blinky. Um exemplo de FSM em um RTS poderia ser a modelagem de um aldeão (coletor de recursos). Poderia se especificar quatro estados:

- **Procurando** – estado onde o agente andaria pelo mapa buscando uma fonte de recursos (minas de ouro, árvores, animais, etc.). Ao encontrar seria disparado um evento “Recurso Encontrado”, por exemplo, e ele mudaria de estado.

- **Coletando** – ao encontrar a fonte de recursos o agente entraria neste estado e começaria a coletá-los. Haveria duas condições de saída deste estado, “Quantidade Máxima Atingida” e “Sendo Atacado”. O primeiro seria disparado quando o limite máximo de recursos fosse atingido pelo agente, fazendo-o ir para o estado “Retornando”. E o segundo seria disparado quando uma unidade inimiga começasse a atacá-lo, que o faria ir para o estado “Fugindo”.

- **Retornando** – quando o agente já está carregado de recursos ele tem que trazê-los de volta a sua base, este estado representa este comportamento. Ao depositar os recursos na base ele retornará para o estado “Procurando” (numa visão bem simplificada).

- **Fugindo** – estado onde o aldeão começa um comportamento de fuga, pois está sendo atacado pelo oponente. O evento que o faz mudar novamente de estado seria “A salvo”,por exemplo, que faria o agente voltar a procurar por recursos.

Este exemplo é uma visão bem simplificada de como usar um FSM em um jogo RTS, mas esta máquina de estados poderia ser bem mais elaborada. Por exemplo, poderia levar em consideração informações previamente obtidas, e ao sair dos estados “Retornado” e “Fugindo”, o aldeão já retornaria ao ponto onde estava, resgatando o estado anterior. Também seria possível acrescentar novos estados detalhando mais os problemas a serem tratados. Essa flexibilidade das FSM as torna bastante versáteis e poderosas, podendo atacar desde problemas simples aos mais complexos.

B.1.2 FUZZY-STATE MACHINES (FUSMS)

Uma Máquina de Estado Fuzzy, conhecidas como FuSM, é uma variante das máquinas de estado finitas. A diferença básica entre elas é fato das FuSMs serem baseadas na lógica Fuzzy. Esta que é comumente definida como um superconjunto da lógica convencional (boolena), onde só existem duas possibilidades “verdadeiro” ou “falso”. Este conceito foi estendido na lógica de Fuzzy permitindo verdades parciais.

Como as FSM, as FuSM possuem uma lista de estados, mas na primeira só existe um único estado atual e de acordo com as transições muda-se para os outros. Já na segunda isto não ocorre, nela o que existe é a possibilidade de se estar em vários estados no mesmo instante, logo não há transições. Cada estado em um sistema fuzzy calcula um

nível de ativação, o qual determina o quanto o sistema está em cada estado. Desse modo, o comportamento final é definido por uma combinação de todos os estados em que se está. As FuSM são menos abrangentes que as FSM, e sua utilização não é tão geral.

FuSM são úteis apenas em sistemas em que se possa estar em mais de um estado num mesmo instante e ter mais do que valores simples, como “cheio” ou “vazio”, “fechado” ou “aberto”, “vivo” ou “morto” [Schwab 2004]. Valores fuzzy representam coisas que se assemelham com: “um pouco”, “quase fechado”, “perto de morrer”, etc. Os jogos RTS se enquadram neste tipo de sistema, logo são candidatos ao uso, assim como das FSM, de Máquinas de Estado Fuzzy. O exemplo a seguir mostra um a utilização da FSM, e a evolução do mesmo problema sendo tratado com FuSM.

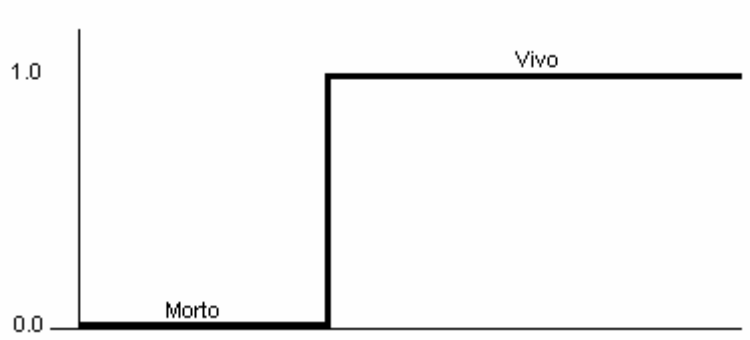


Ilustração B.2 - Representação de estado na lógica booleana

Na Ilustração 19 tem-se uma representação de como são tratados os estado na lógica booleana, onde o valor 1.0 equivale a “verdadeiro” e o 0.0 equivale ao “falso”, essa abordagem é utilizada nas FSM. A ilustração representa dois estados de uma unidade (“Morto” e “Vivo”), onde uma transição o faria alternar entre estes estados, como visto no tópico anterior.

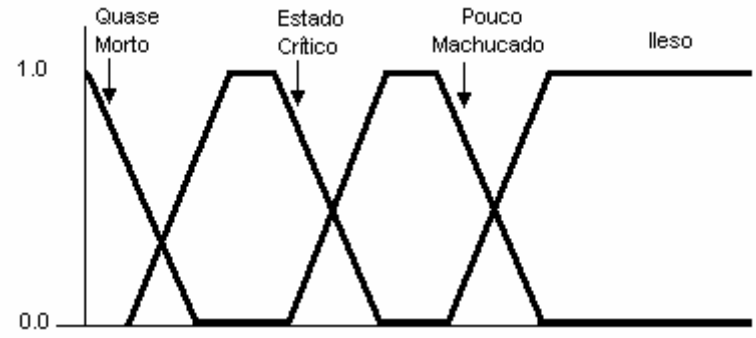


Ilustração B.3 - Representação dos estados na lógica fuzzy

Na Ilustração 20 está representado como é possível estender a FSM utilizando a lógica de fuzzy. Os valores 1.0 e 0.0 têm os mesmo significados do anterior (“Verdadeiro” e “Falso), mas o grande diferencial são os estados intermediários. É possível que se esteja, no mesmo instante, no “Estado Crítico” e “Quase Morto”, se o estivesse em torno de 0.3, por exemplo. Isto permite uma gama de novas possibilidades. É possível se modelar uma unidade militar, um arqueiro, por exemplo, de modo que ele consiga fugir e atacar ao mesmo tempo. Pode parecer confuso se estar fugindo e atacando, mas ao se analisar a situação conclui-se que é uma modelagem mais realista. No estado de fuga o arqueiro teria que andar na direção oposta do oponente. Mas o que o impediria de andar atirando flechas no inimigo? Isto torna o jogo muito mais realista, do ponto de vista do comportamento dos agentes.

B.1.3 MESSAGING

No desenvolvimento de jogos, somente uma técnica é mais usada que as máquinas de estado. É o uso de mensagens (ou eventos, como também são chamados) [Schwab 2004], ou *messaging*. Seu objetivo é manter as unidades informadas do estado e das mudanças do “Mundo” de forma mais eficiente. Seu conceito é bastante simples, suponha duas unidades A e B, onde uma tem que saber o estado da outra. Ao invés de A ter que ficar acompanhando o andamento de B todo o tempo, ela será informada, receberá uma mensagem, sempre que o estado de B for alterado. Dessa forma não há desperdício de processamento, pois só serão executadas operações quando houver informações importantes a serem obtidas. A técnica de *Messaging* é mais uma técnica de comunicação que uma de IA, mas é extremamente usada, principalmente em sistemas multiagentes. É bastante útil para melhorar organização, otimização, e facilitar comunicação entre as entidades.

Os jogos de estratégia envolvem, sobretudo, a comunicação entre as unidades e elementos do jogo. Esta é fundamental para delegar responsabilidades e atividades à outras unidades, descobrir quais fornecem determinados serviços, obter informações gerais sobre estado do mundo, entre várias outras funções [Vieira 2005].

Uma abordagem bastante utilizada em jogos RTS, e também em outros estilos, é o chamado *BlackBoard*, termo que vem do inglês e significa quadro-negro. Este nome vem do seu funcionamento, que é análogo ao de um quadro-negro. Os agentes ao obterem informações relevantes para os outros as publicam (escrevem) no *BlackBoard*, de modo que os outros tenham acesso as informações conseguidas.

Por exemplo, uma unidade é mandada para explorar o mapa, em determinado local ela encontra a base inimiga. Esta informação é publicada no *BlackBoard*, e quando um ataque for feito, todas as outras unidades já saberão o ponto exato onde o ataque deve ser executado.

Existem ainda diversas formas de estender esta técnica, como o *Message Priority*, que consiste em associar prioridade às mensagens enviadas. Por exemplo, uma tropa é enviada para executar um ataque na base do oponente, mas após elas saírem sua base começa a ser atacada. Neste caso uma mensagem com prioridade máxima é enviada à tropa ordenando que voltem para defender a base. Esta abordagem difere do *BlackBoard*, pois nela a ação da unidade é interropida, dependendo da prioridade, quando uma nova mensagem é enviada. Já na outra o agente tem que requisitar novas atualizações do mundo, as quais estão centralizadas no "quadro-negro", poupando o processamento de ter procurá-la.

B.1.4 PLANNING

Planning, ou Planejamento, é definido no dicionário como: "elaboração, por etapas, com bases técnicas, de planos e programas com objetivos definidos". Ou seja, é o ato de estruturar todas as ações que serão tomadas, geralmente fundamentadas em uma base de conhecimentos prévia. O planejamento envolve saber em qual estado se está, e qual estado se deseja atingir, e então achar uma seqüência de ações que possibilitem esta transição entre o estado inicial e o desejado. Ele é visto como uma das principais qualidades humanas, entretanto, sistemas de IA bastante avançados estão cada vez mais começando a planejar de forma mais inteligente e aproximando-se dos humanos.

É uma técnica que tem grande importância na Inteligência Artificial de jogos RTS. Planejamento começou a ser seriamente usado nestes jogos devido ao avançado nível de estratégias utilizadas, para realizar tarefas de alto nível (por exemplo, defender o lado esquerdo da base contra ataques aéreos) algumas tarefas prévias têm que ser adicionadas ao planejamento da IA. Neste caso, além de ter que cuidar da quantidade de recursos que deve ser gasta, teria-se que ter concluído todo o processo de evolução necessário para executar a tarefa ("defender contra ataques aéreos"). Lidar com a árvore de tecnologias e escolha das evoluções é apenas uma área do planejamento.

Suponha, em *Age of Empires II*, o computador manda um dos seus barcos explorar o oceano buscando as docas do inimigo. Neste momento passa por ele um *War Galleon* (traduzindo, uma Galera de War, que é o barco mais evoluído da civilização), que atira grandes flechas de fogo e é bastante resistente. O computador agora sabe que o inimigo já possui este tipo de barco tão evoluído, e começa a planejar como ele poderá derrotá-lo. Há duas opções, ou ele evolui seus barcos para ficarem no mesmo nível do inimigo, ou constrói em sua orla *BombardTowers*, que são torres de defesa extremamente fortes. Tendo a árvore de tecnologia da sua civilização, que descreve os pré-requisitos necessários para evoluir uma determinada unidade ou construção, a IA elabora o caminho, passo a passo, que deve ser seguido para atingir o objetivo.

B.1.5 HIERARCHICAL AI

Não é de se espantar o fato dos RTS terem as maiores demandas do ponto de vista de IA. Ela precisa atingir a expectativa do jogador, de um jogo de estratégia desafiante, ainda tem que tomar decisões em milisegundos para atingir os requisitos do jogo. A abordagem da IA Hierárquica tem sido muito bem sucedida visando atingir essas necessidades [Gamasutra].

Na IA Hierárquica há diferentes camada de Inteligência Artificial. A IA estratégica (*Strategic AI*) que toma decisões de alto nível, como “Quais unidades devem ser criadas?”. A IA Tática (*Tactical AI*) que executa as ordens dadas pela estratégica da melhor maneira possível, decidindo coisas como: “Onde é o melhor lugar para criar as unidades solicitadas?”. Há também uma terceira camada, que é chamada IA da Entidade (*Entity AI*) [Gamasutra]. Ela representa as unidades do jogo, como soldados e grupos, e é manipulada pela Tática. A IA Hierárquica consiste, basicamente, em dividir os problemas da Inteligência Artificial em níveis, para torná-los mais simples de serem tratados. Com essa abordagem o nível mais elevado (IA Estratégica) se preocupa apenas com decisões alto nível, e as informa a sua subordinada, um nível abaixo (IA Tática), que as executa, ou ainda repassa a ordem para o nível seguinte. Dessa forma cada camada é responsável única e exclusivamente pelos seus problemas.

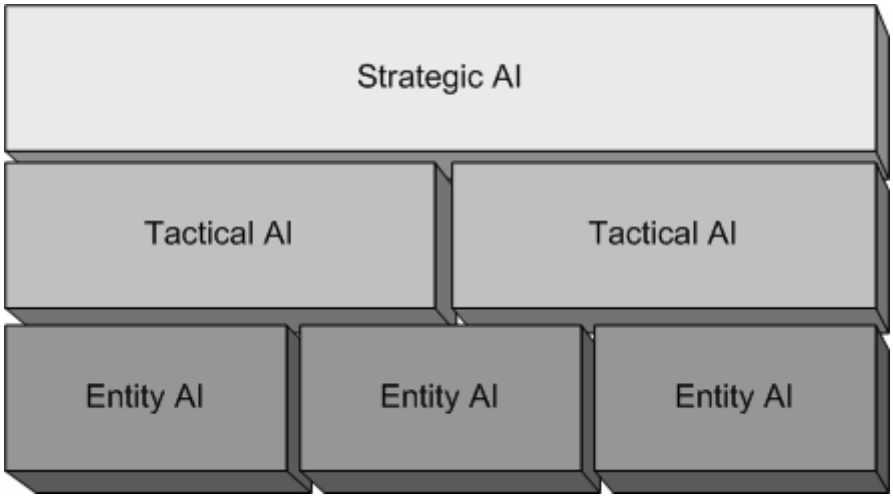


Ilustração B.4 - Exemplo de estrutura da IA Hierárquica

Com a evolução dos jogos a utilização de técnicas como esta está sendo aprimorada. Por exemplo, em *Cataclysm*, é largamente usada a idéia de combinar comportamentos mais simples [Gamasutra]. O nível de agressividade da unidade, controlado pela *Entity AI*, define o quão longe a unidade irá perseguir um oponente. Se as unidades estiverem com a agressividade muito baixa, elas irão evitar o contato com inimigos durante suas patrulhas. *WarCraft 3* [War3] é outro jogo que faz uso de técnicas como esta, nele confia-se muito em agentes com comportamentos autônomos como uma forma de aliviar o “peso” que recai sobre as camadas de mais alto nível.

B.1.6 FORMAÇÕES

Os humanos logo aprenderam que combater em grupo é mais eficiente que combater sozinho. Desde os primórdios, já nas caçadas era possível perceber movimentos coordenados dos povos, as formações. Atualmente, formações são esperadas de qualquer tipo de movimentação de um grupo, em uma batalha. Esse tipo de comportamento foi logo trazido para dentro dos jogos RTS, visto que eles simulam guerras, e nelas as formações são usadas extensivamente.

Formações, em jogos de estratégia, podem ser úteis no combate como forma de proteger algumas unidades, deixá-las em uma posição privilegiada durante o ataque, para confrontar a formação escolhida pelo pelotão do adversário. A Inteligência Artificial tem que analisar diversos fatores, antes de decidir qual a melhor a ser utilizada naquele determinado momento, como: “que tipos de unidade compõem o pelotão?”, “está atacando ou defendendo?”, “que tipos de unidade têm o pelotão oponente?“, “qual a formação adotada pelo oponente?”, entre outros fatores. Mas esta técnica não é apenas útil no combate, ela ajuda muito na redução do processamento que a IA requer. Por exemplo, em diversos jogos

na movimentação de tropas pelo mapa a formação comumente utilizada, caso não seja definida uma outra, é a em Coluna (ver ilustração a seguir), ou Cobra. Isto porque as unidades fazem uma fila, e o calculo do caminho é feito apenas para as unidades da frente, as demais apenas as seguem.

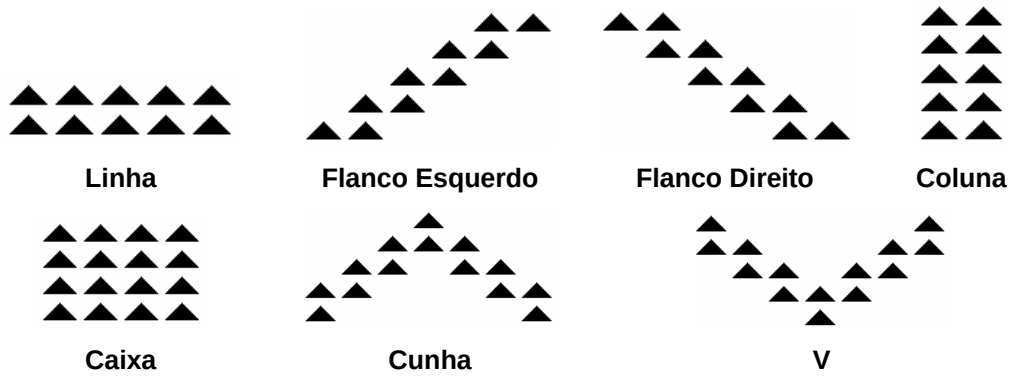


Tabela B.1 - Exemplos de Formações

Destas formações, as mais comuns nos jogos são: Coluna, pela sua grande ajuda na movimentação; Linha, por serem bastante úteis para segurar um ataque formando uma "parede"; e Caixa, que é comumente usada quando existem unidades variadas no pelotão, onde as mais frágeis ou importantes são posicionadas no centro, dificultando o ataque do inimigo a elas. Formações acrescentam bastante a qualquer jogo que utilize movimento coordenado de unidades.

O combate em *Age of Empires III* [AoE3] enfatiza bastante a questão de formações, muito mais que sua versão anterior. Dependendo da formação que o pelotão adote, ele poderá ganhar uma grande vantagem sobre o inimigo, se tornando menos vulnerável.

Ilustração B.5 - *Age of Empires III*, demonstração do uso de formações, Caixa neste caso.

B.1.7 MAPA DE INFLUÊNCIA

Mapas de Influência, também chamados de IM (acrônimo de *Influence Map*, seu nome original em inglês), estão, aos poucos, se tornando uma das técnicas de IA mais comumente utilizadas em jogos. É uma estrutura de dados bem simples de implementar e bastante fácil de adaptar a diferentes problemas. O termo mapa de influência se refere ao uso de um simples array de dados (uma tabela, de tamanho equivalente ao mapa), onde cada elemento é mapeado para representar as informações sobre uma posição específica do mundo [Schwab 2004]. Os dados desse array são analisados levando-se em conta alguns fatores (inimigos, geografia, etc.), e para cada posição é calculado valor, e utiliza-se uma função para propagar este valor. Dessa forma, ao terminar este processo para todas as posições, tem-se uma estrutura de dados com valores que representam as influências, daí o nome da técnica, de uma posição sobre a outra, de acordo com algum critério específico.

Na ilustração a seguir tem-se um exemplo de mapa de influência. Nele existem três unidades e dois times. Um time possui uma unidade representada pelo triângulo com o valor 10, que representa seu poder de ataque. O outro possui duas unidades, representadas pelos triângulos brancos, com valor -5, representando também seu poder de ataque. O sinal negativo é apenas um indicativo que pertence a time opostos. Neste exemplo a função de propagação utilizada, aplicada recursivamente, é dividir, por dois, o valor da posição e atribuir o resultado a cada um de seus vizinhos. Ao se terminar os cálculos obtém-se este mapa de influência, onde as posições com valores positivos pertence a um time, as com valores negativos ao outro. E as posições contendo zeros, representam os territórios neutros, que não são influenciados por nenhum dos dois times.

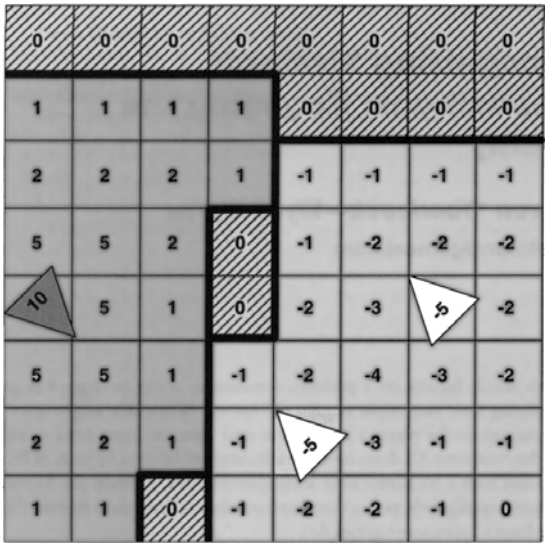


Ilustração B.6 - Mapa de Influência básico, baseado em forças de ataque [Rabin 2002].

Esta técnica é extremamente útil para a tomada de decisões estratégicas do jogo. Através dela é possível inferir informações sobre as estratégias e táticas adotadas pelo oponente. Veja a figura a seguir, uma abordagem mais detalhada do exemplo anterior.

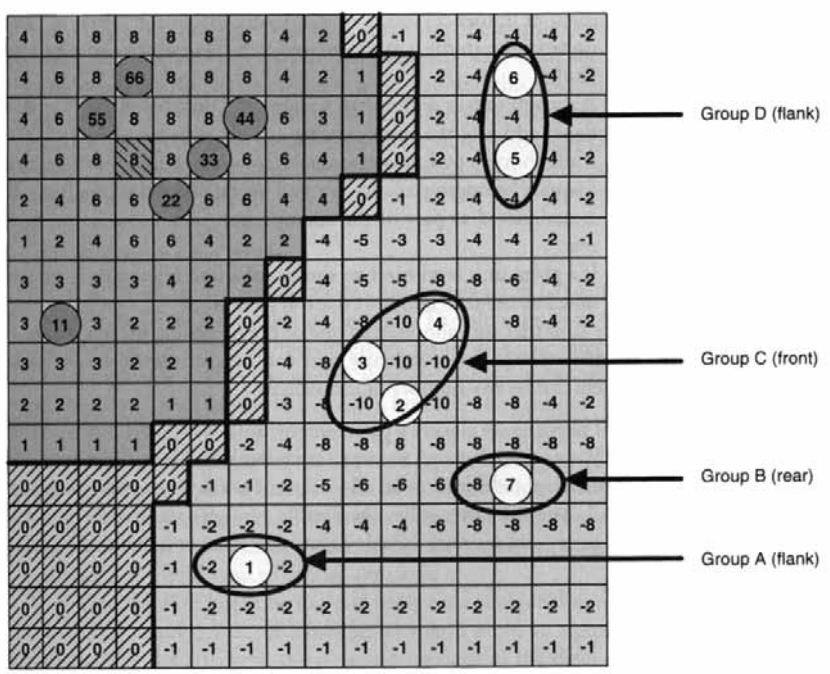


Ilustração B.7 - Abordagem detalhada do Mapa de Influência [Rabin 2002].

Nesta abordagem é feito um tratamento mais detalhado das informações do mapa. Ao se analisar é possível perceber o posicionamento das tropas inimigas, identificando uma possível estratégia do oponente. Este tipo de informação é fundamental para montar sua própria estratégia e defender-se de ataques de forma mais eficiente. Por exemplo, é possível perceber que o "*Group A*" está mais desprotegido. Possui o menor valor (Poder de Ataque) e ainda encontra-se próximo a uma zona neutra (valores igual a zero), ou seja, está distante de outras tropas inimigas. A partir destas informações o computador pode tomar a decisão de atacar esse grupo, aumentando sua chance de sucesso bastante alta.

Outra implementação útil de IM é manter registros de eventos acontecidos durante o jogo. Essas informações podem interferir no comportamento, de modo que a IA passe a não executar sempre as mesmas ações. Por exemplo, se em um RTS o jogador constrói uma torre num caminho frequentemente utilizado pelas unidades inimigas, e começa a matar todas as unidades que passam por perto. Ao guardar informações sobre estes eventos é possível que a IA adote outro comportamento para fugir dessa armadilha, ou ainda melhor, mandar uma tropa sua para destruir a torre inimiga.

Outra utilização bastante comum de Mapas de Influência é a Análise de Terreno. Isto é frequentemente usado em jogos RTS para achar “becos” no mapa, pontos de emboscada onde se possa ter uma vantagem ao atacar o inimigo. Estes becos podem também ser usados na construção das muralhas, pois permite o fechamento de uma região utilizando menos recursos. Outro uso importante dos IM nos jogos de estratégia é determinar a melhor forma de se construir a cidade [Schwab 2004]. Elas devem ser feitas com certo planejamento, visando evitar grandes aglomerados, pois pode vir a causar problemas para encontrar caminhos devido ao grande número de colisões entre as unidades. Analisar um mapa encontrar e áreas importantes (posições estratégicas) é algo que os humanos fazem de forma intuitiva e bastante eficiente. Daí a importância desta técnica nos jogos, tentar modelar o comportamento do computador da forma mais parecida possível com um jogador real.