MINISTÉRIO DA DEFESA

EXÉRCITO BRASILEIRO

DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA

INSTITUTO MILITAR DE ENGENHARIA

CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇÃO

ROBERTSON SCHITCOSKI

UMA ARQUITETURA MODULAR PARA SISTEMAS DE TREINAMENTO MILITAR EM OPERAÇÕES TÁTICAS

Rio de Janeiro

2009

INSTITUTO MILITAR DE ENGENHARIA

ROBERTSON SCHITCOSKI

UMA ARQUITETURA MODULAR PARA SISTEMAS DE TREINAMENTO MILITAR EM OPERAÇÕES TÁTICAS

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientador: Prof. Jauvane C. de Oliveira – Ph.D.

Co-orientador: Prof. Paulo F. F. Rosa – Ph.D.

Rio de Janeiro

2009

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 – Praia Vermelha
Rio de Janeiro - RJ CEP: 22290-270

S337u Schitcoski, R.

Uma Arquitetura Modular para Sistemas de Treinamento Militar em Operações Táticas / Robertson Schitcoski – Rio de Janeiro: Instituto Militar de Engenharia, 2009.

104 p.:il.

Dissertação (mestrado) – Instituto Militar de Engenharia – Rio de Janeiro, 2009.

1. Operações táticas – Treinamento militar. 2. Formação militar.

CDD 355.42

INSTITUTO MILITAR DE ENGENHARIA

ROBERTSON SCHITCOSKI

UMA ARQUITETURA MODULAR PARA SISTEMAS DE TREINAMENTO MILITAR EM OPERAÇÕES TÁTICAS

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

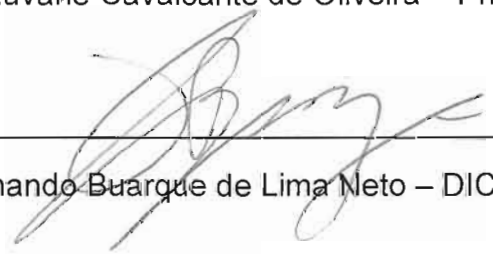
Orientador: Prof. Jauvane Cavalcante de Oliveira – Ph.D.

Co-orientador: Prof. Paulo Fernando Ferreira Rosa- Ph.D.

Aprovada em 28 de janeiro de 2009 pela seguinte Banca Examinadora:

Prof. Paulo Fernando Ferreira Rosa – Ph.D. do IME – Presidente

Prof. Jauvane Cavalcante de Oliveira – Ph.D. do LNCC



Prof. Fernando Buarque de Lima Neto – DIC Ph.D. da UPE

MAJ Ronaldo Moreira Salles – Ph.D. do IME

Rio de Janeiro

2009

À minha esposa Cristiane, que me apoiou e motivou nos momentos de maior dificuldade,

e ao meu filho Rafael, que veio trazer uma alegria extra na conclusão deste trabalho.

AGRADECIMENTOS

Agradeço a todas as pessoas que me incentivaram e me apoiaram, permitindo a consolidação deste trabalho.

Em especial ao meu Professor Orientador Jauvane Cavalcante de Oliveira, que me deu a motivação inicial e enriqueceu meu trabalho com críticas e sugestões essenciais.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

RESUMO

Desde os primórdios da civilização, os jogos de guerra têm sido empregados como uma importante ferramenta de simulação, para generais, e doutrinação, para seus comandados. No princípio, os jogos de guerra se limitavam a peças de madeira ou osso dispostas em tabuleiros ou caixas de areia. Com a evolução das tecnologias militares os jogos se tornaram mais complexos, tornando necessário o uso de vários manuais para decidir os combates simulados. Com o advento da Segunda Guerra Mundial, surgiram os computadores, empregados a princípio em cálculos balísticos e na quebra de códigos dos inimigos. Logo se percebeu que eles poderiam ser empregados como uma ferramenta de simulação de combates. Por muito tempo os principais jogos de guerra que eram executados pelos computadores foram os simuladores de veículos militares. Com o avanço das tecnologias computacionais e o advento dos jogos voltados para o entretenimento, começaram então a surgir vários jogos de guerra táticos e estratégicos, capazes de simular as mais variadas condições de combate presentes nos campos de batalha.

O Exército Brasileiro ainda se encontra nos estágios iniciais de desenvolvimento de jogos de guerra, estando concentrado, sobretudo, em sistemas de treinamento a nível estratégico, com jogos como o AZUVER, o SISTAB e o SABRE. Levando tal fato em conta, este trabalho tem como proposta uma arquitetura de software que sirva como modelo de projeto e incentivo a pesquisa para o desenvolvimento de softwares de simulação táticos para o Exército Brasileiro.

A arquitetura aqui proposta foi fundamentada nos princípios modernos de engenharia de software, empregando o paradigma de orientação a objeto e os padrões de projeto. Ela é derivada do estilo arquitetural de sistemas centrados em dados, composta por um núcleo de gerenciamento de objetos e vários módulos independentes. Esses módulos são responsáveis pela manipulação dos dados dos seguintes domínios lógicos: entrada de dados, interface do usuário, gráficos tridimensionais, rede, inteligência artificial, áudio e efeitos físicos.

ABSTRACT

Since the beginning of the civilization, wargames has been used like an important simulation tool, for generals, and indoctrination, for their soldiers. At first, wargames were limited to be only wood or bone pieces disposed on boards or sandboxes. With the evolution of military technologies the games became more complex, making necessary the use of many manuals to take decisions about the results. With the advent of the World War II, the computers rose to be used in the beginning, in ballistics and to break enemy codes. Then, it was noted that computers could be used as a tool for combat simulations. For a long time the main wargames running on computers were military vehicles simulators. With the evolution of computer technologies and the rising of entertainment games, many tactical and strategical wargames were produced, able to simulate the most adverse combat situations in the battlefields.

The Brazilian Army is at the first steps of wargame development and is focused on training systems at the strategic level, e.g. games AZUVER, SISTAB and SABRE. Considering this fact, in this work we introduce a software architecture to be used like a model and incentive to research and development of tactical simulation softwares for the Brazilian Army.

The architecture proposed here is based on modern principles of software engineering and design patterns. It is derivative from the data centered architectural style, and is composed by an object management core surrounded by some independent modules. These modules are responsible for manipulation of data from the following logic domains: user input, user interface, three-dimensional graphics, network, artificial intelligence, sound and physics.

1 INTRODUÇÃO

1.1 UM BREVE HISTÓRICO DOS JOGOS DE GUERRA

O exército é provavelmente a organização com a mais longa história de utilização de jogos para o treinamento de seus membros. O xadrez (FIG 1.1), cujas origens datam do século VII, é um dos mais bem conhecidos jogos de treinamento militar. Embora altamente estilizado, o xadrez é considerado uma das melhores representações de combates da era pré-armas de fogo. Oficiais em treinamento recebiam aulas de xadrez na esperança de que melhorassem seu desempenho no campo de batalha. Mesmo o xadrez, na verdade se originou em jogos de guerra mais antigos ainda, de milhares de anos atrás, o que não deixa dúvida de que guerra e jogos têm andado lado a lado ao longo de toda a história da humanidade.

FIG.1.1 Peças e tabuleiro de Xadrez

De acordo com autores Dunnigan e Perla, o moderno jogo de guerra surgiu no século XVII. Esses jogos começaram como simples variações do xadrez, evoluindo o conjunto de peças de modo a representar unidades militares contemporâneas e adicionando maior complexidade ao terreno. Nos séculos seguintes, porém, surgiram simulações muito mais complexas e sofisticadas, muitas das quais

requeriam variadas estatísticas e tabelas cobrindo uma miríade de possibilidades. Ao contrário do xadrez, poucos desses jogos se difundiram no mundo civil como forma de entretenimento.

Com a Segunda Guerra Mundial surgiram as primeiras máquinas computacionais, os primeiros computadores, que viriam a ser empregados para o desenvolvimento dos mais complexos e avançados sistemas de simulação já utilizados. E com o avanço quase exponencial da tecnologia computacional do último século, sistemas bastante realistas têm auxiliado cada vez mais os exércitos no treinamento de seus contingentes. (DUNNIGAN, 1992) (PERLA, 1990)

1.2 A EVOLUÇÃO DOS JOGOS DE GUERRA

O xadrez é considerado por muitos como o jogo de guerra original, o que não é verdade. No seu livro *The Art of Wargaming* (PERLA, 1990), Peter Perla traçou a origem do xadrez até o jogo de tabuleiro conhecido como *Chaturanga*, jogado pela nobreza indiana de cerca de quatro mil anos atrás. Acredita-se que *Chaturanga* foi desenvolvido mais ou menos na mesma época do jogo chinês *Wei Hei* que evolui para o clássico jogo japonês *Go* (FIG 1.2), cujo objetivo é cercar as peças do adversário, de modo a capturá-las, vencendo o jogador que obtiver o maior número de peças no tabuleiro.

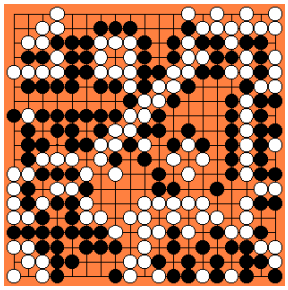


FIG.1.2 Tabuleiro do jogo Go

O aprendizado do xadrez nos séculos anteriores à invenção da pólvora forneceu para os novos oficiais da aristocracia uma visualização de como eles poderiam empregar diferentes tipos de tropas. Este aprendizado, limitado pelas regras simples de interação entre as peças, ao menos permitia que esses oficiais fossem para a batalha com um plano. O velho ditado de que “nenhum plano de batalha sobrevive a um contato com o inimigo” significava que os planos mudavam durante uma batalha, mas mesmo essa mínima preparação poderia significar a diferença entre a vitória e a derrota.

A realidade condensada de um jogo remove as distrações que poderiam de outro modo impedir o vislumbre de novas idéias que podem ser aplicadas no mundo real. Utilizando um jogo de xadrez, um comandante pode experimentar diferentes tipos de estratégias para compreender como elas funcionariam no mundo real sem arriscar nenhuma de suas tropas ou outros recursos valiosos. Visualizando o “campo de batalha” como um todo ele não é distraído ou desviado por pequenos aspectos do conflito, podendo então se concentrar na criação de um plano de batalha abrangente.

Por outro lado, deve ser considerado que os jogos podem, em muitos casos, eliminar ou não permitir uma representação adequada de certos detalhes de uma batalha, como por exemplo, a moral de uma tropa. Esse tipo de fator depende muito de elementos subjetivos, como a capacidade de comando dos líderes quando em campo, além de outros elementos mais sutis, como tempo de permanência das tropas em campo e qualidade das rações disponibilizadas. Tudo isso pode ser decisivo numa batalha, e são detalhes que devem ser submetidos a outros tipos de análises.

Outro modo como um jogo de guerra pode auxiliar um comandante é colocando ele contra outro comandante. A interação entre um plano e outro contra-plano dá ao comandante a chance de que ele descubra furos na sua estratégia, que

de outro modo ele não perceberia. Como o outro jogador também está tentando ganhar, ele procurará por fraquezas que possa explorar em seu adversário.

O valor desse aprendizado depende de quão bem um jogo representa a realidade na qual ele se baseia. Por essa razão a maioria dos jogos modernos não possui a simplicidade e jogabilidade do xadrez. O xadrez possui somente seis diferentes peças (rei, rainha, bispo, torre, cavalo e peão) e uma simples matriz de 8x8 casas de duas cores como tabuleiro. Mesmo considerando que cada uma de suas seis peças possui suas próprias regras de movimento e ataque, comparado às complexas simulações usadas pelo Exército Prussiano no século XIX, o xadrez é infantil em sua simplicidade.

O *Kriegspiel*, desenvolvido na primeira metade do século XIX pelo tenente George Heinrich Rudolph Johann Von Reisswitz e baseado em um jogo de guerra desenvolvido pelo seu pai, o Barão Von Reisswitz, teve uma carreira militar mais longa que o seu criador. O jogo usava mapas topográficos na escala 1:8000 e tiras de metal para representar as tropas. Através de um árbitro, o jogo mantinha a "neblina de batalha", mostrando no mapa somente as tropas que podiam ser vistas pelos combatentes. (LENOIR, et al., 2005)

Após as vitórias prussianas na "Guerra das Seis Semanas" contra a Áustria em 1866 e na Guerra Franco-Prussiana de 1870-1871, o resto do mundo descobriu o *Kriegspiel.* Militares de outros países começaram a examinar o *Kriegspiel* modificando-o de modo a adaptá-lo as suas próprias necessidades de treinamento e mantendo-o atualizado com as armas mais modernas. Porém esses jogos inspirados no *Kriegspiel* estavam destinados a se tornarem cada vez mais complexos. Adaptados para acomodar armas como metralhadoras, carros de combate, caças, bombardeiros, porta-aviões e táticas como forças combinadas, comando e controle, e mais, a complexidade subiu a níveis sem precedentes. (CAFFREY, 2000)

Num típico jogo de guerra moderno, as regras de movimentação de toda a variedade de armas e tropas disponível, as tabelas para determinação do resultado dos conflitos entre unidades individuais, considerando cada variação de condições de terreno e de encontro, requerem páginas de texto suficientes para preencher um livro. A mudança de foco das operações militares da destruição da força oponente para a destruição da capacidade de luta do inimigo adicionou ainda mais casos que devem ser considerados. Jogar tal jogo de guerra, mesmo uma única batalha, pode levar horas ou mesmo dias, à medida que os jogadores têm de consultar regras e tabelas para posicionar e mover suas unidades e então resolver os encontros entre as unidades.

Com tal complexidade de interação e julgamento, os computadores se tornaram a solução adequada para os jogos de guerra. Os computadores atuam tanto como um juiz imparcial quanto como um incansável analisador de regras, que assegurará que os jogadores sigam em cada situação as regras apropriadas de movimentação, efeitos das armas, leis físicas e assim por diante. Jogos de guerra civis se aproveitaram desse perfeito papel dos computadores tão logo os microcomputadores se tornaram disponíveis. Os militares, porém, levaram mais tempo para se aproveitar dessas tecnologias.

Após a Segunda Guerra, o uso de jogos de guerra de computador pelo Exército Norte Americano foi limitado à “pesquisa operacional” a nível estratégico. Pesquisa operacional é um tipo de análise de sistemas que envolve ciência moderna, matemática e estatística na resolução de um problema específico. Neste caso em particular, o problema era a defesa nacional na era nuclear. Uma das mais amplamente utilizadas simulações de pesquisa operacional foi o *ATLAS*, desenvolvido pelo Departamento de Defesa Norte Americano (*Department of Defense – DoD*) nos primórdios dos anos 60 e utilizado até a década de 80. (KIRBY, 2003)

Ainda nos anos 60, no Departamento de Estudos Avançados da Divisão de Sistema de Mísseis Raytheon, uma equipe desenvolveu simulações de computador de batalhas aéreas, missões espaciais, troca de mísseis, sistemas de inspeção de desarmamento e disputas político-econômicas internacionais. (MICHAEL, et al., 2006)

A primeira Guerra do Golfo Pérsico, o fim da União Soviética na segunda metade do século 20, juntamente com o crescimento explosivo da renderização realística 3D em jogos de computador, marcaram o início do fim da antiga abordagem e a concentração em novas metodologias. Antes de se verificar quais foram essas novas metodologias é necessário examinar o uso militar de outra ferramenta comum em jogos de computador: o simulador.

Simuladores, especialmente simuladores de veículos, têm sido há muito tempo uma ferramenta essencial no treinamento militar. Simuladores de vôo são o exemplo mais conhecido do uso militar de simulações extremamente precisas no treinamento de pessoal. Os melhores exemplos de como tais simuladores são simples jogos são a série *Microsoft Flight Simulator* (FIG 1.3), e os jogos *Falcon* e *Comanche*.

FIG.1.3 Tela do jogo Microsoft Flight Simulator

Embora simuladores de vôo sejam considerados um passo necessário no treinamento de pilotos hoje em dia, levou quase duas décadas para que eles fossem aceitos como um modo viável de treinar novos pilotos. À medida que a experiência de pilotagem e comando de aeronaves durante a guerra crescia, observou-se que a taxa de mortes de pilotos em tempos de guerra estava relacionada com o número de vôos que o piloto já havia realizado. Para um novo piloto o primeiro vôo era o mais perigoso. À medida que um piloto voava mais e mais missões, a probabilidade de morte diminuía. Se pilotos veteranos que tinham experiência em combates podiam aprender através da experiência, então os simuladores de vôo poderiam permitir que os novatos se tornassem “veteranos virtuais” e ficassem mais bem preparados para enfrentar problemas em vôo. Assim, todo treinamento de vôo atual, seja militar ou civil, envolve centenas de horas de vôo em simuladores antes que os novos pilotos possam pilotar uma aeronave real.

Os primeiros simuladores de vôo, construídos antes da Primeira Guerra e durante os anos 20, eram puramente mecânicos e pneumáticos. Os céticos criticavam a eficiência dessas parafernálias de aspecto estranho. Somente quando o vôo por instrumentos e a navegação se tornaram importantes é que foi percebido o real valor dos simuladores de vôo. O treinamento de vôo por instrumentos em um simulador de vôo provou ser muito mais seguro para todos os envolvidos do que o simples treinamento com um avião real.

A Segunda Guerra, com a necessidade de treinar grande número de pilotos rapidamente, acelerou o desenvolvimento dos simuladores de vôo, bem como as subseqüentes necessidades militares geradas com a Guerra Fria. Com o passar dos anos os simuladores foram aprimorados. Primeiramente, foram empregados os computadores analógicos (ou analisadores diferenciais, como eram conhecidos na época) e então se seguiram os computadores digitais. Mais recentemente, têm sido empregadas redes de computadores nos simuladores de vôo e de outros veículos, o que tem disponibilizado uma nova gama de opções e possibilidades de treinamento.

Nos anos 80, o uso militar de redes de computadores nos simuladores tornou possível o treinamento de vários indivíduos no mesmo espaço de batalha, tanto como aliados como oponentes. Com essa mudança, de acordo com o Cel. Matthew Caffrey, reserva da Força Aérea Norte Americana e professor da *Air Command and Staff College* (ACSC), os simuladores passaram de adestradores puramente procedurais para adestradores táticos. (CAFFREY, 2000)

Hoje, todos concordam que os simuladores de vôo são ferramentas de treinamento muito úteis. Além disso, os simuladores são uma maneira segura de manter as habilidades dos pilotos atualizadas. Os militares empregam ainda outros tipos além dos simuladores de aviões de asa fixa e de helicópteros. Existem simuladores de carros de combate, simuladores de *humvee* (espécie de jipe de combate pesado) e muitos outros. Se o veículo pode ser pilotado ou dirigido, os modernos exércitos possuem um simulador para treinar seus soldados na sua operação e uma infra-estruturar para treiná-los em grupo.

1.3 O USO MODERNO DOS JOGOS DE GUERRA

Afirmar que os exércitos, particularmente dos Estados Unidos, estão apenas “interessados” em jogos de computador para treinamento é no mínimo uma afirmação ingênua. O Orçamento de 2003 para a Seção de Defesa Nacional dos Estados Unidos especificou uma verba de US$ 10 bilhões para treinamento. Para o Pentágono, são destinados anualmente US$ 4 bilhões para equipamentos de simulação e jogos de guerra. Outra prova do crescente comprometimento com os jogos de treinamento foi o encorajamento dos Estados Unidos, aos países da OTAN, para o uso de tais jogos, na conferência realizada em outubro de 2004, chamada de *Exploiting Commercial Games for Military Use* (Explorando Jogos Comerciais para Uso Militar). (NAT08)

Em 2002 o Exército Norte Americano criou o *America´s Army* (FIG 1.4). Baseado no motor gráfico do jogo *Unreal Tournament, America´s Army* provou ser uma das melhores ferramentas de recrutamento já criadas. O Exército precisava recrutar 80.000 novos soldados todo ano, e o jogo se mostrou um grande sucesso, custando apenas 15% dos outros programas de recrutamento. Uma pesquisa do Exército, realizada para verificar a eficiência do seu sistema de recrutamento, descobriu que dentre os adultos jovens, na faixa de 16-24 anos, o jogo lhes causou uma impressão muito mais positiva que os demais esforços de recrutamento do Exército. O Exército pretende realizar estudos a longo prazo para verificar nos novos recrutas se as habilidades aprendidas no *America´s Army* permanecem após o treinamento básico e se o jogo pode ser utilizado para direcionar as escolhas de carreira no Exército. (LENOIR, 2003)

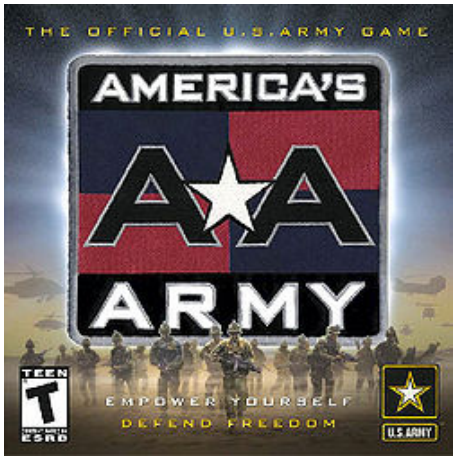


FIG.1.4 Logo do jogo America´s Army

No outono de 2004, o jogo *America´s Army* já tinha sido baixado mais de 17 milhões de vezes, tinha uma comunidade de quatro milhões de jogadores registrados (tanto civis quanto militares), com 100 mil novos jogadores novos aderindo a cada mês. Além disso, 30% dos jogadores de *America´s Army* não são norte americanos e sim de outros países, como Suécia, Alemanha, França e Brasil.

De acordo com o Coronel Casey Wardynski, diretor do *U.S. Army´s Office of Economic Manpower Analysis* (OEMA – Escritório para Análise do Poder Econômico Humano do Exército Norte Americano) e coordenador do Exército para supervisão do desenvolvimento do jogo, um dos principais objetivos do *America´s Army* era a verossimilhança, de modo a ser tão realista quanto possível. “Melhor do que isso”, disse o Cel. Wardynski em uma entrevista, “só se fosse um soldado real.” Embora o jogo sacrifique um pouco o realismo em nome do entretenimento (por exemplo, é possível se proteger seguramente de tiros de fuzil atrás de carros no jogo, o que é extremamente perigoso na realidade), a sensação no jogo é de inegável realismo. (Ame08)

Mesmo se a autenticidade de *America´s Army* dissuadir potenciais recrutas, ele trará algum benefício. No passado, estimava-se que 13,7 % dos novos recrutas desistiam antes de completar o treinamento básico, custando para o Exército em média US$ 15.000 por pessoa em investimento perdido. Ao mostrar o que acontece durante o treinamento básico e adiante, *America´s Army* apela para aqueles que têm maior probabilidade de permanecer no Exército e dissuade aqueles que não têm condições de completar o treinamento básico.

Além do uso como ferramenta de recrutamento, *America´s Army* tem auxiliado no pré-treinamento de novos recrutas. Desde seu lançamento, muitos dos novos recrutas que chegam em Fort Bening, Georgia, EUA, já estão familiarizados com a disposição do campo de treinamento e têm uma compreensão geral de como o treinamento irá ocorrer. Além do detalhamento de certos cenários, como Fort Bening, uniformes e armamentos no jogo são tão autênticos quanto possível.

Em 2003, o Exército expandiu a linha lançando *America´s Army: Special Forces* (FIG 1.5) para destacar o papel das Forças Especiais e, em 2005, lançou juntamente com a *Ubisoft* o jogo *America´s Army: Rise of a Soldier*, a versão de console para Playstation 2 e Xbox.

FIG.1.5 Tela do jogo America´s Army: Special Forces

Desde seu lançamento, *America´s Army* têm sido modificado e estendido para se tornar uma ferramenta de treinamento para ser usada por mais do que potenciais recrutas. Soldados na ativa têm utilizado o jogo para se prepararem para missões e até mesmo praticar o desarmamento de bombas. Além disso, o grupo de Aplicações Futuras do *America´s Army* o tem empregado para prototipar futuros sistemas de armas, como o Sistema de Armas XM25. Ao empregar o *America´s Army* com esses novos propósitos, o Exército Norte Americano têm obtido uma significativa economia de tempo e dinheiro. Por exemplo, o robô de treinamento TALON, atualmente em uso, levou de 2 a 3 meses para ser desenvolvido a um custo de US$ 60.000 como uma extensão do *America´s Army*. Antes, os soldados tinham de ser deslocados para um local de treinamento para treinar com um novo equipamento, porém com o TALON em conjunto com o *America´s Army*, os instrutores podem colocá-lo em um laptop e treinar em qualquer lugar. (Ame081)

O interesse dos exércitos em jogos de computador vai além de suas próprias, extremamente especializadas e caras versões desses jogos. Os jogos voltados para

o entretenimento têm se provados úteis para o uso militar também, além desses jogos e as pessoas que os jogam terem se tornado o foco de grande atenção.

Os militares têm descoberto muitos benefícios numa geração de recrutas que cresceu jogando videogames. O mais óbvio desses é melhoria da coordenação olho-mão. Videogames, especialmente aqueles populares entre garotos, quase sempre incorporam um elemento de "*twitch"*, que é a habilidade de detectar uma mudança de estado no jogo e responder rapidamente. De fato, pesquisadores da Universidade de Rochester (Nova Iorque) documentaram que jogadores de jogos "*twitch*" são mais eficientes para processar mudanças rápidas de informação visual (ROCHESTER, 2007). Existem ainda outras vantagens que surgem naturalmente naqueles que jogam videogames:

- Aumento da habilidade para realizar várias tarefas ao mesmo tempo: em jogos orientados para a ação, os jogadores têm de se preocupar com várias coisas simultaneamente: onde estão posicionados no jogo como um todo, onde estão na missão ou nível atual, onde seus companheiros estão, que tipos de recursos estão disponíveis, o número e o tipo de inimigos e obstáculos que eles devem vencer e assim por diante. Permanecer calmo e controlado em situações caóticas é uma habilidade útil para os militares.

- Melhor diferenciação de alvos: uma habilidade chave em situações de combate é ser apto a identificar corretamente aliados e inimigos, de modo a atingir os últimos sem ferir os primeiros. Jogos de tiro em primeira pessoa (FPS – "*first person shooters*") em rede, especialmente, dão aos jogadores muita prática nessa habilidade.

- Priorização de alvos: não é importante somente identificar os alvos inimigos, mas também é necessário identificar o alvo correto (ou mais

correto) que deverá ser atacado antes. O realismo de muitos videogames tem fornecido muitas oportunidades para a prática de priorização de alvos. E, ao contrário da vida real, num videogame é possível praticar repetidamente esta técnica.

- Habilidade para trabalhar com um time utilizando o mínimo de comunicação: quando em missão, é imperativo que os membros de um time se comuniquem rapidamente e de modo claro. Em jogos com vários jogadores, o número de opções de comunicação é normalmente limitado e os jogadores têm de desenvolver habilidades para superar essas limitações. Eles aprendem o seu papel no time e antecipam onde eles devem estar e onde são mais necessários. Eles também adquirem experiência em saber quais informações são úteis para serem passadas adiante, de modo a auxiliar seus companheiros de time.

- Desensiblização para atirar em alvos humanos: um efeito colateral de crescer com videogames que mostram cenas altamente realistas de combate, é que os novos soldados são mais insensíveis para atirar em alvos humanos do que os soldados das gerações anteriores.

- Pré-disposição para tomar ações agressivas: O ciclo de morrer, recarregar, tentar novamente dos videogames, tem dado aos jogadores um espírito agressivo, do tipo “vamos com tudo e vejamos o que acontece”. Embora o sentimento de invencibilidade possa não resistir ao primeiro encontro com munição real, esta prática de tomar a iniciativa pode ser de valor em situações de combate.

Se um soldado puder treinar habilidades necessárias durante o seu tempo de folga, será o ideal. De acordo com especialistas militares em sistemas de treinamento, soldados com conhecimentos de videogames têm se saído muito bem

na realização de tarefas militares semelhantes aos jogos, tais como a operação de veículos aéreos não tripulados em missões de longo alcance. No futuro, veículos aéreos não tripulados poderão ser empregados para espionar o espaço aéreo inimigo e interferir com seus sistemas de defesa aéreos. Recentemente a Agência de Pesquisa Avançada de Projetos de Defesa dos EUA (DARPA – *Defense Advanced Research Projects Agency*) concedeu a Boeing um contrato de US$ 766,7 milhões para continuar trabalhando nesses veículos aéreos.

Os exércitos também têm mostrado interesse em jogos comerciais disponíveis para o público. Em alguns casos, o jogo pode ser usado para treinamento e, em outros, o custo de modificá-lo para atender os requisitos militares é muito mais aceitável do que desenvolver um produto inteiramente novo. Michael Robel, em seu artigo “A Diferença entre Jogos de Guerra Civis e Militares” (*The Diference Between Military and Civilian Wargames*) (ROBEL, 2004) lista alguns desses jogos que são utilizados por forças armadas:

- ***TacOps***, (FIG 1.6) publicado pela Battlefront.com, é, de acordo com sua própria propaganda, “...uma simulação de combates táticos terrestres, contemporâneos ou de um futuro próximo, entre forças dos Estados Unidos (Exército ou Fuzileiros), Canadenses, Neo-Zelandesas / Australianas, e Alemãs contra várias forças oponentes (FOROP), simulando a ex-União Soviética, China, Coréia do Norte, etc...”

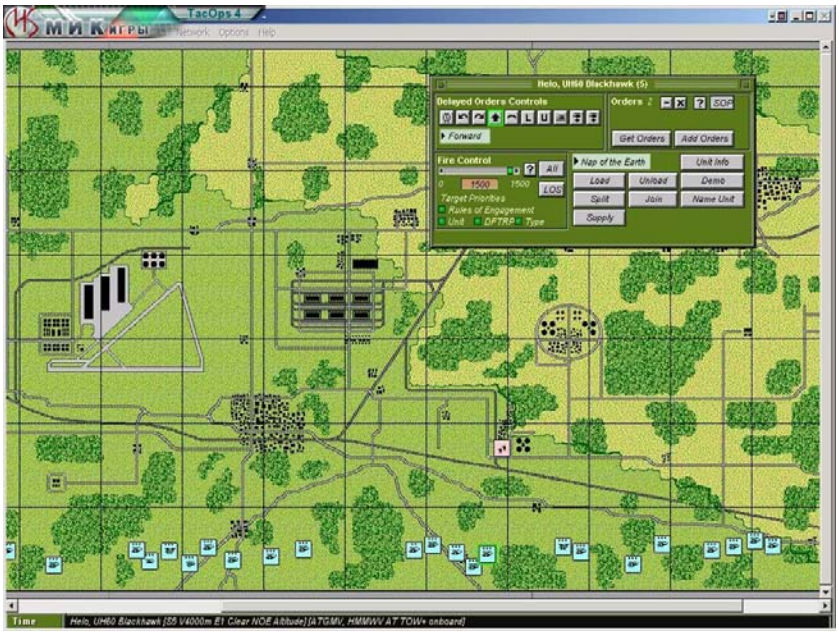


FIG.1.6 Tela do jogo TacOps

- ***Brigade Combat Team***, publicado pela Shrapnel Games, é descrito como "... um jogo de estratégia em tempo real que examina as complexidades do combate no campo de batalha moderno".

- ***Decisive Action***, publicado pela HPS Simulations, é "... uma simulação moderna a nível de Divisão e Corpo que representa combates com manobras de brigadas e batalhões com apoio de artilharia, ataques aéreos, guerra eletrônica, engenharia e helicópteros.

- ***Harpoon 3***, publicado pela Advanced Gaming Systems, é "... jogo de guerra naval em tempo real a nível operacional / tático. Ele modela e simula com grande precisão o combate aéreo e naval. Inclui a maior parte dos aspectos do combate aéreo e naval incluindo plataformas editáveis, sensores e armas." *Harpoon* foi originalmente um jogo de miniaturas de mesmo nome, publicado em 1979 pela Adventure Games.

De acordo com a página na Internet da Advanced Gaming Systems, o Departamento de Defesa Australiano financiou o desenvolvimento de duas variantes do *Harpoon 3.* As primeiras versões foram utilizadas pela Escola de Guerra Naval da Marinha do Brasil, pelo Comando da Força Aérea dos Estados Unidos, e (não oficialmente) por Holanda, Grécia, Turquia, Reino Unido e Coréia.

Outros jogos comerciais como o jogo Warcraft, da Blizzard Entertainment, têm sido modificados para o uso militar. No passado, o Corpo de Fuzileiros dos EUA utilizou uma versão modificada do jogo *Doom* (FIG 1.7) e uma versão especial do jogo *Close Combat* para treinamento. De modo similar, a Força Aérea Norte Americana tem utilizado o jogo *Microsoft Flight Simulator*. (LENOIR, 2003) (LENOIR, et al., 2005)

FIG.1.7 Tela do jogo Doom

Para atender suas necessidades atuais, a DARPA está financiando o *DARWARS*, iniciado em 2003, para aperfeiçoar o treinamento militar. Alguns dos sistemas desenvolvidos já estão sendo usados em campo, tais como o *DARWARS Ambush!* O jogo *DARWARS Ambush!*, que é uma versão modificada do jogo *Operation Flashpoint*, se concentra no treinamento de sobrevivência de comboios,

uma necessidade urgente para o Exército Norte Americano em suas contínuas missões no Iraque e Afeganistão. (DARPA, 2003)

1.4 TREINAMENTO NÃO-COMBATIVO

Nem todo treinamento militar por meio de jogos é voltado para combates. Os soldados Norte Americanos, por exemplo, quando levados para fora dos EUA muitas vezes se deparam com culturas muitos diferentes e não estão aptos a falar a língua local. Várias empresas e universidades estão se esforçando para resolver estes dois problemas.

Em 2004, pesquisadores do Instituo de Ciências da Universidade do Sul da Califórnia começaram a trabalhar no jogo *Tactical Iraqi*, um esforço para ensinar árabe aos soldados norte americanos (a maioria só conhece uma única língua, o inglês) antes que eles embarcassem. Estes tipos de jogos envolvem trabalho com tecnologia de reconhecimento de voz, pois falar uma língua é essencial ao seu aprendizado. Um facilitador humano monitora e corrige os alunos, pois esta tecnologia ainda é relativamente nova. (Tac08)

Seguindo um caminho similar, há o *VECTOR*. Com a natureza dos combates mudando de guerras de agressão para missões de paz e embates com insurgentes e guerrilhas, os soldados precisam entender melhor e interagir com as culturas estrangeiras. O *VECTOR* vai além de simplesmente ensinar a língua local e usa uma modelagem cognitiva e emocional para fornecer treinamento cultural para os soldados, ao mesmo tempo em que tenta ser também um jogo de engajamento e entretenimento. Os jogadores fazem com que seus avatares realizem no jogo gestos apropriados de modo a evitar ofender membros simulados da população nativa. Novamente, facilitadores humanos são necessários para ajudar a colocar o que for aprendido dentro do contexto cultural próprio do jogador, bem como verificar se os jogadores estão realizando os gestos corretos. Assim, embora estes jogos auxiliem

no processo de treinamento, eles ainda não substituem completamente os métodos mais tradicionais.

A maior parte dos militares não esta envolvida diretamente na linha de frente de um combate. Os modernos combatentes são apoiados por um grupo de analistas, motoristas, cozinheiros, etc. que estão realizando suas tarefas normais sob condições extremamente adversas. Os comandantes estão cientes de que também precisam de treinamento para seu pessoal não-combatente. Durante a Guerra do Iraque, para ilustrar, as tropas não combatentes sofreram mais perdas que as combatentes. Jogos podem ser utilizados para treinar esse pessoal também. (CAFFREY, 2000)

Para médicos de campanha, trabalhar bem sob a pressão de um combate é muito importante. Salas de emergência podem prover condições de pressão semelhantes, mas para muitos, servir sob condições de guerra será provavelmente muito diferente do que qualquer situação já enfrentada antes. A incerteza sobre sua segurança combinada com ruídos altos e situações precárias pode ser desconcertante para aqueles que precisam agir rápida e precisamente para salvar vidas. Uma simulação fornece o melhor cenário que não um combate real para o treinamento de médicos. No jogo *America´s Army*, os jogadores podem receber aulas de treinamento médico virtual e assumir o papel de médicos de combate.

À medida que os jogos e as simulações de treinamento se tornam mais amplamente conhecidas e mais utilizadas, novas possibilidades estão sendo investigadas e exploradas, tais como: desarmamento de bombas, treinamento médico, treinamento de escolta de comboios entre outros.

1.5 O FUTURO DOS JOGOS DE GUERRA

Em 1999, o Exército Norte Americano juntamente com a Universidade do Sul da Califórnia criou o Instituto para Tecnologias Criativas (ICT – *Institute for Creative Technologies*), reunindo educadores, desenvolvedores de jogos e outras companhias de entretenimento para criar a próxima geração de ferramentas de treinamento militar e simulações. O JFETS (*Joint Fires and Effects Trainer System*) é um dos projetos resultantes do ICT. No JFETS, o local da missão, com pessoas e defesas simuladas é apresentado ao jogador. Desde que as missões são realizadas por times, o treinamento se torna um jogo multi-jogador. Superiores podem monitorar o desempenho dos indivíduos, bem como de times inteiros, e assim fornecer respostas, tanto negativas quanto positivas, em avaliações pós-ação. Se o método de simulação for suficientemente estimulante, é bem possível que os soldados tenham vontade de jogar em suas horas livres, combinando entretenimento não supervisionado com treinamento. (LENOIR, et al., 2005)

Notando a utilidade das simulações multi-jogador, os exércitos têm visualizado o potencial dos jogos massivos em rede para múltiplos jogadores (MMOGs – *Massive Multiplayer Online Games*). Robert Gehorsam, presidente da Forterra Systems se aproximou do Exército Norte Americano em 2002 com a idéia de utilizar a tecnologia do jogo *There* para simular condições de guerra contra insurgentes em ambientes urbanos. *There* é um MMOG em que há particular atenção ao realismo, especialmente no que diz respeito aos avatares. O realismo dos mundos virtuais torna os MMOGs ideais para tratar de situações de combate urbano, tais como ocupações e engajamentos com insurgentes. Em outubro de 2004, os oficiais do Estado Maior Conjunto dos EUA testaram suas possibilidades conduzindo a maior simulação em tempo real de um combate urbano da história, com jogadores em três localidades diferentes controlando mais de cem mil entidades. (GONZALEZ, 2004)

Examinando as possibilidades dos MMOGs, Jason Robar (ROBAR, 2004), do AISA Group, escreveu “está claro que uma tecnologia que pode hospedar 600.000

jogadores simultâneos em um ambiente de competição entre guildas e clãs, sendo cada um uma organização político-militar, possui alguma aplicação militar.” Os MMOGs, de acordo com ele, “oferecem novas possibilidades que podem auxiliar e melhorar a maneira como os combatentes e as comunidades de inteligência se preparam e treinam para ... o século 21.”

Operações de treinamento real, colocando centenas ou mesmo milhares de pessoas em campo, tem sido há séculos uma necessidade de treinamento militar. O custo de tais operações, porém, em termos de homens e equipamento tem sido proibitivo. Com a tecnologia dos MMOGs reunindo tropas de todo o mundo, tais operações podem ser feitas a um custo muito menor e com muito mais sigilo.

Além da tecnologia dos MMOGs, os exércitos estão investindo nos sistemas de treinamento de realidade virtual. O Exército Norte Americano possui um programa de US$ 6 milhões para desenvolvimento de tecnologia de treinamento virtual baseado na experiência dos atuais veteranos de guerras. O programa usa tecnologia do tipo realidade virtual, ambientes com múltiplos jogadores e até mesmo manequins computadorizados para auxiliar no treinamento de pessoal médico. Ao incorporar a experiência de combatentes veteranos, os novatos podem receber simulações muito mais acuradas.

1.6 MOTIVAÇÃO

As forças armadas de vários países têm empregado efetivamente sistemas computacionais para o treinamento de suas tropas e a simulação de combates históricos ou fictícios

Um caso de particular destaque é o Exército Norte Americano, que conta com simuladores militares para aplicação em diversas áreas. Na área da educação pode-se citar como exemplos, o simulador de carros de combate *Steel Beasts,* empregado

pela Academia Militar de West Point, e o simulador tático operacional *The Operational Art of War*, empregado pela *School of Advanced Military Studies (SAMS)*. Já na área de treinamento da tropa, destacam-se sistemas como: o *TACOPS*, para treinamento estratégico; o simulador de carros de combate *Spearhead II*; a série de sistemas *Full Spectrum* (FIG 1.8), que possui variantes para treino de comando de companhias, pelotões e grupos de combate; e a série *Virtual Battle Space* 1 e 2, empregados para o treinamento individual de soldados em vários tipos de operações.

FIG.1.8 Tela do jogo Full Spectrum Warrior

O Exército Brasileiro tem desenvolvido e empregado alguns sistemas que podem ser classificados como jogos de guerra, porém encontra-se ainda numa fase inicial em termos de pesquisa e desenvolvimento deste tipo de sistema. Os sistemas atualmente em utilização são:

- O programa AZUVER, empregado no Exercício de Simulação de Combate de mesmo nome, realizado simultaneamente nas Escolas de

Comando e Estado Maior do Exército (ECEME), da Aeronáutica (ECEMAR) e Escola de Guerra Naval (EGN). Trata-se de um simulador de caráter estratégico-operacional, pois é voltado para o treinamento de estados-maiores nos diversos escalões da Força Terrestre.

- O programa SISTAB (Sistema de Simulação Tático de Brigada – FIG 1.9), empregado em Exercícios de Simulação de Combate aplicados pelos CAESC (Centros de Aplicação de Exercícios de Simulação de Combate) dos Comandos Militares de Área. Permite o treinamento de comandantes de brigada e batalhão em simulações de operações de combate convencional, possuindo, portanto, tanto um caráter estratégico-operacional quanto tático.

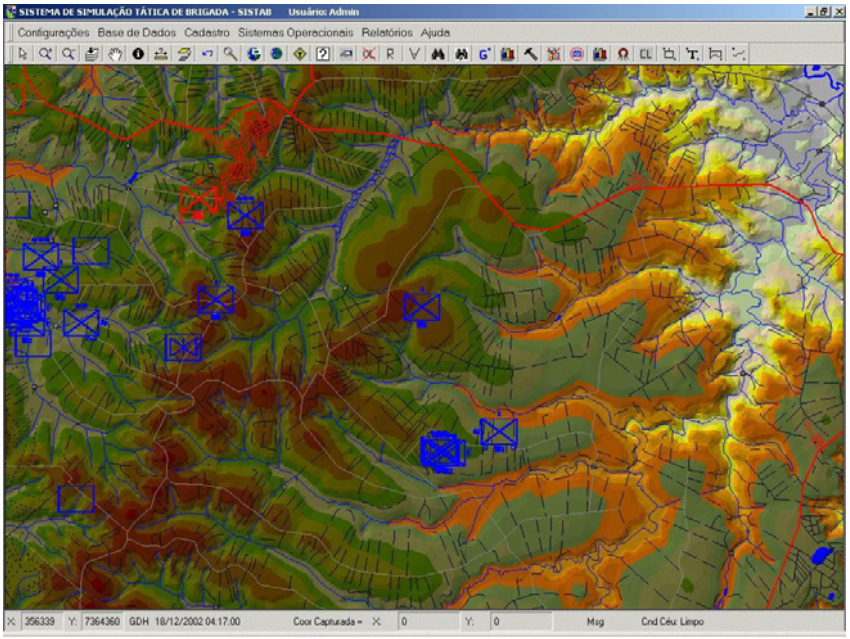


FIG.1.9 Tela do simulador SISTAB

O programa SABRE (Sistema de Adestramento de Batalhões e Regimentos do Exército – FIG 1.10), empregado em Exercícios de Simulação de Combate em diversas unidades militares. É utilizado para o treinamento de comandantes e

subcomandantes de batalhões e regimentos. Assim como o SISTAB, embora simule detalhadamente condições da tropa (armamento, munição, combustível, fadiga, etc...) ainda trata-se de um sistema que enfoca os resultados das decisões do comandante, sem levar em conta as ações de cada indivíduo que compõe a tropa. (RIBEIRO, 2005)

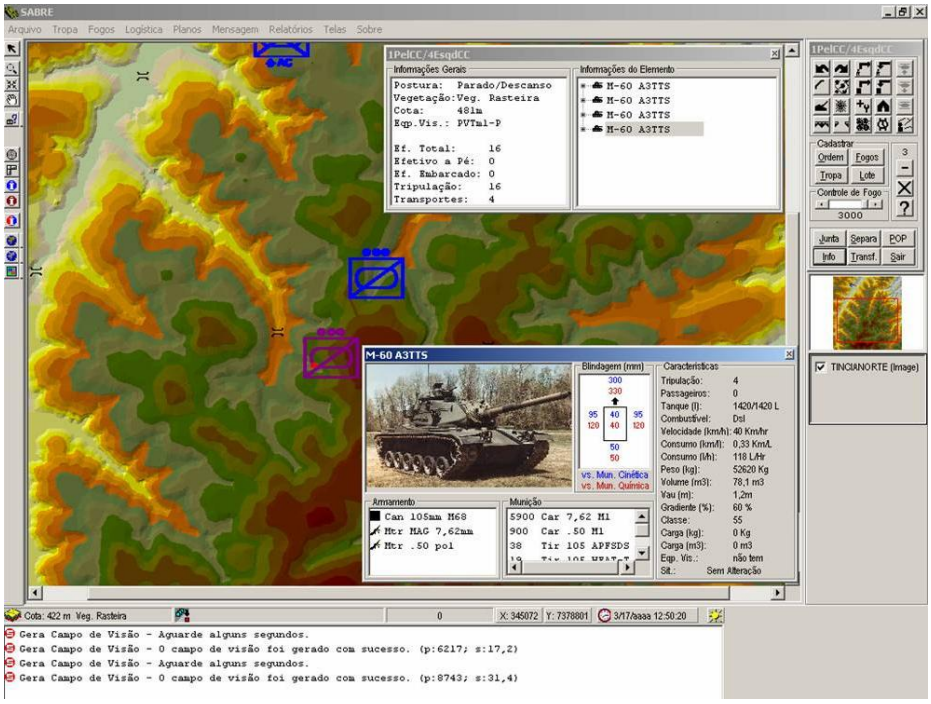


FIG.1.10 Tela do simulador SABRE

Por esse quadro pode-se constatar que não existe um sistema outorgado pelo Comando de Operações Terrestres do Exército Brasileiro, que se destine ao treinamento de caráter predominantemente tático. Ou seja, há a carência de um sistema computacional de treinamento, focalizado na ação individual ou de pequenos grupos.

1.7 A PROPOSTA DESTE TRABALHO

Este trabalho se propõe a descrever uma arquitetura de software que seja a base para o desenvolvimento de sistemas voltados para o treinamento de pequenas frações das unidades militares, como pelotões, grupos de combate e até mesmo soldados individualmente.

Esse tipo de sistema de simulação tático seria inicialmente de grande aplicabilidade nas Escolas destinadas à formação de oficiais e praças, que desfrutariam assim de um ambiente virtual que permitiria uma maior gama de experiências de combates simulados. Poderia ser futuramente estendido ao treinamento de soldados em unidades operacionais e de pronto-emprego, que precisam conciliar a demanda por constante treinamento com restrições de gastos com exercícios reais. Esse sistema poderia ainda ser empregado em outras Forças, como o Corpo de Fuzileiros Navais e a Infantaria da Aeronáutica.

Além dessas aplicações militares, as Forças Auxiliares e as Empresas de Segurança também poderiam desfrutar dos benefícios desse sistema de treinamento individualizado. Em resumo, trata-se de um objeto de pesquisa e desenvolvimento que pode contribuir para as mais diversas entidades de defesa e segurança.

2 A ENGENHARIA DE SOFTWARE

Projetos orientados a objetos provêem suporte ao desenvolvimento particionado de sistemas, possibilitam modelagens baseadas em objetos reais, e ajudam a assegurar um bom nível de qualidade e gerenciamento de produtos e do processo de desenvolvimento (MEYER, 1997). Este capítulo apresenta uma discussão sobre os princípios de projetos orientados a objetos que compõem o estado da arte da engenharia de software e são extensamente utilizados pela indústria software em geral. O objetivo deste capítulo é dar o embasamento necessário ao entendimento da arquitetura proposta neste trabalho.

2.1 PRINCÍPIOS BÁSICOS DA ENGENHARIA DE SOFTWARE

Projetos orientados a objetos têm apresentado sucesso na indústria de software em geral devido à melhor qualidade oferecida as aplicações e por permitir a construção de grandes sistemas melhor gerenciáveis.

Uma razão para o sucesso do paradigma orientado a objetos é que ele é bem mais voltado para a maneira como pensamos. As funcionalidades de um sistema podem ser corretamente particionadas em componentes e então empacotadas, isoladas de outros subsistemas de um projeto, o que é difícil conseguir com a programação procedural.

Os fatores de qualidade implicam na condução de propostas para a construção de arquiteturas flexíveis, produzidas por componentes de software autônomos e reusáveis (módulos). O uso de componentes têm-se apresentado como uma excelente técnica de desenvolvimento. Ao invés de se escrever um único e abrangente componente de software (desenvolvimento monolítico), escreve-se um conjunto de pequenos componentes que se comunicam entre si. Esta decomposição caracteriza um bom grau de independência que pode ser aplicado a estes

componentes, de maneira que eles possam ser utilizados em diferentes projetos sem a necessidade de ajustes, além da possibilidade de realizar modificações e testes também de forma independente.

Estas técnicas, aplicadas de forma ideal, provêem componentes que podem ser facilmente conectados para construir um novo sistema. Não é nada interessante que todo novo desenvolvimento deva partir do início. Deveria existir um catálogo de componentes de software, como existem catálogos de dispositivos de hardware, possibilitando a reutilização destes componentes, sem a necessidade de "reinvenção da roda". Conseqüentemente, diminuindo a quantidade de código a ser construído, e provavelmente gerando um resultado muito melhor.

A seguir são apresentados os principais conceitos e técnicas, da engenharia de software, relacionados ao desenvolvimento de projetos orientados a objetos, os quais podem ser largamente utilizados na construção de jogos de computador.

2.1.1 MODULARIDADE

No desenvolvimento de um sistema de software, módulos são pacotes organizados que apresentam autonomia, coerência, e robustez (SOMMERVILLE, 2000). Eles devem ser projetados para serem extensíveis e portáveis, de modo que possam ser plugados em outros sistemas.

Existem cinco critérios que ajudam a caracterizar a modularidade (FLYNT, et al., 2005), são eles:

- Decomposibilidade: o problema deve ser decomposto em diversos subproblemas de maneira que sua solução possa ser encontrada separadamente;

- Composibilidade: o sistema suporta a produção de elementos de software que podem ser livremente combinados para produzir um novo sistema;

- Compreensibilidade: o sistema produz módulos que podem ser compreendidos separadamente, ou juntamente com uma pequena quantidade de outros módulos;

- Continuidade: uma modificação necessária na especificação de um problema resulta em alteração de apenas um (ou poucos) módulo(s);

- Proteção: o sistema apresenta uma arquitetura em que o efeito de condições anormais num determinado módulo, em tempo de execução, permanece restrito a ele próprio.

Estes cinco critérios apontam para cinco princípios que devem ser seguidos para atingir a modularidade (FLYNT, et al., 2005). A relação entre eles é apresentada entre parênteses.

- Unidade lingüística modular: módulos devem corresponder à unidades sintáticas na linguagem utilizada (decomposibilidade, composibilidade, proteção);

- Poucas interfaces: todo módulo deve se comunicar com uma menor quantidade possível de outros módulos (continuidade, proteção);

- Pequenas interfaces: se dois módulos devem se comunicar, eles devem trocar um mínimo de informações possíveis (continuidade, proteção);

- Interfaces explícitas: sempre que dois módulos se comunicarem, eles devem se acoplar diretamente, ou seja, as informações trocadas devem ser restritas aos seus dados (decomposibilidade, composibilidade, continuidade, compreensibilidade);

- Ocultar informações: toda a informação sobre um módulo deve ser privada, a menos que seja extremamente necessário declará-la como pública (continuidade, não necessariamente proteção).

Módulos possuem duas maneiras de formatação, eles podem ser abertos ou fechados. Um módulo aberto é caracterizado por ser disponibilizado para extensão. É possível adicionar novos campos a sua estrutura de dados ou adicionar novas funções para operar com estas estruturas. Um módulo fechado é caracterizado por ser disponibilizado apenas para ser usado por outros módulos. Ele deve apresentar uma interface estável e bem definida.

2.1.2 REUSABILIDADE EM SOFTWARE

Reusabilidade é um problema básico de engenharia de software. Por que gastar tempo desenvolvendo algo quando este já existe? No entanto, esta questão não tem uma simples resposta. Vários fatores estão envolvidos neste problema. Bibliotecas que apresentem um bom conjunto de características necessárias para o desenvolvimento de uma determinada aplicação podem ser encontradas, mas podem necessitar de licenciamento, adaptações, ou apresentarem problemas que não serão solucionados no intervalo de tempo requerido. Neste caso, quando componentes tiverem de ser desenvolvidos localmente, devem ser projetados de forma que o seu reuso seja maximizado. As duas principais formas de reusabilidade no desenvolvimento de software são reuso de código e reuso de projeto (PRESSMAN, 2005).

O reuso de código é apresentado pela produção de um conjunto de componentes e ferramentas que possam ser aproveitados por uma série de produtos. Desde o início do desenvolvimento, estes componentes e ferramentas devem ser projetadas com a possibilidade de expansões futuras. Devido à necessidade de expansões, um conjunto de componentes (módulos) deve ser construído de forma que possa ser liberado de forma incremental, ou seja, deve permitir o funcionamento parcial de um sistema. Abaixo são apresentados alguns problemas inerentes às estruturas modulares, que devem ser resolvidos para produzir componentes reusáveis.

- Variação de tipos: um módulo deve ser aplicado a estruturas de diferentes tipos;

- Variação em estrutura de dados e algoritmos: como as ações executadas durante um algoritmo podem depender do tipo da estrutura de dados, o módulo deve permitir o direcionamento para variações de diferentes estruturas;

- Rotinas relacionadas: um módulo deve ter acesso às rotinas para manipular as estruturas de dados diferentes;

- Independência de representação: um módulo deve permitir ao usuário especificar uma operação sem saber como ela é implementada ou qual o tipo de estrutura de dados que está sendo utilizada;

- Comunidade com outros subgrupos: devem ser construídas interfaces abstratas que não exponham estruturas de dados, permitindo a sua implementação de diferentes formas, garantindo a não repetição de blocos de código similares.

- Um grande feito da comunidade de desenvolvimento de software, que tem resultado em um dos mais utilizados métodos de reuso, tem sido o reuso de projeto através de padrões, formando um catálogo de soluções para diversos problemas de projeto. Devido ao reuso de projeto ser o foco maior deste trabalho na estruturação da arquitetura para sistemas de simulação de combate, ele é apresentado numa seção à parte.

2.2 REUSO DE PROJETO – PADRÕES DE PROJETO

Padrões de projeto são bastante populares na comunidade de orientação a objetos devido a proverem a reutilização das informações de bons projetos. Um padrão descreve um problema que ocorre diversas vezes no ambiente, e a sua solução. Esta descrição é feita de uma maneira que possa ser utilizada infinitas vezes, sem a necessidade de ser rescrita. As soluções são expressas em termos de objetos e interfaces. Normalmente, um padrão tem quatro elementos essenciais: o seu nome, a descrição do problema em questão, a solução, e as possíveis conseqüências (GAMMA, et al., 1995).

O *nome do padrão* é um identificador que é utilizado para descrever um problema, a sua solução, e quais suas conseqüências. O nome de um padrão incrementa o vocabulário de soluções de problemas em geral. Ele deve ser projetado com alto nível de abstração, fazendo-se interessante para possibilitar diálogos e documentações padronizadas, conseqüentemente aumentando a facilidade para pensar sobre projetos e prover melhor nível de comunicação. Utilizar bons nomes para padrões é necessário para uma atualização bem definida do seu catálogo.

A *descrição do problema* explica quando aplicar o padrão, mostrando a situação e seu contexto. Ela pode descrever cada problema de projeto específico e como representar algoritmos em objetos, e pode descrever estruturas de classes ou

objetos que são sintomáticas de um projeto inflexível. Algumas vezes o problema poderá incluir uma lista de condições que devem ser verificadas antes que o padrão seja aplicado.

A *solução* descreve os elementos que constroem o projeto, seus relacionamentos, responsabilidades, e colaborações. A solução não descreve um projeto concreto ou implementação em particular, porque um padrão é como um "*template*" (parametrizador que realiza operações independentes do tipo do objeto) que pode ser aplicado em muitas diferentes situações. Ela provê uma descrição abstrata de um problema de projeto e sua resolução através de uma conFIGção geral de elementos (classes e objetos).

As *conseqüências* descrevem os resultados da aplicação do padrão. Elas são críticas na disponibilização de alternativas de projeto em relação ao custo e benefício de sua aplicação. As conseqüências para o software freqüentemente se relacionam ao tempo gasto em se obter o resultado. Quando necessário, são descritos os problemas que podem ocorrer com aplicações em determinadas linguagens e implementações. As conseqüências de um padrão incluem o impacto dele na flexibilidade, extensibilidade ou portabilidade de um sistema.

O ponto de vista dos desenvolvedores pode afetar a interpretação do que é ou não é um padrão. Um padrão pessoal pode ser um bloco qualquer de construção primitiva. Porém, padrões de projeto têm um certo nível de abstração. Eles não são projetados como listas ligadas e tabelas *hash* que podem ser codificadas em classes e reutilizadas. Eles não são complexos, projetos de domínio específico para uma aplicação, ou subsistemas. Padrões de projeto são descrições de comunicação de objetos e classes que são personalizadas para resolver um problema de projeto geral num contexto particular. Eles geram sistemas reusáveis, flexíveis, extensíveis, e portáveis.

Embora padrões apenas descrevam projetos, em orientação a objetos, eles são baseados em soluções práticas que têm sido extensamente implementadas nas principais linguagens de programação orientadas a objetos. A escolha da linguagem de programação é importante porque ela influencia no ponto de vista, e determina o que pode e não pode ser implementado facilmente. Atualmente, Smalltalk, C++ e Java são linguagens características para a utilização de padrões de projeto.

Padrões de projeto tornam mais fácil o sucesso do reuso de projetos e arquiteturas, conseqüentemente, tornando mais fácil o desenvolvimento de novos sistemas. Neste contexto, se inserem as arquiteturas, que são grandes consumidores de padrões. Além das arquiteturas poderem utilizar padrões, elas também podem utilizar a linguagem de padrões para a sua documentação. A documentação de uma arquitetura deve apresentar três diretrizes as quais a linguagem de padrões ajuda a descrever: a descrição da sua proposta, como desenvolvedores de aplicação podem usá-la, e os seus detalhes de projeto (estruturação). A linguagem de padrões não é uma linguagem formal, mas sim uma coleção de padrões inter-relacionados, embora eles apresentem um vocabulário que fala sobre um problema particular. Ela provê um processo para resolução ordenada de problemas de desenvolvimento de software. Ambos, padrões e a linguagem de padrões, ajudam na documentação e gerenciamento de sistemas por fornecer uma especificação explícita de interações de classes e objetos. Além disso, ajudam os desenvolvedores na comunicação do conhecimento e na aprendizagem de um novo paradigma de projeto ou estilo arquitetural, e, por fim, ajudam novos desenvolvedores a fugir das armadilhas que têm tradicionalmente sido conhecidas somente com o alto custo da experiência.

2.3 CONCLUSÃO DO CAPÍTULO

Neste capítulo foram analisadas técnicas empregadas no desenvolvimento de sistemas de software em geral, especialmente técnicas consagradas na indústria de software, que auxiliam os desenvolvedores a obter produtos padronizados, de

melhor qualidade e de mais fácil manutenção, além de acelerar o desenvolvimento de sistemas.

Sendo assim, fica claro que estas técnicas de engenharia de software devem estar presentes também numa arquitetura de software que se proponha a ser um modelo para jogos de treinamento militar. No capítulo seguinte será demonstrada a aplicação prática destas técnicas na elaboração de uma arquitetura de software, mais especificamente uma arquitetura para jogos.

3 ARQUITETURA DE SOFTWARE

Apesar do grande interesse em arquitetura de software como área de pesquisa, existe pouca concordância entre os pesquisadores sobre o que deve ser incluído na definição de arquitetura. Embora o termo arquitetura de software seja relativamente novo na indústria, os princípios fundamentais deste campo vêm sendo aplicados esporadicamente pela engenharia de software desde o inicio dos anos 80. As primeiras tentativas de capturar e explicar a arquitetura de software foram imprecisas e desorganizadas – freqüentemente caracterizadas por um conjunto de diagramas. Neste capítulo será apresentado um conjunto de conceitos coerentes para arquitetura de software, com base no estudo de fontes bibliográficas diversas, com o objetivo de elucidar a terminologia empregada na proposta deste trabalho.

3.1 DEFINIÇÃO

De acordo com os autores Len Bass e Paul Clements (BASS, et al., 2003), a arquitetura de software de um programa ou sistema de computação pode ser definida como a estrutura ou estruturas do sistema, composta por *elementos de software* (*componentes, conectores ou dados*), as *propriedades externamente visíveis* destes elementos e os *relacionamentos* entre eles.

Propriedades externamente visíveis são as suposições que outros elementos podem fazer sobre um elemento, tais como os serviços que ele provê, suas características de desempenho, manipulação de falhas, uso de recursos compartilhados e assim por diante.

Em primeiro lugar, uma arquitetura define *elementos de software.* A arquitetura guarda informações sobre como os elementos se relacionam uns com os outros. Isto significa que ela propositalmente omite as informações sobre os elementos que não pertencem às suas interações. Assim, uma arquitetura é acima de tudo uma

abstração de um sistema que suprime detalhes dos elementos que não afetam como eles usam, são usados, se relacionam ou interagem com outros elementos. Na maior parte dos sistemas modernos, os elementos interagem uns com os outros por meio de interfaces que dividem seus detalhes sobre um elemento em partes públicas e privadas. A arquitetura contém apenas a parte pública desta divisão; detalhes privados – aqueles relacionados exclusivamente com a implementação interna – não são arquiteturais.

Em segundo lugar, de acordo com a definição, está claro que sistemas podem ser compostos por mais de uma estrutura e que nenhuma estrutura pode isoladamente ser considerada uma arquitetura. Por exemplo, todos os projetos não-triviais são divididos em unidades de implementação; a estas unidades são atribuídas responsabilidades específicas e são freqüentemente a base da delegação de trabalhos para equipes de programação. Este tipo de elemento é composto por programas e dados que softwares em outras unidades de implementação podem chamar ou acessar, e programas e dados que são privados. Em grandes projetos, estes elementos são quase que certamente subdivididos para sub-equipes de desenvolvimento. Este é normalmente o tipo de estrutura usada para descrever um sistema. É extremamente estático, pois se concentra na divisão de funcionalidades e sua designação para equipes de desenvolvimento.

Outras estruturas são muito mais concentradas na maneira como os elementos interagem uns com os outros em tempo de execução para realizar a função do sistema. Teoricamente o sistema é construído como um conjunto de processos paralelos. Os processos que existirão em tempo de execução, os programas nas várias unidades de implementação descritas anteriormente que estão conectados em série para formar cada processo, e as relações de sincronização entre os processos formam outro tipo de estrutura normalmente utilizado para descrever um sistema.

3.1.1 COMPONENTES

Um componente é uma unidade abstrata de instruções de software e estados internos que provê uma transformação de dados através de sua interface (PERRY, et al., 1992). Exemplos de transformações incluem o carregamento em memória, a partir do armazenamento em disco, a realização de algum cálculo, a tradução para um diferente formato, o encapsulamento com outro dado, etc. O comportamento de cada componente é parte da arquitetura enquanto esse comportamento possa ser observado ou discernido do ponto de vista de outro componente. Em outras palavras, um componente é definido pela sua interface e serviços que ele provê para outros componentes, ao invés da implementação por trás da sua interface. Isto é definido então como o conjunto de suposições que outras unidades arquiteturais podem fazer sobre o componente.

3.1.2 CONECTORES

Um conector é um mecanismo abstrato que faz a mediação da comunicação, coordenação e cooperação entre os componentes (PERRY, et al., 1992). Exemplos incluem representações compartilhadas, chamadas de procedimentos remotos, protocolos de passagem de mensagens e fluxos de dados.

Talvez a melhor maneira de pensar sobre os conectores é contrastá-los com os componentes. Conectores possibilitam a comunicação entre componentes através da transferência de elementos de dados de uma interface para a outra sem alterar os dados. Internamente um conector pode consistir de um subsistema de componentes que transformam os dados para transferência, realizam a transferência e então fazem a transformação inversa para a entrega. No entanto, a abstração comportamental externa capturada pela arquitetura ignora esses detalhes. Em contraste, um componente pode, mas nem sempre o faz, transformações de dados sob uma perspectiva externa.

3.1.3 DADOS

Um dado é um elemento de informação que é transferido de um componente, ou recebido por um componente, através de um conector (PERRY, et al., 1992). Exemplos incluem seqüências de bytes, mensagens, parâmetros de controle, e objetos serializados, mas não incluem informações que são permanentemente residentes ou estão ocultas junto de um componente. De uma perspectiva arquitetural, um “arquivo” é uma transformação que um componente de sistema de arquivos pode obter a partir de um dado do tipo “nome de arquivo” recebido em sua interface como uma seqüência de bytes gravada junto de um sistema de armazenamento interno oculto. Componentes podem gerar dados, como no caso do encapsulamento via software de um relógio ou sensor.

3.2 CONFIGURAÇÕES

Uma configuração é a estrutura de relacionamentos arquiteturais entre componentes, conectores e dados durante um período de execução do sistema (ABOWD, et al., 1995). Estritamente falando, uma configuração pode ser vista como o equivalente a um conjunto de restrições específicas para a interação dos componentes.

3.3 PROPRIEDADES

O conjunto de propriedades arquiteturais de uma arquitetura de software inclui todas as propriedades que derivam da seleção e arranjo de componentes, conectores e dados do sistema (BASS, et al., 2003). Exemplos incluem tanto as propriedades funcionais obtidas por um sistema como as propriedades não funcionais, tais como a facilidade de evolução, reusabilidade de componentes, eficiência e extensibilidade dinâmica, normalmente referidos como atributos de qualidade.

Propriedades são induzidas por um conjunto de restrições relativas à arquitetura. Restrições são normalmente motivadas pela aplicação de um princípio de engenharia de software a um aspecto dos elementos arquiteturais. Por exemplo, o estilo “canal e filtro uniformes” obtêm as qualidades de reusabilidade de componentes e configurabilidade da aplicação aplicando através do uso de generalidade nas interfaces de seus componentes, restringindo os componentes a um único tipo de interface. Assim, a restrição arquitetural é “interface de componente uniforme”, motivado pelo princípio da generalidade, de modo a se obter duas qualidades desejáveis que se tornarão propriedades arquiteturais dos componentes reusáveis e configuráveis, quando esse estilo for empregado em uma arquitetura.

O objetivo do projeto arquitetural é criar uma arquitetura com um conjunto de propriedades arquiteturais que formem um superconjunto dos requisitos do sistema. A importância relativa de várias propriedades arquiteturais depende da natureza do sistema desejado.

3.4 ESTILOS ARQUITETURAIS

Um estilo arquitetural é um conjunto coordenado de restrições que definem os papéis e características dos elementos arquiteturais e os relacionamentos permitidos entre esses elementos, em qualquer arquitetura que obedeça a esse estilo (BASS, et al., 2003).

Como uma arquitetura engloba tanto propriedades funcionais e não-funcionais, pode ser difícil comparar arquiteturas para diferentes tipos de sistemas ou mesmo para o mesmo tipo de sistema em diferentes ambientes. Estilos são um mecanismo para categorizar arquiteturas e para definir suas características comuns. Cada estilo provê uma abstração para as interações dos componentes, capturando a essência de um padrão de interação ignorando detalhes do restante da arquitetura.

Novas arquiteturas podem ser definidas como instâncias de estilos específicos. Como os estilos arquiteturais podem definir diferentes aspectos da arquitetura de software, uma dada arquitetura pode ser composta por múltiplos estilos. Por outro lado, um estilo híbrido pode ser formado pela combinação de múltiplos estilos básicos em um único estilo coordenado.

A seguir serão descritos alguns estilos arquiteturais básicos mais comuns, suas vantagens e desvantagens. Porém é necessário salientar que existem muitos outros estilos possíveis, simples ou compostos, porém não tão relevantes para este trabalho.

3.4.1 ARQUITETURA EM CAMADAS

O estilo de arquitetura em camadas (FIG 3.1) (PRESSMAN, 2005) divide as funcionalidades do sistema em camadas hierárquicas onde cada camada provê serviços para as camadas acima e abaixo dela. A estratégia em camadas tende a promover o reuso mantendo o código específico da aplicação nas camadas mais elevadas, permitindo aos desenvolvedores o reuso da estrutura abaixo. O reuso está diretamente relacionado aos requisitos de flexibilidade e modificabilidade, que são importantes para a arquitetura proposta neste trabalho.

3.4.2 ARQUITETURA CENTRADA EM DADOS

O estilo centrado em dados (FIG 3.2) (PRESSMAN, 2005) é essencialmente um repositório de dados centralizado com clientes independentes que se conectam para operar sobre os dados. Estilos centrados em dados oferecem uma maneira simples de integrar diferentes sistemas porque os clientes são independentes uns dos outros, e o repositório de dados é independente dos clientes.

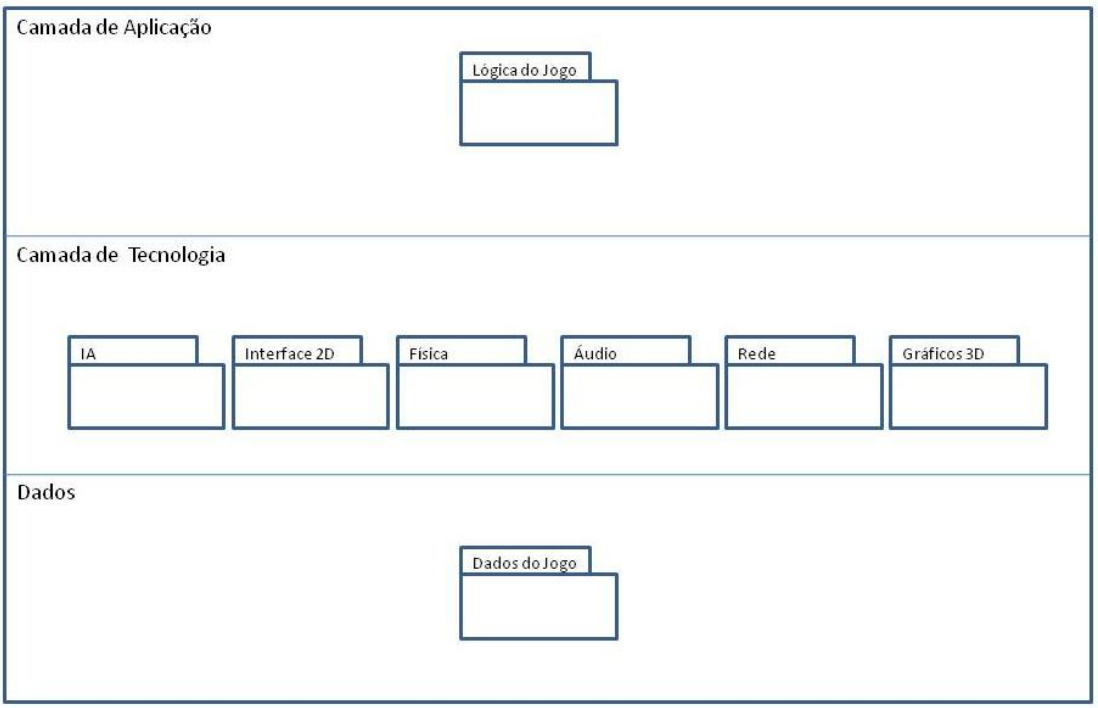


FIG.3.1 Arquitetura em Camadas

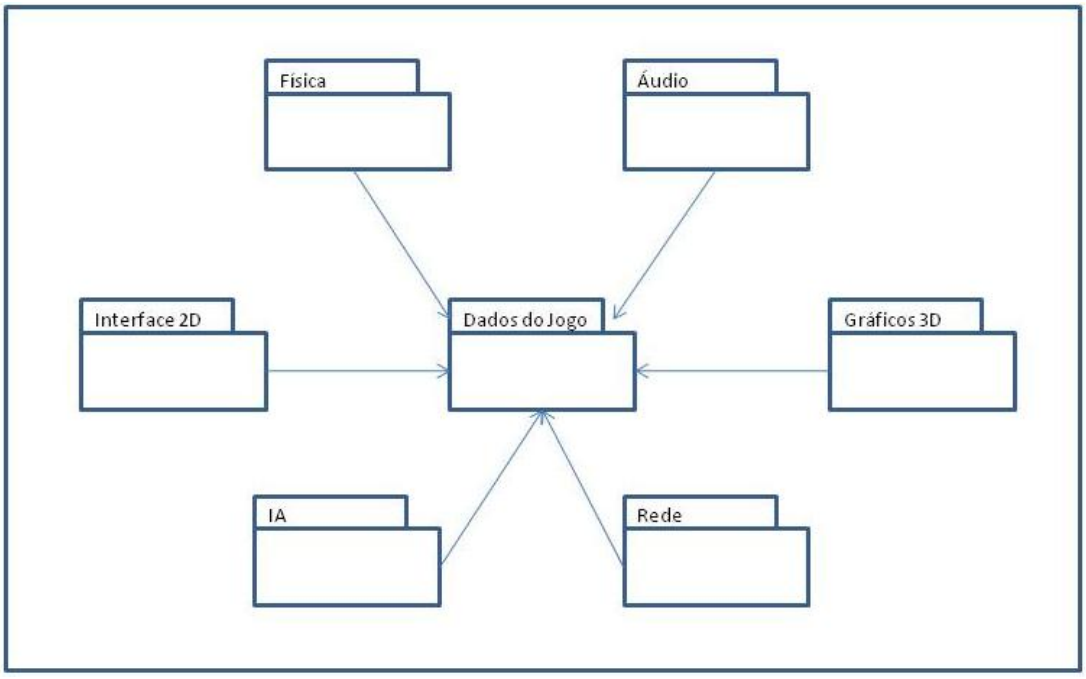


FIG.3.2 Arquitetura Centrada em Dados

3.4.3 COMPONENTES INDEPENDENTES

Processos independentes se comunicando através de mensagens definem a arquitetura de componentes independentes (FIG 3.3) (PRESSMAN, 2005). Os componentes registram os tipos de informações que eles podem processar e se comunicam através de mensagens. Uma vantagem interessante do estilo de componentes independentes é que nem todos os componentes precisam coexistir. Um sistema de comunicação desacoplado permite que nem todas as mensagens geradas necessitem possuir um recipiente. Um sistema bem projetado pode acrescentar e remover funcionalidades à vontade.

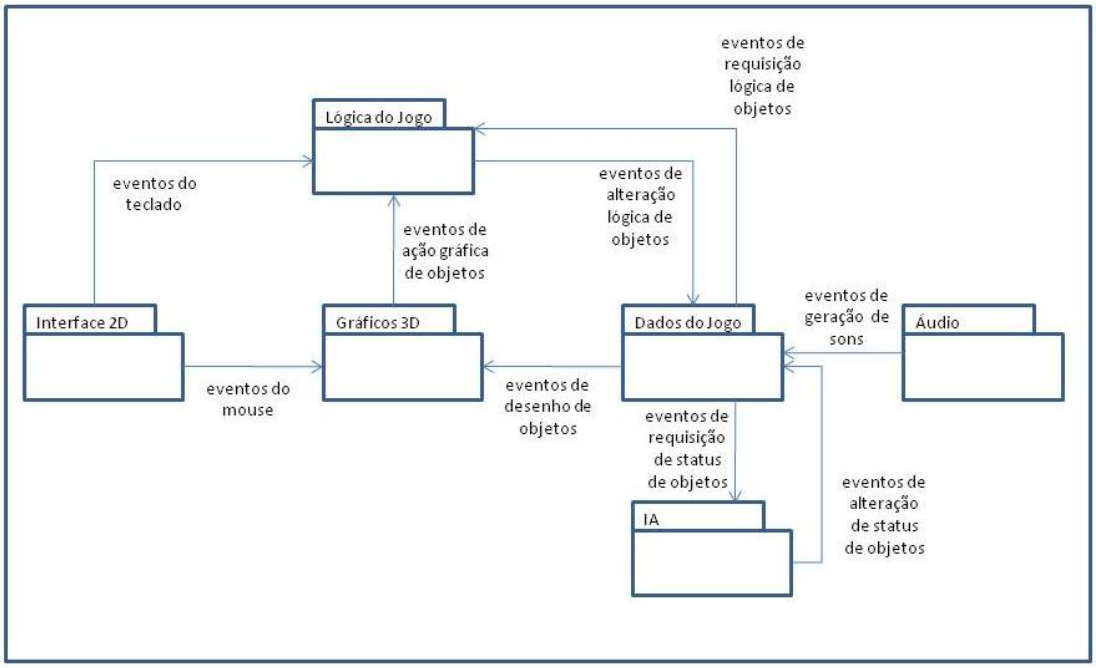


FIG.3.3 Arquitetura de Componentes Independentes

3.4.4 FLUXO DE DADOS

Estilos arquiteturais do tipo fluxo de dados (FIG 3.4) (PRESSMAN, 2005) são como canais e filtros, que tendem a oferecer grandes possibilidades de reuso, e são

geralmente fáceis de se manter e expandir. Concentrados em transformações incrementais de dados, tais sistemas podem ser facilmente compreendidos e alterados. Esses sistemas podem ser facilmente expandidos ou modificados simplesmente através da adição de novos componentes de processamento. O problema que surge é que os dados manipulados em cada estágio são completamente diferentes. E isso pode provocar uma sobrecarga para filtragem e interpretação dos dados em cada estágio. Porém ela pode ser interessante se aplicada a nível de componente, ou seja, dentro de um mesmo componente os dados (que nesse caso não serão diferentes) são processados em estágios.

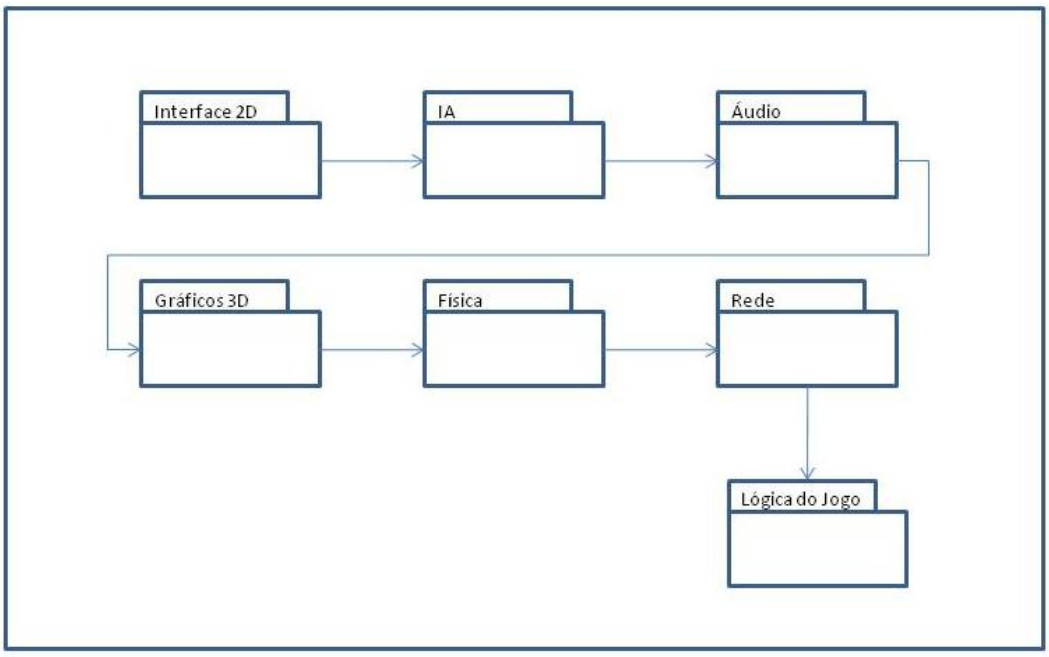


FIG.3.4 Arquitetura de Fluxo de Dados

3.4.5 SISTEMA DE SISTEMAS

A arquitetura de sistema de sistemas (BECK, et al., 1994) é parte do trabalho de engenharia realizado para integrar vários sistemas complexos. Essa abordagem é interessante porque tanto simuladores quanto aplicações comerciais precisam

muitas vezes integrar diferentes sistemas de domínios completamente diferentes. Gráficos, simulações de física, inteligência artificial, etc. são domínios distintos que precisam ser integrados numa única aplicação. Outro aspecto interessante desse estilo é que um sistema emerge da integração de subsistemas individuais. Em outras palavras, um simulador de combate é o resultado da integração de um sistema gráfico, um sistema de rede, um sistema de física, etc.

3.5 ARQUITETURAS DE JOGOS

Como foi visto no capítulo introdutório, tecnicamente a única diferença entre os jogos de guerra e os jogos de entretenimento (ou simplesmente jogos) são as qualidades dos dados manipulados por ambos. Nos jogos de guerra os dados são mais precisos e detalhados, de modo à melhor refletir a realidade, e muitas vezes envolvem informações técnicas sigilosas. Já no caso dos jogos comuns, os dados são definidos de modo apenas a resultar em uma boa jogabilidade, pois o foco é o fator entretenimento e não o realismo.

Sendo assim, é possível comparar a evolução das técnicas de desenvolvimento dos jogos de guerra com as técnicas empregadas nos jogos comuns. Isso é fato, tanto que nos dias de hoje a indústria militar de simulações trabalha mais do que nunca em cooperação com a indústria do entretenimento eletrônico. Tendo esse fato em vista, segue-se uma análise da evolução dos conceitos arquiteturais empregados nos jogos comerciais, que é aplicável ao universo dos jogos de guerra.

Os primeiros jogos de computador desenvolvidos seguiam um conceito monolítico de desenvolvimento. O código que executava todas as tarefas do jogo, como entrada de dados, desenho na tela e geração de sons, bem como o próprio armazenamento e manipulação de dados era concentrado num único bloco, sem módulos externos. Nos primórdios, tal técnica até poderia ser aceitável, tendo em

vista a simplicidade desses jogos. Porém, logo se constatou que este tipo de design era problemático, pois não oferecia flexibilidade nenhuma, além de ser de difícil manutenção e modificação (ROLLINGS, et al., 2004).

Uma das primeiras questões que levaram a uma quebra deste paradigma de desenvolvimento foi a questão da abstração do hardware. Num sistema monolítico, cada vez que a plataforma é alterada, o sistema inteiro tem de ser recompilado. A solução foi a criação de uma camada de software intermediária entre o hardware e o sistema propriamente dito, daí o conceito de que este software intermediário seria uma “abstração do hardware” por baixo dele.

No caso dos jogos, um marco nesse conceito de abstração foi o jogo *Quake II*, da empresa *iD Software.* Ele foi revolucionário no sentido de que permitia uma fácil portabilidade para várias plataformas, sem a necessidade de se alterar o código central do jogo. Ele empregou uma arquitetura do tipo cliente-servidor, mesmo na modalidade de jogo solo. Deste modo, todo o código independente de hardware, como o responsável pelo processamento da IA (inteligência artificial), pelos cálculos de posicionamento de objetos do jogo e pelo processamento da entrada fornecida pelo jogador era executado pelo módulo servidor. Já o módulo cliente, que poderia ser local (estar na mesma máquina) ou remoto (no caso de jogos em rede) continha o código dependente do hardware, responsável por gerenciar a entrada de dados dos dispositivos, bem como desenhar os dados gráficos, gerados pelo servidor, na tela do monitor. E a comunicação entre estes módulos era realizada então através de um protocolo também independente do hardware.

Seguiram-se então várias melhorias tecnológicas no hardware utilizado pelos jogos, o que provocou uma crescente complexidade na programação dessas camadas de abstração. A solução foi a criação de várias abstrações em separado, uma para cada tipo de hardware, ou mesmo para cada tipo de funcionalidade. Assim surgiram softwares intermediários (também conhecidos como “middlewares”) para gerenciamento de som, de gráficos 3D, de gráficos 2D, de física, de IA, etc. Com o

aumento da complexidade dos middlewares surgiu a necessidade do reuso dos módulos funcionais, pois as empresas começavam a dispender cada vez mais tempo e recursos, cada vez que tinham de reescrever o código desses middlewares do zero.

O próximo passo foi a evolução dos middlewares dos jogos, que se tornaram pacotes independentes, reutilizados pelas empresas de desenvolvimento de jogos. Esses módulos se tornaram bibliotecas dinâmicas, motores gráficos, plugins de áudio e vários outros tipos de componentes dos jogos. Gratuitos ou licenciados, esses pacotes permitiram que o desenvolvimento de novos jogos fosse focado no sistema de controle da lógica do jogo e na produção artística. As arquiteturas dos jogos modernos se concentram em como conectar todos os módulos disponíveis e permitir o acesso deles aos dados do jogo.

Porém, esses novos sistemas complexos ainda hoje são desenvolvidos seguindo uma certa lógica monolítica. Embora os módulos abstraiam as diferentes funcionalidades do sistema, em geral os projetos são construídos do zero, definindo uma arquitetura que usa um conjunto fixo de módulos, que não podem ser substituídos por diferentes módulos, embora eles possam ser melhorados e modificados. Os jogos assim desenvolvidos podem ser licenciados para terem apenas o conteúdo do jogo (arte e regras do jogo) alterado, dando origem a jogos derivados. É o caso, por exemplo, do jogo *America´s Army,* derivado do jogo *Unreal.*

Quando apenas autoriza-se a alteração do conteúdo artístico (geralmente sem custo de licenciamento), têm-se os chamados “*mods*” (abreviação de *modifications* - modificações). Contudo, essas formas de licenciamento restringem o acesso à tecnologia utilizada por trás dos módulos, o que pode ser um problema em maior ou menor grau, quando se pretende empregar os jogos comerciais em aplicações de treinamento militar.

3.6 CONCLUSÃO DO CAPÍTULO

Neste capítulo foram analisados os conceitos que envolvem a arquitetura de softwares, permitindo assim um entendimento da natureza fundamental da proposta deste trabalho. Além disso, foi analisado como evoluiu a arquitetura dos jogos, o que permite uma contextualização da proposta no cenário contemporâneo de projeto de jogos.

Assim, pode-se destacar neste ponto que a arquitetura proposta visa uma integração entre os conceitos arquiteturais tradicionais e os jogos modernos, de modo a garantir uma maior flexibilidade e liberdade no desenvolvimento de jogos de guerra, particularmente aqueles voltados para o treinamento tático.

4 O PROJETO DA ARQUITETURA ASTROS

ASTROS é o acrônimo para **A**rquitetura Modular para **S**istemas de **Tr**einamento Militar em **O**perações Táticas, sendo a designação escolhida para a arquitetura de software proposta neste trabalho (não confundir com o conjunto lançador de foguetes Astros, desenvolvido pela Avibrás). Portanto, no restante deste trabalho ela será referenciada apenas como arquitetura ASTROS. Essa designação foi assim escolhida, pois ela sintetiza a característica e o objetivo principais da arquitetura, ou seja, ela tem como característica fundamental a modularidade, e como objetivo imediato a aplicação em simulações para treinamento militar tático.

Nesse capítulo serão descritos os passos tomados para o projeto da arquitetura, bem como suas principais vantagens e desvantagens, além do detalhamento da estrutura da arquitetura propriamente dita, e de orientações para seu emprego no desenvolvimento de simuladores de combate.

4.1 ATRIBUTOS DA ARQUITETURA

O primeiro passo para o projeto da arquitetura ASTROS foi a definição de quais requisitos mínimos ela deveria atender, de modo a apresentar vantagens concretas sobre os métodos tradicionais de desenvolvimento de aplicações de simulação e jogos em geral. A escolha dos atributos foi realizada com base na análise de sistemas existentes como os jogos *Virtual Battle Space 1 e 2* (VBS08), além de outras arquiteturas genéricas de jogos de entretenimento 3D em primeira pessoa (os chamados “*first person shooters”*) (PLUMMER, 2004) (HADWIGER, 2000).

A seguir são listados os atributos que orientaram a estruturação da arquitetura, incluindo, além de uma descrição de cada um, a principal vantagem advinda da incorporação do atributo e possíveis desvantagens.

4.1.1 REUSABILIDADE DE PACOTES DE SOFTWARE JÁ EXISTENTES

Trata-se do reaproveitamento de componentes de software, como bibliotecas, módulos e plugins já desenvolvidos por outras empresas ou pessoas, e que são disponibilizados através de licenciamento, como software aberto, ou mesmo livres de qualquer restrição legal.

Esse é um dos principais atributos almejados no desenvolvimento da maioria dos jogos atuais, em menor ou maior grau. Isso se deve ao fator redução de custos, pois em muitos casos o custo de tempo e dinheiro para o desenvolvimento de um novo componente supera o custo de aquisição de um componente já disponível comercialmente ou até mesmo gratuitamente. Embora seja um fator crucial na indústria de entretenimento, a questão de redução de custos também é importante para o Exército, devido à limitação de recursos para o desenvolvimento de sistemas de treinamento.

Além da vantagem óbvia de redução de custos, outra vantagem do reuso de componentes já disponíveis é o fato de que eles já foram, em muitos casos, extensivamente testados, otimizados e empregados em outros sistemas, o que garantirá uma maior confiabilidade e melhor desempenho.

As desvantagens que podem advir do reuso de software estão ligadas principalmente a necessidade de se adaptar os componentes que serão reutilizados às necessidades específicas do sistema em desenvolvimento. Essa necessidade pode gerar dificuldades tanto maiores quanto mais elevados forem os níveis de abstração dos componentes reutilizados. Assim, como exemplo, pode ser mais difícil adaptar às necessidades específicas do sistema um motor gráfico completo, do que uma API gráfica de baixo nível, como o DirectX.

4.1.2 REUSABILIDADE DOS PRÓPRIOS COMPONENTES DA ARQUITETURA

Outra questão importante no reuso de software diz respeito também ao design interno de um sistema. Nesse caso, o reuso visa o reaproveitamento futuro dos próprios componentes desenvolvidos para o sistema. Esse reaproveitamento pode se destinar tanto a versões aprimoradas do sistema quanto a novos sistemas que possuam funcionalidades diferentes, mas que tenham em comum alguma funcionalidade encontrada num componente do sistema atual.

Como já foi visto no capítulo referente à engenharia de software, a reusabilidade é obtida a partir de um cuidadoso desenvolvimento modular, que permita o funcionamento independente dos componentes do sistema. As vantagens neste caso são a facilitação da manutenção futura e do desenvolvimento de expansões e modificações, pois os componentes podem ser mais facilmente substituídos e alterados.

4.1.3 FLEXIBILIDADE ESTRUTURAL

A flexibilidade estrutural surge em parte como conseqüência do reuso. Como o reuso depende de certa independência entre os componentes, ele permite uma maior flexibilidade de substituição e modificação de componentes. Porém a flexibilidade estrutural é mais abrangente, no sentido de que o emprego da arquitetura em si deve permitir diferentes configurações de seus componentes, ou seja, o desenvolvedor deve ter a possibilidade de optar por diferentes funcionalidades de cada componente, de acordo com a necessidade do sistema que está sendo projetado. Essa flexibilidade pode se refletir também em diferentes arranjos possíveis para o mesmo conjunto de componentes.

A principal vantagem da flexibilização da arquitetura é a possibilidade de ela atender a um maior número de sistemas, permitindo assim uma melhor personalização das funcionalidades desejadas pelo usuário final.

4.1.4 EXPANSIBILIDADE E MANUTENIBILIDADE

Expansibilidade de um sistema significa a possibilidade de se agregar funcionalidades ao sistema atual, sem que se façam necessárias alterações significativas. Em termos de arquitetura, isso significa que é possível adicionar novos componentes, executando outras funcionalidades em paralelo com os componentes atuais, sem que estes precisem ser modificados. Um exemplo num simulador de combate, que emprega IA para simular o comportamento dos inimigos, seria adicionar novos módulos de IA, de modo a acrescentar novos aspectos comportamentais aos inimigos, como um módulo de decisão tático aprimorado, sem alterar o módulo de IA pré-existente. A manutenibilidade surge então como conseqüência, pois ela representa a facilidade com que é possível modificar o sistema de modo a acrescentar melhorias, aperfeiçoamentos, além de corrigir erros e defeitos de projeto.

Obviamente estes atributos só contribuem positivamente para uma arquitetura que os possua, pois se tratam de características sempre desejáveis num sistema, que facilitam o desenvolvimento, aperfeiçoamento e atualização de um projeto, qualquer que seja sua natureza.

4.2 DECOMPOSIÇÃO MODULAR DAS FUNCIONALIDADES

Uma vez definidos os atributos que se deseja que a arquitetura possua, é necessário realizar a divisão das funcionalidades a serem oferecidas pelos componentes arquiteturais. Divisão essa que deve ser bem clara, de modo que não

ocorram conflitos e sobreposições de funcionalidades. Como será visto a seguir, essa divisão influenciará também a escolha do estilo arquitetural a ser seguido.

A arquitetura ASTROS objetiva o suporte ao desenvolvimento de jogos de guerra de foco tático. Um cenário de aplicação deste tipo de jogo de guerra envolve um ou mais usuários que deverão interagir através de uma interface do jogo com um ambiente virtual, que simule, por exemplo, um campo de batalha. Essa simulação deve representar como resultado visual um cenário tridimensional, onde estão imersos virtualmente outros usuários remotos (representados por avatares), objetos do jogo (como armas e veículos militares), inimigos e aliados virtuais (controlados por inteligência artificial) e representações do terreno (solo, vegetação, hidrografia, etc.). Para dar suporte a todos estes elementos, são necessárias então as seguintes funcionalidades, que devem estar claramente separadas:

Módulo gráfico 3D: responsável pelos cálculos, manipulação e apresentação do ambiente tridimensional virtual, bem como dos objetos nele inseridos, de modo a permitir a transformação dos dados sobre os cenários em informação visual, disponível em um monitor. Além do desenho dos elementos gráficos, esse módulo deve ser responsável pelo cálculo básico de colisões entre os elementos, pela delimitação dos elementos visíveis e pelos cálculos dos efeitos de iluminação.

Módulo de interface 2D: responsável pela apresentação da interface bidimensional de interação com o usuário, representada por elementos tais como caixas de texto, botões, janelas e caixas de seleção.

Módulo de entrada: responsável pelo controle dos dispositivos de entrada básicos e sua integração com os módulos gráfico 3D e de interface 2D. Os dispositivos considerados devem ser pelo menos mouse e/ou joystick e teclado.

Módulo de rede: responsável por prover e controlar a comunicação remota entre as diferentes instâncias do jogo de guerra que empregue a arquitetura. Até este ponto não é relevante se a arquitetura será do tipo cliente-servidor ou cliente-cliente. Está escolha não deve afetar a independência do módulo de rede, pois este deve se limitar a prover uma interface de troca de dados entre instâncias (clientes ou servidores) separadas por um meio de rede qualquer (rede local ou internet).

Módulo de áudio: responsável pelo controle de reprodução de efeitos sonoros, vozes e músicas, por meio de alto-falantes ou fones de ouvido, durante o uso da interface 2D e principalmente durante a execução da simulação de combate.

Módulo de física: responsável pela simulação dos efeitos físicos mecânicos sobre os objetos contidos no ambiente virtual, tais como gravidade, detecção de colisão aprimorada, inércia, atrito, etc.

Módulo de inteligência artificial (IA): responsável pelo controle das ações de inimigos e aliados não controlados por outros usuários, determinando o uso de táticas (básicas, como ataque e fuga, e avançadas, como ataque coordenado e busca de cobertura), além do cálculo de melhores caminhos através do ambiente virtual.

Para assegurar que só exista uma instância de cada módulo lógico, deve ser empregado o padrão de projeto criacional *singleton* (GAMMA, et al., 1995).

4.3 ESCOLHA DO ESTILO ARQUITETURAL

A escolha da topologia determinará a estrutura a partir da qual a arquitetura irá evoluir. Ela determinará como os sistemas poderão ser ampliados e modificados, tendo assim um grande impacto sobre os atributos finais desejados. As topologias arbitrárias normalmente sobrecarregam a comunicação entre os módulos de modo a

manter os subsistemas realmente independentes, que é um requisito fundamental para a arquitetura ASTROS. Embora o desempenho não seja um requisito chave para a arquitetura, existem outros estilos capazes de atender aos atributos desejados sem comprometer muito o desempenho.

A arquitetura de fluxo de dados parece não ser apropriada, pois os domínios lógicos (funcionalidades dos módulos) são muito diferentes. Tentar projetar um canal de dados universal para todos os dados envolvidos num jogo de guerra não parece a abordagem correta. A análise realizada sugere que um estilo arquitetural de fluxo de dados não é o ponto de partida ideal para o tipo de abstração desejada para o sistema.

O estilo em camadas é mais estruturado e poderia possivelmente permitir um melhor desempenho que as outras topologias menos estruturadas. A topologia em camadas, porém, não permite uma fácil abstração das funcionalidades de modo a minimizar os efeitos da mudança de tecnologias. Em geral, mudanças de tecnologia acarretam um grande impacto na manutenibilidade de um sistema, pois se faz necessário substituir componentes fortemente ligados aos componentes de lógica do jogo. Como as tecnologias estão em constante evolução, é importante que a atualização dos módulos que utilizem tecnologias seja rápida e prática. Uma solução nesse caso seria colocar o controle de lógica do jogo em uma camada distinta, porém isso acabaria recaindo numa topologia de fluxo de dados, com os problemas já apresentados.

Finalmente, a topologia centrada em dados foi escolhida para uma análise mais aprofundada, pois ela permite um bom balanceamento de flexibilidade e desempenho. As outras abordagens possuem características realmente desejáveis, porém possuem também uma quantidade significativa de desvantagens, difíceis de serem superadas. A abordagem da centralização de dados ainda permite uma boa independência dos subsistemas e um acesso direto aos dados do jogo. Neste ponto

fica evidente que um design centrado em dados oferece as melhores qualidades para a estruturação de uma arquitetura que alcance os atributos desejados.

A utilização de uma topologia centrada em dados prioriza principalmente a flexibilidade no uso de tecnologias, ao mesmo tempo em que permite o reuso de componentes já disponíveis gratuita ou comercialmente. Dada a grande complexidade de um jogo de guerra moderno, o desenvolvimento a partir do zero de cada componente é inviável. Assim, uma arquitetura que permita uma fácil integração de componentes pré-fabricados facilitará em muito o trabalho dos desenvolvedores que vierem a empregar a ASTROS. Essa abordagem ainda tem a vantagem de não eliminar a possibilidade de uso de outras topologias em outros níveis de abstração, como por exemplo, a topologia em camadas, que permite a criação de uma camada exclusivamente de lógica de controle e outra tecnológica dos módulos.

4.4 TRANSMISSÃO DOS DADOS

Uma vez que foi definida a topologia a ser seguida pelos módulos lógicos, deve ser definido como os dados serão transmitidos. Os módulos poderão trocar informações entre si, porém deve ser definido como ocorrerá essa troca. No caso da topologia centrada em dados, existem dois métodos possíveis para a comunicação entre os clientes e os dados (BERARD, 1992).

O primeiro método é o repositório de dados, onde os dados residem em um repositório passivo e os clientes são responsáveis por obter os dados e decidir se eles foram alterados. O repositório é provavelmente o método mais simples, pois os dados são armazenados segundo a lógica de um banco de dados. A técnica de armazenamento em si não é relevante, pois pode ser através da serialização de estruturas de dados em arquivos ou mesmo através de um sistema de banco de dados de fato. O único problema deste método é o intenso tráfego de dados entre os

clientes e o repositório, pois o acesso aos dados é realizado mesmo que eles não tenham sido alterados.

O segundo modelo de comunicação de dados é o modelo conhecido como “quadro-negro”. Neste modelo o repositório de dados é ativo e envia mensagens de atualização para os clientes, os informado sobre alterações nos dados. O método do quadro-negro é uma tentativa de reduzir a quantidade de comunicação e ao mesmo tempo manter uma sincronia entre os clientes e os dados armazenados.

Para o caso de um jogo de guerra o modelo de repositório é o mais adequado. Em primeiro lugar, a questão do intenso tráfego entre o cliente e o repositório de dados não é um problema significativo, pois ambos estarão localizados no mesmo computador. Em segundo, alguns módulos precisarão acessar os dados mesmo que estes não tenham sido alterados. Um exemplo é o módulo gráfico 3D, que precisa acessar os dados sobre a posição de um objeto mesmo que este não tenha se movido desde a última vez que ele foi desenhado. E finalmente, o modelo de repositório também possui a vantagem de que todos os dados são mantidos em uma única área, o que libera os módulos da necessidade de manter uma cópia local dos dados. Em termos de padrões de projeto, essa solução pode ser implementada através do padrão criacional *object pool* (GAMMA, et al., 1995).

4.5 SINCRONIZAÇÃO DOS DADOS

Outra questão importante é a sincronização dos dados. Ela trata do modo como os dados e os controles passam de um módulo para outro. Como a sincronização está fortemente relacionada com a topologia e com o método de comunicação escolhidos algumas opções já podem ser descartadas. Por exemplo, como o método de comunicação “quadro-negro” foi descartado, os métodos de sincronização assíncronos não são provavelmente boas escolhas. Os métodos de desenvolvimento tradicionais de jogos normalmente utilizam uma técnica de

“marcação” dos objetos do jogo, o que é uma prova de que os jogos podem empregar uma abordagem síncrona.

Existem basicamente duas abordagens síncronas (BERARD, 1992). A primeira é a sincronização a nível de objetos. Neste caso todas as funcionalidades de um objeto são completadas antes do processamento passar para outro objeto. Ou seja, são realizadas todas as operações de simulação física, IA e desenho em um objeto antes de se passar para o próximo.

Outra abordagem é a sincronização em blocos. Neste caso um conjunto de objetos é completamente processado antes de se passar para o próximo grupo. Um exemplo seria a realização de todos os cálculos de IA de um grupo de objetos antes de eles serem desenhados. Este tipo de abordagem faz mais sentido no caso de funcionalidades mais complexas que dependem de grande interação entre objetos.

Atualmente costuma-se empregar no desenvolvimento de jogos uma abordagem que é híbrida dessas duas já apresentadas, ou seja, trata-se de uma mistura entre a sincronização a nível de objetos e a nível de componentes. Assim, a “marcação” de um objeto pode resultar no cálculo de sua IA e geração de sons, enquanto que o desenho de todos os objetos é realizado em um único bloco. Isso é um reflexo da própria evolução dos jogos. Nos primórdios, os jogos eram suficientemente simples para permitir a sincronização a nível de objeto. À medida que a tecnologia começou a se tornar mais complexa, passou a ser mais fácil programar “motores” completos para realizar operações como renderização gráfica como uma operação em blocos.

Para a arquitetura ASTROS optou-se pela sincronização em blocos, por alguns motivos importantes. Em primeiro lugar, a sincronização a nível de objeto não faz muito sentido no caso de uma topologia centrada em objetos, com o uso de um repositório passivo. A sincronização a nível de objeto elimina completamente a

funcionalidade independente dos módulos ao redor de um repositório de dados comum, pois todos os módulos precisariam operar num objeto antes que o objeto seguinte pudesse ser acessado. Em segundo lugar, o objetivo principal deste trabalho é lidar com o processamento de dados de domínios específicos cada vez mais complexos. E como é comum o uso de "motores" nos jogos tradicionais para executar cálculos em lotes, esse tipo de sincronização também é o mais adequado para a arquitetura ASTROS.

4.6 ORGANIZAÇÃO DA ARQUITETURA

4.6.1 UMA VISÃO GERAL DA ARQUITETURA

O conceito da topologia centrada em dados, empregada na arquitetura ASTROS, define que todos os dados do jogo se concentram num repositório único. Estes dados podem então ser acessados por cada um dos módulos de maneira independente. Cada módulo opera sobre um domínio lógico diferente, ou seja, é responsável pela manipulação de um tipo específico de dado, atuando como uma espécie de subsistema. Alguns dados podem ser acessados por mais de um módulo, como no caso da posição espacial dos objetos visíveis, que pode ser acessada pelo módulo gráfico e pelo módulo de física. No entanto, a ação de um módulo ocorre de forma independente do outro, ou seja, sem necessidade de comunicação direta entre os módulos. A estrutura geral da arquitetura ASTROS está representada na FIG 4.1.

Uma primeira vantagem que pode ser notada nesse tipo de estrutura é que um módulo lógico pode ser substituído por outro sem que sejam necessárias alterações em outros módulos. Essa característica contribui, portanto, para os atributos de flexibilidade e manutenibilidade, já mencionados anteriormente. Um aparente problema que poderia ocorrer nessa estrutura seria com respeito à questão de velocidade. Como cada módulo precisa acessar os dados de todos os objetos do

jogo, isso poderia comprometer o desempenho do sistema. A solução é a utilização de um acesso seletivo aos dados apenas dos objetos relevantes para o avatar do jogador. Esse acesso seletivo é provido então através de um componente de gerenciamento de objetos, que define qual dado é relevante para cada módulo a cada momento. Assim, a estrutura fica mais bem representada pela FIG 4.2.

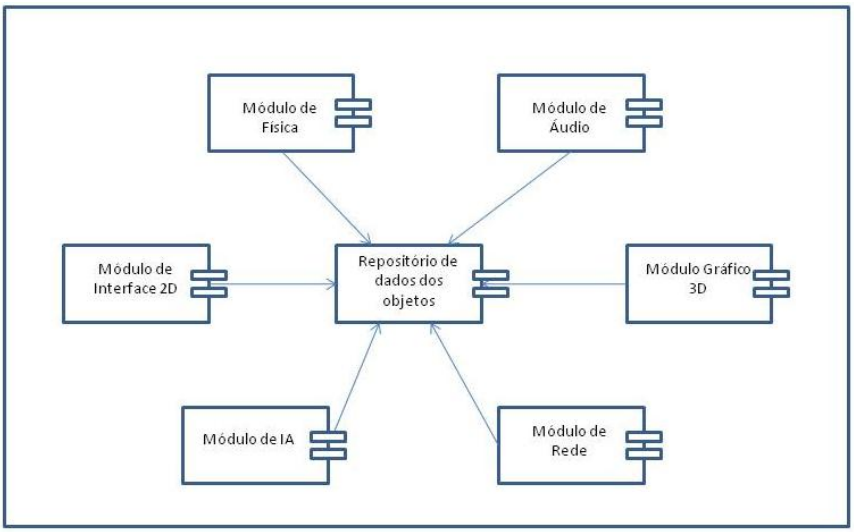


FIG.4.1 Visão geral da arquitetura ASTROS

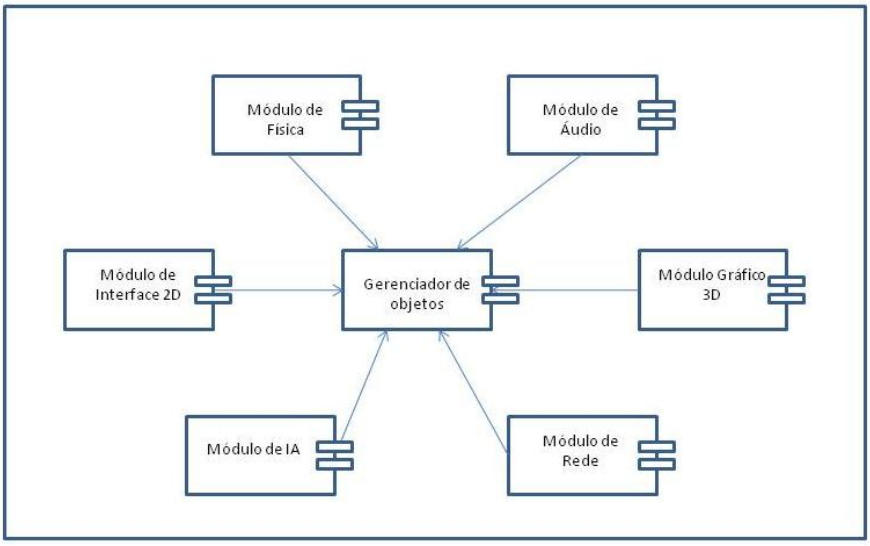


FIG.4.2 Gerenciamento de Objetos

4.6.2 SISTEMA DE COMUNICAÇÃO

Como foi visto anteriormente, a comunicação entre os componentes de uma arquitetura centrada em dados pode ser realizada através de um repositório de dados ativo ou passivo. Como se optou pela técnica do repositório passivo e os módulos lógicos possuem comunicação independente, o sistema de comunicação é composto por uma conexão unidirecional (como uma chamada de função, por exemplo) entre cada módulo e o componente de gerenciamento de objetos (ver FIG 4.3).

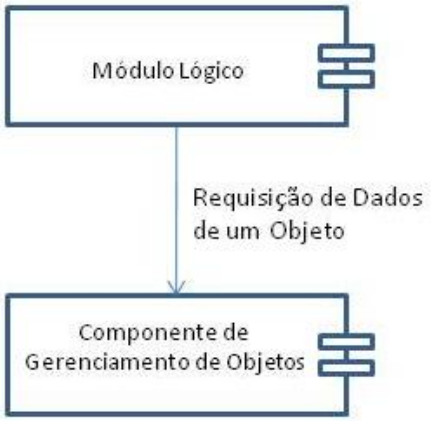


FIG.4.3 Modelo de Comunicação

De modo a manter a flexibilidade, a expansibilidade e a manutenibilidade da arquitetura, esse sistema de comunicação deve empregar classes do tipo interface, que contenham apenas funções virtuais. O objetivo do uso de interfaces é tornar a comunicação entre os dados e os módulos lógicos independente de como cada módulo é implementado e de que tecnologia ele utiliza. Neste caso, para que esse objetivo seja atendido, deve ser empregado o padrão de projeto criacional *abstract factory* (GAMMA, et al., 1995).

4.6.3 SINCRONIZAÇÃO DOS DADOS

Observando-se a estrutura proposta para a arquitetura, é possível notar que não existe nenhuma comunicação direta entre os diferentes módulos lógicos, que permita que um componente informe ao outro sobre alterações sofridas nos objetos. Dependendo do tipo de aplicação, esta falta de sincronismo entre os módulos poderia representar um sério problema devido à possibilidade de inconsistências nos dados. Isso ocorreria, pois, ao final de uma interação sobre todos os objetos, um módulo pode determinar que um objeto já não mais exista (e.g. módulo de IA) embora outro módulo tenha operado sobre esse mesmo objeto (módulo gráfico por ex.). No caso de uma simulação de combate em tempo real esse tipo de problema não é relevante, já que cada interação entre todos os objetos corresponde a um único quadro exibido no monitor. Assim, embora possam ocorrer inconsistências, elas são corrigias tão rapidamente (em frações de segundo) que o usuário dificilmente conseguirá perceber.

A arquitetura ASTROS possui sincronização de dados, porém ela ocorre a nível de módulos ao invés de objetos. Ao contrário de arquiteturas que realizam a sincronização a nível de objetos, não é necessário marcar um objeto cada vez que uma operação, tal como desenho, som, cálculo de IA, etc., precise ser realizada sobre ele. Deste modo, cada módulo executa suas operações em todos os objetos de cada vez. Isso implica que deve existir um componente mestre que notifique cada módulo quando este deve executar suas operações sobre os objetos (ver FIG 4.4).

4.6.4 SINCRONIZAÇÃO DISTRIBUÍDA

Até agora se assumiu que todos os módulos estão operando na mesma máquina. O método de sincronização proposto não seria factível caso cada módulo estivesse localizado em máquinas diferentes. Porém, isso não significa que diferentes instâncias do sistema possam funcionar de modo distribuído operando

sobre o mesmo conjunto de objetos. De fato, para que seja possível o funcionamento do sistema em rede basta adicionar um componente de rede ao sistema local. Tal abordagem permitiria tanto uma comunicação ponto-a-ponto quanto uma comunicação cliente-servidor (ver FIG 4.5).

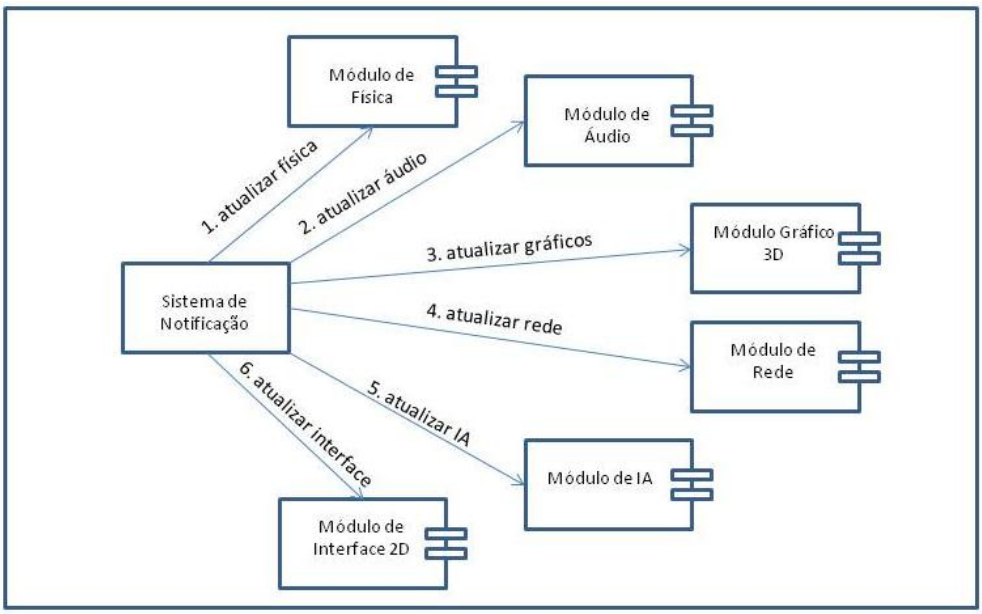


FIG.4.4 Método de Sincronização

A arquitetura ASTROS, entretanto, deve utilizar somente o modo de comunicação cliente-servidor. Essa decisão foi tomada com base na experiência de outros sistemas de simulação de combate, que empregam em sua totalidade essa topologia. Além disso, há o fato de que em redes com retardos elevados é muito difícil manter a sincronia entre todos os clientes numa topologia ponto-a-ponto, devido ao intenso tráfego replicado entre todos os clientes. Conseqüentemente, haveria uma perda significativa de desempenho, acarretando taxas de atualização de quadros baixas (problema conhecido como “*lag*”).

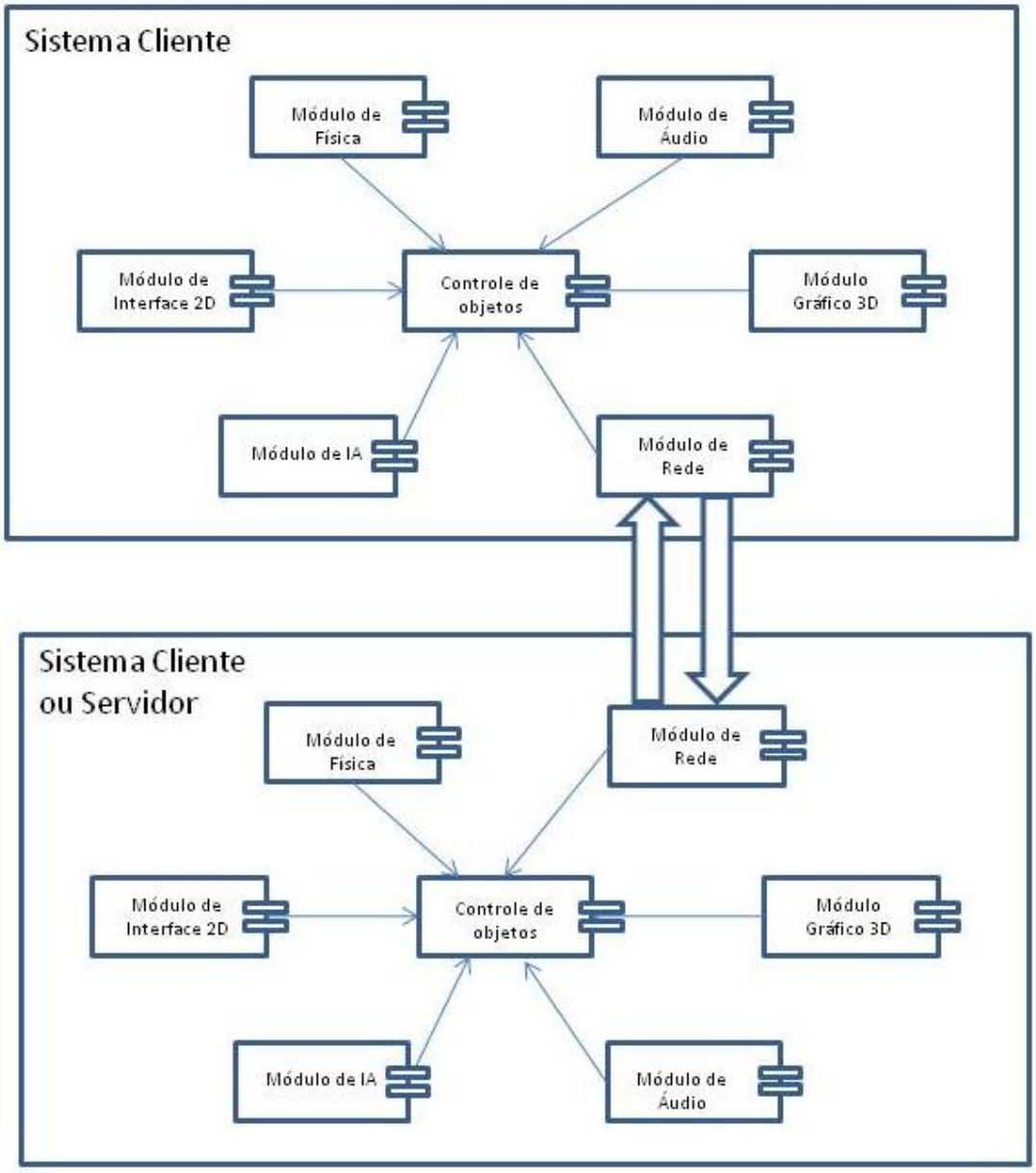


FIG.4.5 Comunicação Ponto-a-Ponto ou Cliente-Servidor

4.6.5 OBJETOS GENÉRICOS

A última particularidade a ser analisada quanto à arquitetura ASTROS é o modo de armazenamento de dados dos objetos. Como já foi visto, os objetos são armazenados num repositório único e o acesso a eles é feito por meio de um

componente gerenciador de objetos. A comunicação entre este componente e os módulos lógicos se dá, no entanto, através de interfaces que ocultam os detalhes de implementação de cada módulo. Surge aí um problema quando um determinado módulo precisa guardar dados de um objeto que são dependentes de implementação. Por exemplo, no caso do módulo gráfico, um objeto visível no ambiente virtual tridimensional requer que sejam armazenados dados sobre a malha e a textura que o compõe. Estes dados podem ser específicos para determinado tipo de motor gráfico empregado, pois a malha pode ser representada por uma estrutura de dados particular, bem como a textura pode estar armazenada num formato de arquivo particular. Assim, ocorre a necessidade de armazenar esse tipo de dado de maneira transparente nos objetos.

A solução é a utilização do padrão de projeto comportamental *observer* (GAMMA, et al., 1995). A cada objeto genérico deve ser possível anexar, por meio de uma requisição de um módulo lógico, um objeto observador que contém os dados específicos (dependentes de implementação) a serem armazenados. Deste modo, cada objeto genérico do repositório pode conter um ou mais objetos observadores, que contém dados específicos, dependentes da implementação utilizada pelos módulos que os criaram (ver FIG 4.6).

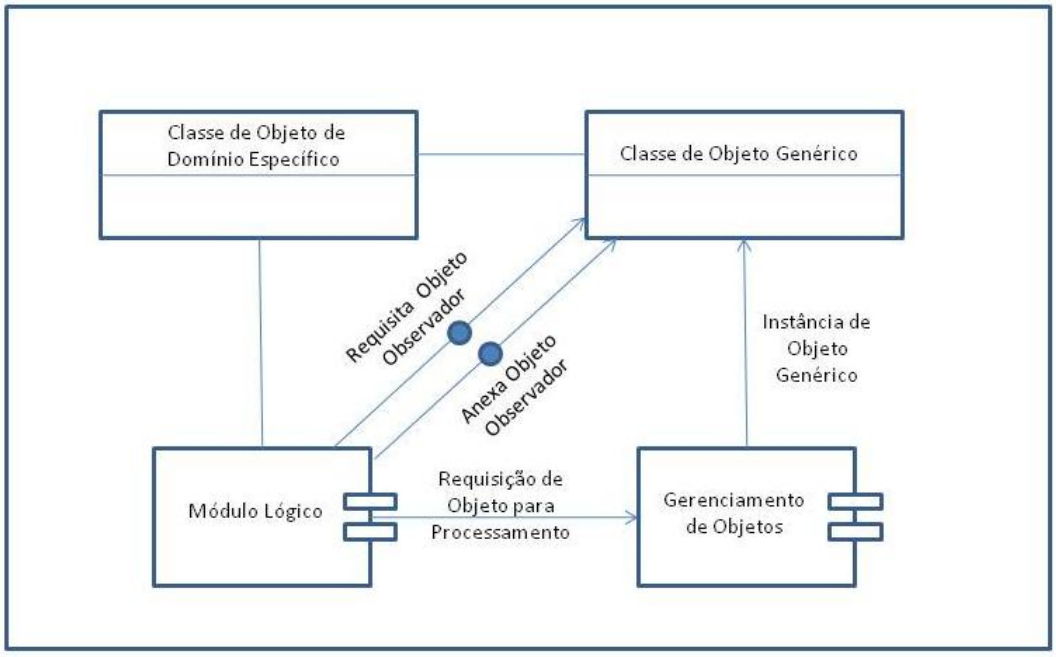


FIG.4.6 Objeto Observador

4.7 CONCLUSÃO DO CAPÍTULO

Neste capítulo foram delineados todos os requisitos que um sistema de simulação de combates deve atender para que ele possa seguir a arquitetura ASTROS de modo a tirar o máximo proveito de suas vantagens.

Embora a arquitetura proposta sirva como uma orientação para o desenvolvimento de sistemas de treinamento militar, ela possui atributos que permitem sua extensão para outros tipos de sistemas de treinamento que envolvam a simulação de ambientes virtuais tridimensionais, com a interação entre avatares controlados por usuários humanos, avatares controlados por IA e objetos virtuais genéricos.

5 UM PROTÓTIPO DE USO DA ARQUITETURA ASTROS

Para demonstrar a utilização da arquitetura ASTROS foi elaborado um protótipo de sistema de simulação de combates táticos, que emprega os principais conceitos abordados até o presente momento. Longe de ser um sistema totalmente operacional, o protótipo desenvolvido teve como objetivo apenas comprovar a viabilidade e as vantagens da adoção da arquitetura proposta em relação a qualquer outra metodologia tradicional de desenvolvimento de jogos de guerra táticos.

Neste capítulo serão descritas as ferramentas empregadas na elaboração do protótipo, tais como bibliotecas e plugins, bem como os componentes desenvolvidos, em termos de classes, métodos, atributos e relacionamentos.

5.1 LINGUAGEM DE PROGRAMAÇÃO E PLATAFORMA

Foram consideradas inicialmente para a elaboração do protótipo as linguagens Java e C++. Ambas possuem características particulares que as tornam extremamente recomendáveis para a programação orientada a objetos. Tais características envolvem modularidade, reusabilidade e robustez. Além disso, elas são portáveis para várias plataformas, uma característica definida como essencial para a elaboração do protótipo. Outra característica importante compartilhada por ambas é a compatibilidade com diversas bibliotecas multimídia (ROLLINGS, et al., 2004).

Porém C++ possui uma versatilidade maior do que Java quando se trata de questões de programação de baixo nível e otimização. Ao mesmo tempo em que C++ possui atributos que permitem a aplicação de padrões de projeto de software, como Java, também possui características da linguagem C, isto é, a possibilidade de tratamento de detalhes da programação de baixo nível, como o acesso direto aos

dispositivos de hardware e o gerenciamento de memória (fundamental para assegurar o desempenho em aplicações de intenso processamento gráfico).

Historicamente, o desenvolvimento de jogos de computador, sobretudo aqueles 3D, tem sido realizado com a linguagem C, devido ao seu alto desempenho e possibilidade de otimização de código. Contudo, com o avanço das tecnologias de hardware gráfico e de otimização em compiladores, C++ passou a ser cada vez mais considerado como uma melhor opção, pois permite uma maior organização de projetos complexos, além de facilitar a manutenção e modificação destes.

Assim, C++ foi a linguagem escolhida devido a sua portabilidade, possibilidade de emprego do paradigma de programação orientada a objetos, compatibilidade com ampla gama de recursos multimídia voltados para o desenvolvimento de jogos, desempenho satisfatório frente a linguagens de mais alto nível, como Java, e sua flexibilidade para programação em baixo nível.

Quanto à questão da plataforma, como já foi dito, o objetivo foi de se desenvolver um sistema portável para qualquer plataforma. Desse modo, todas as escolhas subseqüentes foram tomadas tendo em mente esta objetivo. Grande parte dos projetos de jogos nos dias de hoje são sempre direcionados para uma plataforma específica, o que demanda um grande retrabalho quando da necessidade de portar o sistema para outra plataforma. Além disso, como a arquitetura ASTROS objetiva também uma redução de custos para as Forças Armadas, um sistema que possa se adaptar aos padrões de software livre é altamente desejável.

5.2 BIBLIOTECAS EXTERNAS

Nessa seção serão analisados diversos recursos que podem ser utilizados na implementação de um sistema baseado na arquitetura ASTROS. Muitas escolhas aqui realizadas não foram de modo alguma definitivas, pois priorizou-se em muitos

caso a facilidade de utilização sobre outros fatores como versatilidade e desempenho. Sendo assim, as opções aqui apresentadas devem ser tomadas como uma sugestão e um incentivo a exploração de outros recursos para o desenvolvimento de um sistema real.

5.2.1 BIBLIOTECAS PARA O MÓDULO DE ENTRADA

São bibliotecas que tem como objetivo capturar todo tipo de entrada fornecida pelo usuário, através dos dispositivos de entrada, tais como mouse, teclado, trackball, joystick e controladores de jogos. Servem também para prover um sistema de mapeamento de ações, ou seja, permitem que seja associado a um determinado evento de entrada (o pressionamento de certo botão do mouse, por exemplo) um evento do sistema, como por exemplo, uma chamada de função. Em alguns casos essas bibliotecas dispõem também de controle de dispositivos com *force-feedback*, ou seja, que recebem também uma saída do sistema. A interface deste tipo de biblioteca deve assegurar transparência para o programador, sendo independente do tipo, marca ou modelo de dispositivo de entrada a ser usado.

As bibliotecas analisadas foram:

- ***DirectInput***: é uma API (*Application Programming Interface* – Interface de Programação de Aplicativo) desenvolvida pela Microsoft como parte do *DirectX*, que é um conjunto de APIs multimídia presente desde o sistema operacional Microsoft Windows 95. O *DirectInput* dá suporte a todos os dispositivos básicos de entrada e possui suporte a *force-feedback.* A partir do *DirectX* 9 foi desenvolvido o *XInput* com o objetivo de dar suporte aos controladores do console Xbox 360. A grande desvantagem dessa API é sua dependência do sistema operacional Windows, o que impossibilita sua portabilidade. (Dir08)

- ***SDL (Simple DirectMedia Layer)***: é uma biblioteca multimídia multiplataforma (suporte para Windows, Linux, BeOS, Solaris, IRIX, FreeBSD, MacOS) que provê acesso de baixo nível a diversos dispositivos de entrada, como mouse, teclado e joystick, além de outros dispositivos multimídia, como placa de som e placa gráfica (via OpenGL). Desenvolvida em C++, possui interface para várias outras linguagens como Ada, Eiffel, Java, Lua, ML, Perl, PHP, Pike, Python, e Ruby). Além de ser multiplataforma, possui a grande vantagem de estar disponível sobre a licença GNU LGPL. Não foi escolhida, pois seria empregada apenas para o módulo de entrada, tendo em vista melhores opções para os outros módulos. (Sim08)

- ***OIS (Object Oriented Input System)***: outra solução multiplataforma, com suporte para todos os dispositivos básicos de entrada (mouse, teclado e joystick), além de suporte a *force-feedback.* Escrito em C++ e Phyton utilizando orientação a objeto e padrões de projeto, é extremamente simples de ser utilizada, pois possui um design claro e objetivo, atendendo perfeitamente aos objetivos que se propõe. Seu código fonte é aberto, distribuído sobre a licença zlib/libpng, sendo portável para Windows e Linux. Devido a todas estas características, foi a opção escolhida. (Obj08)

5.2.2 BIBLIOTECAS PARA O MÓDULO DE INTERFACE 2D

O módulo de interface 2D, embora seja independente logicamente do módulo gráfico 3D, normalmente dependerá tecnologicamente deste último. Isso acontece, pois as rotinas de desenho de imagens 2D (janelas, botões, etc.) em geral fazem parte das funcionalidades das bibliotecas gráficas 3D ou são no mínimo dependentes delas. Tomando como exemplo, a biblioteca GLUT (*OpenGL Utility Toolkit*), ela depende da utilização de *OpenGL* como sistema gráfico 3D, o que

impossibilitaria seu uso conjunto com *DirectX.* Outro exemplo é o *DirectDraw*, que é outra API que faz parte do pacote *DirectX*, sendo portanto, dependente deste.

Sendo assim, as opções consideradas para a criação de elementos de interface 2D foram selecionadas dentre as que são voltadas para o uso com a biblioteca gráfica *Ogre 3D* (foi a biblioteca 3D escolhida, ver próxima seção). Como conseqüência, todas as bibliotecas analisadas são multiplataforma (Windows, MacOS e Linux) e possuem código fonte aberto.

As bibliotecas analisadas foram:

- ***MyGUI:*** opção interessante por ser bem simples e de fácil uso, porém possui poucos recursos e não dispõe de um editor de interfaces, o que limitaria e dificultaria seu uso em um sistema de uso real. (MyG08)

- ***Open GUI:*** promete várias funcionalidades interessantes como sistema de eventos similar ao do *Windows Forms*, além de permitir a virtualização da interface, podendo assim ser exibida em mais de um *viewport*. A desvantagem é que se encontra desatualizada, além de possuir pouco suporte. (Ope083)

- ***CEGUI System****:* ao contrário das anteriores, possui diversas funcionalidades, um editor de interfaces, vários exemplos e tutoriais, além de estar em constante atualização e possuir uma grande comunidade que lhe dá suporte. Outra vantagem é que além da integração com a biblioteca gráfica *Ogre 3D,* pode ser facilmente integrada à biblioteca gráfica *Irrlicht* ou diretamente às APIs *DirectX* e *OpenGL*. Por todas essas vantagens, foi a opção escolhida. (Cra08)

5.2.3 BIBLIOTECAS PARA O MÓDULO GRÁFICO 3D

As opções mais óbvias seriam as APIs gráficas *OpenGL* e *DirectX*. *DirectX* poderia ser descartada a princípio por ser exclusiva das plataformas Windows. Porém o maior problema, inerente a ambas, é que elas são APIs gráficas de baixo nível, ou seja, só tratam de primitivas básicas como pontos, polígonos e texturas. Para suprir as necessidades do módulo gráfico 3D seria necessário programar um motor gráfico completo, que permitisse a manipulação de objetos gráficos complexos, como malhas de pontos e animações, além de efeitos de iluminação, gerenciamento de imagens, gerenciamento de cenas, etc. Esse tipo de projeto fugiria ao escopo deste trabalho além de ser desnecessário. A solução é a utilização de motores gráficos já existentes que incorporam todas as funcionalidades já citadas e muitas outras, além de permitirem em alguns casos que se opte em tempo de execução pela API *OpenGL* ou *DirectX*, quando esta estiver disponível.

Os principais critérios adotados para a seleção de um conjunto inicial de motores gráficos foram: código fonte aberto (licença GPL ou LGPL), atualizado, possuidor de comunidade desenvolvedora ativa, bem documentado e de uso não excessivamente complexo. Além disso, deveria possuir no mínimo as seguintes funcionalidades: suporte a iluminação por pixel e por vértice, suporte a mapas de iluminação, suporte a texturas simples, multi-texturas e mapas de rugosidade, suporte a algum tipo de algoritmo de gerenciamento de cena, suporte a animações, suporte a malhas de polígonos complexas, suporte a efeitos diversos (de água, céu, partículas, névoa e *sprites*) e suporte a renderização de terrenos.

Sendo assim, os motores gráficos considerados foram:

- ***Irrlicht:*** um motor gráfico relativamente novo, não possui tantas funcionalidades e recursos quanto um motor gráfico comercial, porém possui uma crescente comunidade de desenvolvimento, bem como

vários projetos que a utilizam. Suporta tanto *DirectX* quanto *OpenGL*, além de possuir seu próprio renderizador por software. Está bem documentado, além de possuir diversas ferramentas de suporte como editores, exportadores e ligações para outras linguagens, como LUA, Perl e Java. A única desvantagem realmente é a falta de algumas funcionalidades mais avançadas, como a possibilidade de configuração mais detalhada dos gerenciadores de cena e a composição de materiais. (irr081)

- ***Crystal Space:*** é um motor gráfico mais maduro do que o *Irrlicht*, possui mais funcionalidades, inclusive com suporte para som. Possui um conjunto de ferramentas para o desenvolvimento de jogos conhecido como CEL (*Crystal Entity Layer*). Entretanto, além desses recursos adicionais não serem úteis para a arquitetura ASTROS, ela funciona apenas com *OpenGL*, e sua documentação e suporte não são tão bons quanto os dos motores *Irrlicht* e o *Ogre 3D*. (Cry08)

- ***Ogre 3D:*** é um motor gráfico equiparável aos motores comerciais, possui todas as funcionalidades que poderia se esperar, sua comunidade de desenvolvimento e suporte é enorme. Possui também um amplo repositório de ferramentas para auxilio no desenvolvimento, na criação de conteúdo e plugins para várias outras bibliotecas. Embora seu uso seja a princípio um pouco complexo, é recompensado por uma extensa documentação, exemplos e tutoriais. Foi a opção escolhida. (Ogr08)

5.2.4 BIBLIOTECAS PARA O MÓDULO DE REDE

São as bibliotecas responsáveis pelo empacotamento, envio e recebimento de pacotes de dados através de dispositivos de rede para outro sistema, além do

tratamento dos erros advindos da comunicação em rede. O único requisito realmente exigido para a biblioteca de rede é que ela fornecesse suporte para comunicação via UDP, tendo em vista que os jogos utilizam um intenso tráfego de rede, que prioriza velocidade ao invés de confiabilidade.

As opções consideradas foram:

- ***Windows Sockets API:*** bastante simples e fácil de usar, possui todas as funcionalidades básicas de rede, com suporte para TCP e UDP. Possui a desvantagem de ser exclusiva para Windows. Embora exista uma versão equivalente para Linux, a *BSD Sockets*, ambas não dispõe de recursos de controle de pacotes em alto nível. (Win08)

- ***OpenTNL:*** é multiplataforma e possui uma vasta gama de recursos para o controle de pacotes em alto nível, como serialização de objetos, gerenciamento de eventos de rede e compressão de dados. Possui ainda a grande vantagem de estar disponível sobre a licença GPL. Embora seja completa, sua utilização não é muito simples e sua documentação deixa a desejar. (Ope087)

- ***RakNet:*** possui todas as vantagens da *OpenTNL,* além de facilidade de utilização e boa documentação, com vários exemplos disponíveis. Possui apenas a pequena restrição de estar disponível gratuitamente somente para uso não comercial, o que não vem a ser um problema para o tipo de utilização a que se destinará (prototipagem e possível uso militar). Por conta de suas várias vantagens, foi a opção escolhida. (Rak08)

5.2.5 BIBLIOTECAS PARA O MÓDULO DE ÁUDIO

As bibliotecas de áudio têm como objetivo a geração de sons, a reprodução de músicas, além do controle de volume e outros efeitos sonoros como eco, reverberação e som posicional. Nenhum atributo especial é necessário para esse tipo de biblioteca, embora seja desejado algum tipo de licença gratuita.

Alguns exemplos considerados, todos multiplataforma:

- ***OpenAL:*** API de áudio 3D recomendada para o uso em jogos, têm sido extensivamente usada em vários projetos, além de ser muito bem documentada, possui diversos recursos para o uso de som posicional (3D). (Ope082)

- ***Audiere:*** é uma API de alto nível que permite a reprodução dos seguintes formatos de áudio: Ogg Vorbis, MP3, FLAC, WAV sem compressão, AIFF, MOD, S3M, XM, e arquivos IT. Para saída de áudio, *Audiere* suporta *DirectSound* ou *WinMM* no Windows, OSS no Linux e Cygwin, e *SGI AL* no IRIX. (Aud08)

- ***IrrKlang:*** é uma biblioteca de áudio voltada para desenvolvimento de jogos, que suporta os formatos WAV, MP3, Ogg Vorbis, MOD, XM, IT, S3M entre outros. É um motor para sons e música bastante versátil, pois possui desde várias funcionalidades básicas como tocar músicas em segundo plano, carregar um arquivo de MP3 e reproduzir, até recursos mais avançados, ao disponibilizar um sofisticado sistema de som *streaming*, modos simples e multitarefa, emulação de sons posicionais (3D), sistema de *plugin* e múltiplos modelos de atenuação entre outros. (irr08)

Este módulo não foi implementado no protótipo.

5.2.6 BIBLIOTECAS PARA O MÓDULO DE FÍSICA

As bibliotecas de física realizam cálculos físicos (de cinemática, dinâmica e estática) com base em parâmetros fornecidos para os objetos de um cenário virtual tridimensional, tais como massa, aceleração da gravidade, força e torque. O resultado é a simulação, próxima da realidade, da interação entre esses objetos e o cenário, com a sua conseqüente variação de velocidade, posição e orientação. Algumas bibliotecas realizam ainda cálculos relacionados com a deformação de objetos, baseados em parâmetros como tensão, elasticidade e dureza. Novamente, nenhum atributo especial é necessário para esse tipo de biblioteca, além de uma desejável licença gratuita.

Algumas bibliotecas que poderiam ser usadas nesse caso, todas multiplataforma:

- ***PhysX***: é um SDK (*Software Development Kit*) completo para física, que vem sendo cada vez mais utilizado, inclusive em projetos comerciais, como no motor para jogos *Unreal*. Existe inclusive uma placa de hardware dedicada (Ageia PhysX PCI) para realizar os cálculos dessa biblioteca. As placas de vídeo da *NVidia*, a partir da série 8, também possuem suporte nativo. É gratuita para usos não comerciais. (Phy08)

- ***ODE (Open Dynamics Engine)***: disponível sobre as licenças LGPL e BSD é a biblioteca mais empregada em conjunto com o motor gráfico *Ogre 3D*. É voltada para cálculos de alta performance de dinâmica de corpos rígidos, além de possuir suporte para uso de vários tipos de juntas e para detecção de colisões. (Ope08)

- ***Newton Game Dynamics***: uma das bibliotecas de física de uso mais simples, permite simulações de ambientes físicos com gerenciamento de cenas, detecção de colisões e comportamentos dinâmicos. Seu uso é gratuito, com algumas restrições. (New08)

Este módulo não foi implementado no protótipo.

5.2.7 BIBLIOTECAS PARA O MÓDULO DE INTELIGÊNCIA ARTIFICIAL

Ao contrário das bibliotecas vistas até agora, não existe nenhuma padronização quanto ao que uma biblioteca de IA deve conter. Existe apenas um consenso sobre alguns problemas que são comuns em jogos e que podem surgir também em um simulador de combates onde se encontrem avatares controlados pelo computador. Alguns desses problemas são a descoberta de caminhos, as tomadas de decisões comportamentais e as definições de estratégias. A maior parte dos jogos possui sua própria implementação de IA.

Algumas possibilidades, todas multiplataforma e de código aberto:

- ***OpenAI:*** é uma biblioteca que pretende ser para a IA o que o *OpenGL* é para os gráficos. Ela não se integra muito bem com o *DirectX,* embora possa ser utilizada com facilidade junto do *OpenGL*. Possui várias funcionalidades como agentes móveis, redes neurais, algoritmos genéticos e máquinas de estados finitos. Porém, não é atualizada desde 2003. (Ope081)

- ***OpenSteer:*** é uma biblioteca voltada para o controle de trajetória de agentes móveis de modo realístico, em jogos e animações. Ele emprega um *framework* baseado em *plugins*, que facilita a prototipagem e ajuste

de parâmetros dos agentes. Além de limitada, também não é atualizada desde 2004. (Ope086)

- ***OpenSkyNet:*** é uma biblioteca voltada para a descoberta de caminhos utilizando o algoritmo A* (a-estrela), além de resolver ações e possuir um sistema de aprendizado. Permite o ajuste de vários parâmetros. É regularmente atualizada. (Ope085)

Este módulo não foi implementado no protótipo.

5.3 ESTRUTURA DO PROTÓTIPO

Embora o protótipo desenvolvido não possua todas as funcionalidades de um sistema para ser empregado na prática, ele foi projetado levando em conta todos os módulos que deveriam compor um sistema real. Como o objetivo era comprovar a viabilidade e avaliar as vantagens da arquitetura proposta, foi suficiente codificar os módulos essenciais para o funcionamento do sistema, embora os módulos em aberto possam ser perfeitamente integrados futuramente ao protótipo. Os módulos não codificados possuem funções acessórias, que não influenciariam o funcionamento geral do sistema. São eles: os módulos de áudio, física e inteligência artificial. Os módulos de áudio e física apenas serviriam para aumentar a verossimilhança com um ambiente real, enquanto que o módulo de IA só seria utilizado caso se desejasse um treinamento de soldados contra “robôs”, ao invés de outros soldados humanos. A FIG 5.1 sumariza a estrutura de classes do protótipo para a arquitetura ASTROS.

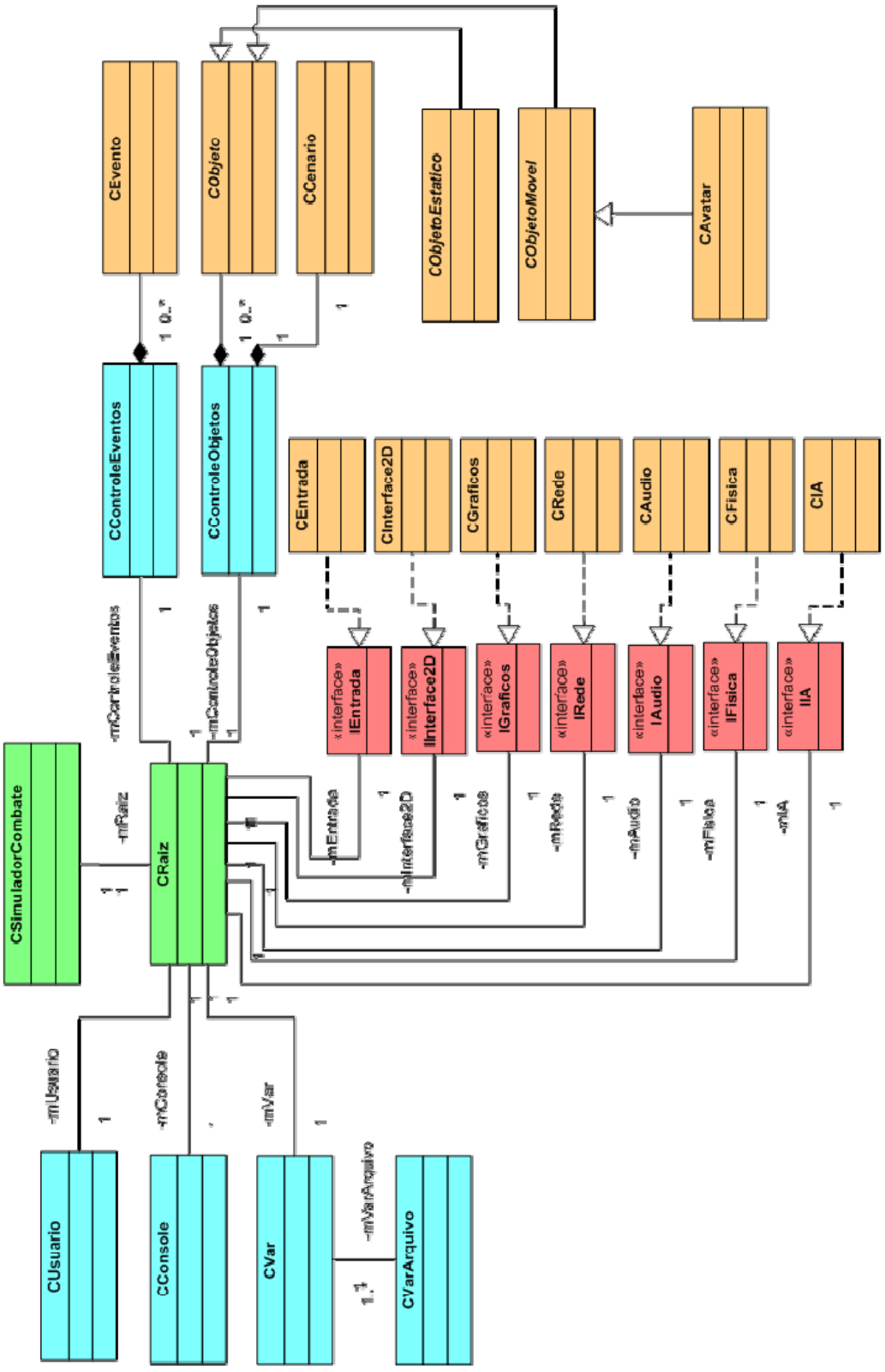


FIG.5.1 Diagrama de Classes

5.3.1 CLASSES NÚCLEO

A classe *CSimuladorCombate* é a classe de inicialização da aplicação, que possui apenas um método de inicialização e uma instância da classe CRaiz.

A classe *CRaiz* é a principal classe do sistema, sendo responsável por instanciar os módulos, além de manter a sincronia do sistema, executando um laço onde é chamado um procedimento de cada módulo. Além de uma instância de cada módulo, possui também uma instância de cada classe de controle (ver próxima seção). Possui um atributo *mStatus*, para o controle do status do sistema. Os seus métodos são *conFIGrSistema(), iniciarSimulacao()* e *processarEntrada().*

5.3.2 CLASSES DE CONTROLE

As classes *CUsuario* e *CConsole* não foram implementadas por não serem essenciais ao funcionamento do sistema. A primeira teria como objetivo manter informações específicas para cada usuário do sistema, como nome, posto, função e nível de privilégio (usuário comum ou administrador). Já a segunda seria utilizada para entrada de comandos do administrador, como criação de novos objetos, alteração de informações de usuários comuns durante uma simulação, configuração avançada de parâmetros do sistema, entre outros, através de linha de comando.

A classe *CVar* efetua a leitura e gravação das variáveis do sistema, que são os parâmetros de áudio, vídeo e configurações das teclas. Essas configurações são armazenadas em arquivos através da classe *CVarArquivo*, que possui uma instância para cada tipo de arquivo de configuração.

A classe *CControleEventos*, também não implementada, teria como função o controle dos objetos do tipo *CEvento*, que seria um tipo especial de objeto que não possui representação física, sendo empregado para a programação de

acontecimentos dentro da simulação, tais como recebimento de informações pelos avatares, alterações climáticas, desencadeamento de ações hostis ou aliadas, etc.

A classe *CControleObjetos* é a segunda classe mais importante, pois ela é responsável pelo acesso dos módulos lógicos aos objetos, atuando tanto como classe intermediadora como repositório dos objetos. O seu método mais importante é o *atualizarObjetos(),* que verifica a consistência da base de dados dos objetos, realizando atualizações e manutenções. O principal atributo é o *mListaObjetos*, que armazena de fato os dados relativos a todos os objetos da simulação. No protótipo esta lista é armazenada em memória, porém, numa aplicação de uso real ela poderia ser armazenada em arquivos ou mesmo em bancos de dados.

5.3.3 CLASSES INTERFACE

As classes *IEntrada, IInterface2D, IGraficos, IRede, IAudio, IFisica e IIA* são as classes virtuais que abstraem a implementação dos respectivos módulo lógicos, que se encontra nas classes modulares (ver próxima seção). Como são classes puramente virtuais, todos seus métodos também o são. Os módulos são instanciados pela classe *CRaiz* através das classes interface.

5.3.4 CLASSES MODULARES E DE OBJETOS

As classes C*Entrada, CInterface2D, CGraficos, CRede, CAudio, CFisica e CIA* possuem a implementação de todos os métodos e atributos que manipulam os dados dos respectivos módulos, empregando as bibliotecas já descritas na seção 5.2. O acesso aos dados dos objetos é realizado por intermédio das classes *CRaiz* e *CControleObjetos*.

A classe *CObjeto* é uma generalização para as classe *CObjetoEstatico* e *CObjetoMovel*. A primeira é utilizada para representar objetos virtuais que não estão

sujeitos à mudança de posição, orientação ou escala (árvores, prédios, etc.). Já a segunda representa os demais objetos, tais como veículos, armamentos, projéteis, soldados, etc. Cada tipo de objeto deveria ter uma classe derivada própria, porém no protótipo a única classe que foi derivada foi a *CAvatar*, que representa tanto os soldados aliados quanto inimigos.

5.4 CONCLUSÃO DO CAPÍTULO

Neste capítulo foi descrito um protótipo elaborado com o objetivo de testar a arquitetura proposta. Porém, além de um simples teste, este protótipo serve também como um guia com respeito às ferramentas e opções de projeto que podem ser seguidas no desenvolvimento de um sistema real de simulação de combates táticos. E mais, um sistema real pode de fato aproveitar o protótipo aqui desenvolvido, estendendo, ampliando e reutilizando os componentes já implementados.

6 CONCLUSÕES

Neste capítulo são apresentados os comentários finais referentes a este trabalho, outras hipóteses que poderiam ser consideradas durante o projeto da arquitetura, bem como sugestões para trabalhos futuros.

6.1 COMENTÁRIOS GERAIS

A arquitetura aqui proposta tem como objetivo principal o estímulo ao desenvolvimento nacional de sistemas de simulação de combate. De modo algum ela pretende superar os jogos e simuladores disponíveis comercialmente. Os produtos comerciais possuem equipes dedicadas e orçamentos tais que o produto final atinge quase o estado da arte. Porém, é necessário que haja um incentivo à pesquisa e ao desenvolvimento deste tipo de tecnologia nas Forças Armadas brasileiras. A arquitetura ASTROS então surge como um precursor para aqueles que pretendem iniciar pesquisas nessa área, sugerindo técnicas e metodologias que facilitarão o desenvolvimento de simuladores táticos militares.

6.2 DISCUSSÃO

A arquitetura ASTROS foi projetada com base em um conjunto de premissas (ou decisões de projeto) que não são necessariamente as melhores e nem tampouco definitivas. É interessante que sejam avaliadas outras possibilidades para algumas das escolhas realizadas.

Uma primeira questão diz respeito à escolha do estilo arquitetural. O estilo selecionado foi o centrado em dados. Porém existe margem para um possível uso conjunto ou com a arquitetura em camadas ou com a arquitetura de sistema de sistemas. No primeiro caso, devem-se considerar as vantagens de se criar outro nível de abstração, onde os módulos lógicos se situam numa camada tecnológica, o

controle e lógica do sistema se encontram numa camada de gerenciamento e os dados se encontram numa camada exclusiva de dados.

No segundo caso, tem-se a possibilidade de se determinar uma maior independência dos módulos lógicos, de modo que estes atuem como unidades de processamento de dados independentes. Assim, eles operariam utilizando uma comunicação por troca de mensagens ao invés de chamadas de procedimento, eliminando a necessidade de criação de interfaces abstratas. Isso poderia provocar uma sobrecarga de comunicação, ao se tornar necessário definir um protocolo de mensagens, porém poderia compensar ao permitir a separação dos módulos lógicos a nível de diferentes processos, que poderiam ser executados inclusive em máquinas separadas. Um exemplo útil desse tipo de funcionalidade seria no caso de um processamento de IA muito intenso (caso de simulações de ambientes complexos, com muitos avatares robôs), esta tarefa poderia ser atribuída a um sistema de processamento dedicado.

Outra questão que vale a pena ser considerada diz respeito à localização do repositório de dados. A escolha do tipo de sincronização de dados foi tomada considerando-se que os dados se encontram na mesma máquina onde são executadas as demais operações dos outros módulos lógicos. Porém, seria possível considerar-se também um caso onde o servidor também detém o armazenamento e controle de acesso aos dados utilizados para a simulação. Uma justificativa para esse tipo de consideração seria no caso de um cenário muito extenso e com dados sujeitos a constantes modificações externas. Nessa situação o módulo de rede atuaria também como parte do controle de acesso aos dados, tornando necessário o uso do método de sincronização quadro-negro, onde o repositório de dados informaria ativamente sobre alterações nos dados. Ainda assim, por questões de desempenho, o sistema local deveria manter uma cópia parcial dos dados que são relevantes para o processamento de informações do cenário, nas imediações da localização do avatar local.

6.3 TRABALHOS FUTUROS

- Implementação de um protótipo completo, que contenha todos os módulos codificados. Para tanto, no capítulo 5 já existem algumas sugestões de tecnologias que podem auxiliar nesse trabalho.

- Adaptação e extensão da arquitetura ASTROS para os outros tipos de jogos de guerra já existentes no Exército Brasileiro. Isto é possível, pois o conceito central da arquitetura e seus atributos, como modularidade e reuso, enriqueceriam em muito outros projetos que utilizam paradigmas estanques de programação, que os tornam rapidamente obsoletos, dados a dificuldade de manutenção, expansão e evolução deles.

- Adaptação da arquitetura ASTROS de modo que ela possa ser empregada em conjunto com os jogos de guerra estratégicos já utilizados pelo Exército Brasileiro. Para tanto, deve-se proceder com alterações no sistema de sincronização remota, pois o servidor passará a ser outro sistema executado a nível estratégico. Deve-se levar em conta a possibilidade da geração de eventos únicos do sistema servidor, que reflitam ações de comando de alto nível, como por exemplo, ordens de deslocamento e fornecimento de reforços.

- Realização de testes comparativos de desempenho, empregando diferentes tecnologias (diferentes bibliotecas) para cada um dos módulos lógicos.

- Compilação e testes de uma mesma versão de um protótipo em diferentes plataformas (sistemas operacionais), de modo a validar a portabilidade da arquitetura.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ABOWD G D, ALLEN R e GARLAN D **Formalizing style to understand descriptions of software architecture** [Conferência] // ACM Transactions on Software Engineering and Methodology. - 1995. - Vol. 4.

America´s Army [Online]. - 24 de Novembro de 2008. - http://www.americasarmy.com/.

America´s Army Platform [Online]. - 24 de Novembro de 2008. - http://info.americasarmy.com/home.php.

Audiere [Online]. - 24 de Novembro de 2008. - http://audiere.sourceforge.net/.

BASS L, CLEMENTS P e KAZMAN R **Software Architecture in Practice** [Livro]. - [s.l.] : Adison Wesley, 2003.

BECK K e JOHNSON R **Patterns Generate Architectures** [Conferência] // ECOOP. - 1994.

BERARD E **Essays in Object-Oriented Software Engineering** [Livro]. - [s.l.] : Prentice Hall, 1992.

CAFFREY M **History of Wargames: Toward a History Based Doctrine for Wargaming as of 6 Jan 2000** [Online]. - 2000. - 24 de Novembro de 2008. - http://www.strategypage.com/wargames/articles/wargame_articles_2004980.asp.

CRAWFORD C **The Art of Computer Game Design** [Livro]. - [s.l.] : Whashington State University, 1997.

Crazy Eddie Graphic User Interface [Online]. - 24 de Novembro de 2008. - http://www.cegui.org.uk/.

Crystal Space [Online]. - 24 de Novembro de 2008. - http://www.crystalspace3d.org/main/Main_Page.

DARPA **DARWARS** [Online]. - 2003. - 24 de Novembro de 2008. - http://www.darwars.org/.

DirectInput [Online]. - 24 de Novembro de 2008. - http://msdn.microsoft.com/en-us/library/bb219802(VS.85).aspx.

DUNNIGAN J F **The Complete Wargames Handbook** [Livro]. - [s.l.] : Quill, 1992.

ESTADO MAIOR DO EXÉRCITO **Glossário de Termos e Expressões para Uso no Exército (Manual C 20-1)** [Livro]. - DF : [s.n.], 2003.

FLYNT J P e SALEM O **Software Engineering for Game Designers** [Livro]. - [s.l.] : Premier Press, 2005.

GAMMA E [et al.] **Design Patterns: Elements of Reusable Object-Oriented Software** [Livro]. - [s.l.] : Addison-Wesley, 1995.

GONZALEZ L **Spot On: The US Army's There-based simulation** [Online] // Gamespot. - 21 de Abril de 2004. - 24 de Novembro de 2008. - http://www.gamespot.com/news/2004/04/21/news_6093860.html.

HADWIGER M **Design and Architecture of a Portable and Extensible Multiplayer 3D Game Engine** // Tese de Mestrado. - [s.l.] : Vienna University of Technology, 2000.

irrKlang [Online]. - 24 de Novembro de 2008. - http://ambiera.com/irrklang.

irrLicht [Online]. - 24 de Novembro de 2008. - http://irrlicht.sourceforge.net.

KIRBY M W **Operational Research in War and Peace** [Livro]. - [s.l.] : Imperial College Press, 2003.

LENOIR T e LOWOOD H E **Theaters of War: The Military-Entertainment Complex** [Artigo] // Collection - Laboratory - Theater: Scenes of Knowledge in the 17th Century. - [s.l.] : Walter de Gruyter, 2005.

LENOIR T **Fashioning the Military Entertainment Complex** [Artigo] // Correspondence: An International Review of Culture and Society. - 2003. - Vol. 10.

MEYER B **Object-Oriented Software Construction** [Livro]. - [s.l.] : ISE Inc., 1997.

MICHAEL D e CHEN S **Serious Games - Games that Educate, Train and Inform** [Livro]. - [s.l.] : Thomsom Course Technology PTR, 2006.

MyGUI [Online]. - 24 de Novembro de 2008. - http://www.ogre3d.org/wiki/index.php/MyGUI.

NATO Research & Technology Organisation [Online]. - 24 de Novembro de 2008. - http://www.rta.nato.int/.

Newton Game Dynamics [Online]. - 24 de Novembro de 2008. - http://www.newtondynamics.com.

Object Oriented Input System [Online]. - 24 de Novembro de 2008. - http://sourceforge.net/projects/wgois.

Ogre 3D [Online]. - 24 de Novembro de 2008. - htp://www.ogre3d.org.

Open Dynamics Engine [Online]. - 24 de Novembro de 2008. - http://ode.org.

Open GUI [Online]. - 24 de Novembro de 2008. - http://opengui.rightbracket.com/index.php.

Open Input [Online]. - 24 de Novembro de 2008. - http://home.gna.org/openinput.

Open TNL [Online]. - 24 de Novembro de 2008. - http://www.opentnl.org.

OpenAI [Online]. - 24 de Novembro de 2008. - http://openai.sourceforge.net.

OpenAL [Online]. - 24 de Novembro de 2008. - http://www.openal.org.

OpenSkyNet [Online]. - 24 de Novembro de 2008. - http://openskynet.sourceforge.net.

OpenSteer [Online]. - 24 de Novembro de 2008. - http://opensteer.sourceforge.net.

PERLA P **The Art of Wargaming** [Livro]. - [s.l.] : Naval Institute Press, 1990.

PERRY D E e WOLF A L **Foundations for the study of software architecture** [Conferência] // ACM SIGSOFT Software Engineering Notes. - 1992. - Vol. 17.

PhysX [Online]. - 24 de Novembro de 2008. - http://www.nvidia.com/object/nvidia_physx.html.

PLUMMER J **A Flexible and Expandable Architecture for Computer Games** // Tese de Mestrado. - [s.l.] : Arizona State University, 2004.

PRESSMAN R S **Software Engineering: A Practitioner's Approach** [Livro]. - [s.l.] : McGraw-Hill, 2005.

RakNet [Online]. - 24 de Novembro de 2008. - http://freshmeat.net/projects/raknet.

RIBEIRO A J **O emprego de jogos comerciais nas atividades de simulação de combate: uma proposta** // Tese de Mestrado. - [s.l.] : Escola de Comando e Estado Maior do Exército, 2005.

ROBAR J **Multiplayer Technology: A Primer** [Artigo] // Military Simulation and Training. - 2004. - Vol. 6.

ROBEL M K **The Difference Between Military & Civilian Wargames** [Online]. - 18 de Junho de 2004. - 24 de Novembro de 2008. - http://www.strategypage.com/wargames/articles/wargame_articles_2004919231.asp.

ROCHESTER University of **Action Video Games Sharpen Vision 20 Percent** [Online]. - 6 de Fevereiro de 2007. - 24 de Novembro de 2008. - http://www.sciencedaily.com/releases/2007/02/070206100601.htm.

ROLLINGS A e MORRIS D **Game Architecture and Desgin: A New Edition** [Livro]. - [s.l.] : New Riders, 2004.

RUCKER R **Software Engineering and Computer Games** [Livro]. - [s.l.] : Adison-Wesley, 2002.

Simple DirectMedia Layer [Online]. - 24 de Novembro de 2008. - http://www.libsdl.org.

SOMMERVILLE I **Software Engineering** [Livro]. - [s.l.] : Adison-Wesley, 2000.

Tactical Language [Online]. - 24 de Novembro de 2008. - http://www.tacticallanguage.com/.

VBS2 [Online]. - 24 de Novembro de 2008. - http://virtualbattlespace.vbs2.com/.

Windows Sockets API [Online]. - 24 de Novembro de 2008. - http://msdn.microsoft.com/en-us/library/ms740673.aspx..

8 APÊNDICE

8.1 APÊNDICE I: ILUSTRAÇÕES DO PROTÓTIPO

FIG.8.1 Tela inicial do simulador de combate

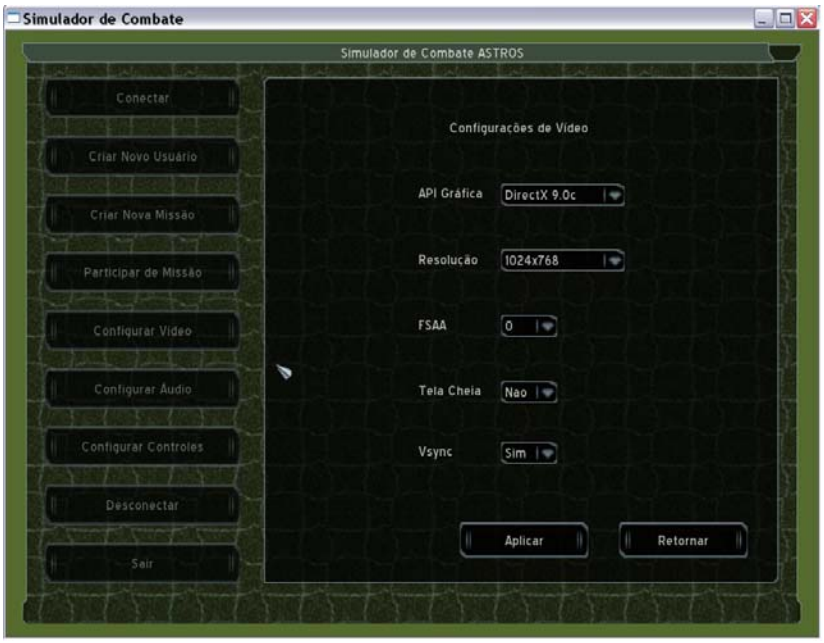


FIG.8.2 Tela da configuração de vídeo

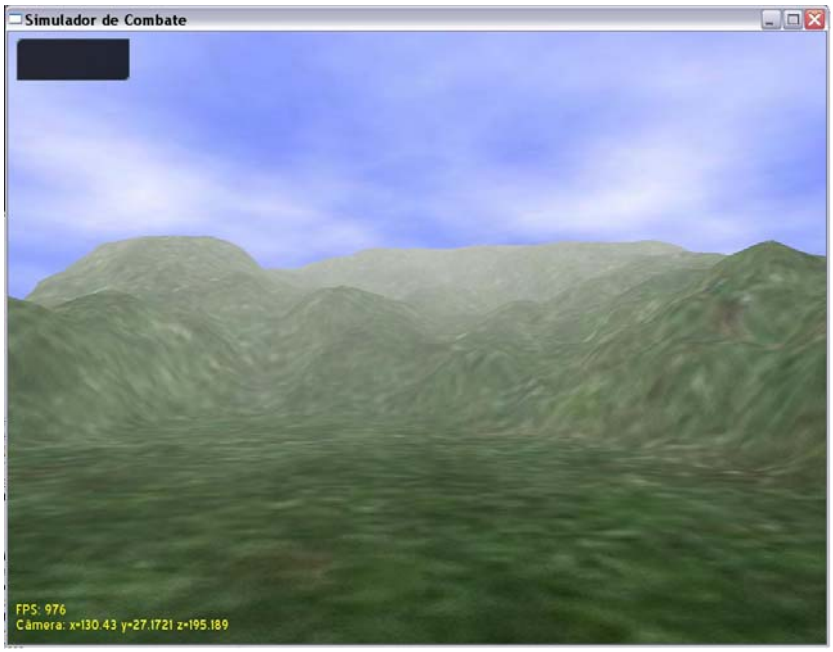


FIG.8.3 Tela de um cenário da simulação

FIG.8.4 Tela do menu interno

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo