Arquiteturas para Jogos *Massive Multiplayer*

Lauro Eduardo Kozovits
Computer Science Department, PUC-Rio
lauro@inf.puc-rio.br

Bruno Feijó
Computer Science Department, PUC-Rio
bruno@inf.puc-rio.br

PUC-RioInf.MCC36/03 October, 2003

Abstract: Multiplayer game technology is a specialized field in networked virtual environments (Net-VE). With the increasing popularity of Internet games, an analysis of the related problems and solutions of these environments became interesting in order to adapt them to the specific needs of these games. An hybrid architecture using both client-server and peer-to-peer aproach is proposed and implemented in this work for a certain class of multiplayer games.

Keywords: Distributed virtual environment, multiplayer games, client/server design, peer-to-peer design, latency, bandwidth, scalability

Resumo: A tecnologia de jogos multi-jogador é uma especialização dentro da área de ambientes virtuais distribuídos. Com a crescente popularidade de jogos pela Internet, torna-se interessante analisar os principais problemas e soluções de arquiteturas destes ambientes, a fim de adaptá-las para as necessidades específicas destes jogos. Uma arquitetura híbrida usando as abordagens cliente-servidor e *peer-to-peer* é proposta e implementada neste trabalho, visando atender uma certa classe de jogos multi-jogador.

Palavras-chave: Ambientes virtuais distribuídos, jogos multi-jogador, arquitetura cliente-servidor, arquitetura peer-to-peer, latência, largura de banda, escalabilidade

---

This work is being sponsored by CNPq and ICAD.

1. Introdução

Muitos trabalhos têm sido publicados nos últimos anos, com o objetivo de aperfeiçoar os ambientes virtuais distribuídos existentes ou mesmo propor novos ambientes. Nestes ambientes, múltiplos usuários podem interagir num mundo compartilhado conectado em rede. Cada usuário tem a sua própria visão deste mundo simulado e todas as alterações de estado do ambiente são, ou idealmente deveriam ser, distribuídas entre os demais participantes.

Estes sistemas podem ser muito úteis para treinamento militar, pesquisa médica, simulações de engenharia etc. Contudo, uma aplicação muito popular para ambientes virtuais distribuídos está na área de jogos *online* multi-jogador. É um fato o grande interesse de um número cada vez maior de pessoas em investir seu tempo e dinheiro para viver uma segunda vida numa realidade alternativa. É fácil observar a proliferação mundial de *websites* dedicados ao tema.

Alguns exemplos de sucesso são os jogos Ultima Online (2003) da Origin-Electronic Arts (vide figura 1), Everquest (2003) da Sony e ToonTown (2003) da Disney.

figura 1: tela capturada do jogo Ultima Online (2003).

Estes jogos *online* não mais estão restritos a uma faixa etária ou categoria específica de público e, por isso, têm despertado a atenção de um bom número de grandes empresas como Microsoft, Disney, Sony, Electronic Arts, etc. Também no meio acadêmico, é possível observar um aumento do número de trabalhos dentro deste tópico de ambientes virtuais distribuídos.

Muito se tem feito para melhorar a sensação imersiva por meio do aumento do realismo gráfico e efeitos sonoros. Cuidados especiais devem, no entanto, ser tomados para usar de modo eficiente os recursos de rede na distribuição do estado corrente do mundo virtual entre os jogadores.

Os atuais PCs já dispõem de sofisticados recursos gráficos que outrora só estavam disponíveis em supercomputadores. Nos aspectos de rede contudo, problemas de atraso no recebimento de mensagens ou limitações na largura de banda são inerentes a Internet atual, que é a rede disponível em larga escala.

Outra questão importante é diferenciar os objetivos e os requisitos de ambientes virtuais distribuídos usados em simulações profissionais daqueles encontrados em jogos multi-jogador: enquanto nos primeiros o objetivo é ser realista, tentando simular, a realidade física e visual, no último, o principal objetivo é o entretenimento, ou numa expressão mais comumente empregada: "obter boa jogabilidade".

Além disso, como jogos são aplicações destinadas ao entretenimento em massa, temos que levar em conta as seguintes características:

· assume-se que os jogadores estão conectados em redes de baixa velocidade ou mesmo conectados à Internet utilizando modems analógicos com uma largura de banda entre 28.8 e 56kbps. Embora o número de usuários de banda larga têm aumentado muito, é importante assumir velocidades baixas como um denominador comum na tarefa de projeto e desenvolvimento para compensar problemas de atraso intermitentes comuns na Internet que é a rede utilizada no caso de ambientes com número elevado de jogadores (*massive multiplayer*).

· nem sempre os recursos computacionais disponíveis representam o estado da arte seja no jogador seja no *game service provider*; é importante levar em conta os custos para os usuários e os investimentos a serem realizados pelo empreendedor.

· a exploração de uma limitação tecnológica, se justificada no contexto do jogo, pode ocorrer sem prejuízo para o usuário, constituindo, às vezes e ironicamente, um "defeito especial". Por exemplo, uma demora de conexão pode ser "maquiada" com um efeito de materialização progressiva do jogador.

Se, na área de ambientes virtuais distribuídos há um grande número de trabalhos publicados, nesta área de jogos especificamente, o número é relativamente discreto. Assim sendo, cabe apresentaremos uma visão geral de arquiteturas de jogos. Finalmente, levando-se em conta o que foi mencionado acima, o presente trabalho irá propor uma arquitetura alternativa para certas classes de jogos multi-jogador.

2. Trabalhos relacionados

Nesta seção são analisadas algumas arquiteturas comumente encontradas e são feitas críticas de modo a justificar a proposta deste trabalho

2.1 Sistemas Peer-to-Peer (P2P) usando unicast

A idéia de desenvolver um ambiente virtual distribuído utilizando comunicação *peer-to-peer* (figura 2) para distribuir o estado entre os clientes é algo simples, de fácil implementação e, portanto, muito utilizada. Entretanto é uma abordagem, a princípio, ingênua e limitada.

É fácil perceber que se há N estações de trabalho participantes da simulação, cada mudança de estado, por exemplo, uma mudança de posição ou de velocidade de um avatar (objeto representando um jogador) tem que ser comunicada as N-1 estações de trabalho participantes da simulação.

A limitação citada refere-se ao problema de escalabilidade, uma vez que o esforço requerido é $O(N^2)$ e com o aumento de jogadores numa instância da simulação, rapidamente atingiríamos um colapso na comunicação entre os jogadores.

A citada facilidade de implementação e a ausência de servidor central fazem com que esta seja a arquitetura mais comum em jogos no varejo (simuladores de vôo, *first person shooters*, simulação de esportes etc). Observando a especificação na própria caixa destes jogos pode-se conferir a limitação do número máximo de jogadores simultâneos (8, 16 ou 32 jogadores por sessão são valores típicos).

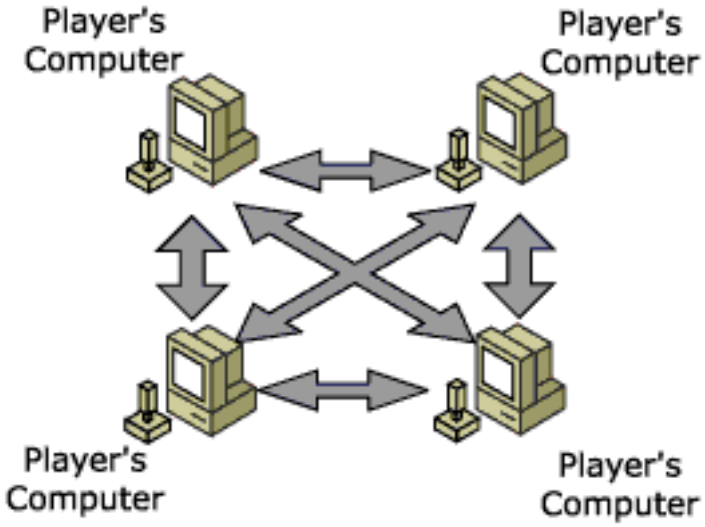

figura 2: arquitetura peer-to-peer típica

2.2 Sistemas usando *broadcast* e *multicast*

Pelo fato de os sistemas *peer-to-peer unicast* não exibirem uma boa escalabilidade, muitos outros tipos de sistemas foram projetados. Uma das técnicas mais comuns é o emprego de *broadcast* de mensagens. Provavelmente o mais conhecido destes sistemas é o SIMNET (Calvin et al., 1993), mantido pela DARPA (Defense Advanced Research Projects Agency). Ele foi projetado para simulações de treinamento militar e é baseado no protocolo padrão IEEE DIS.

Uma arquitetura *broadcast* é basicamente ainda uma arquitetura *peer-to-peer* mas, ao invés de ocorrer o envio de mensagens para cada participante (*unicast*), apenas uma única mensagem *broadcast* será enviada a cada atualização de cada participante. Isto reduz o número de mensagens requeridas diminuindo a complexidade para O(N). Contudo, numa simulação onde estão envolvidos centenas de participantes, há uma sobrecarga óbvia, pois cada participante terá que receber e processar cada atualização.

Na prática, isto não deveria ser necessário, pois nem todo participante está diretamente interagindo com todos os demais, ocorrendo, portanto, um excesso de mensagens desnecessárias. Jogadores que estão espacialmente separados, seja por longas distâncias, seja por muros ou níveis em uma edificação estão, nesta estratégia *broadcast*, enviando mensagens para outros que sequer conseguem visualizar. A taxa de atualizações entretanto, tem que ser mantida, pois se reduzida, irá impactar a fluidez do movimento de jogadores próximos e visíveis.

Algo importante quando consideramos um projeto de um jogo multi-jogador é que não é possível utilizar *broadcast* na Internet.

Um caminho natural de ambientes virtuais distribuídos é, portanto, buscar modos de se livrar de atualizações desnecessárias. A extensão natural para sistemas utilizando *broadcast* é tentar remediar o problema citado por meio de comunicação *multicast*.

Usando este tipo de comunicação, cada participante da simulação deve fazer parte de um grupo *multicast* e irá somente receber mensagens destinadas a esse grupo. Pelo particionamento lógico do espaço virtual, o número de mensagens de rede transmitidas pode ser muito reduzido.

NPSNET (Macedonia et al., 1994) (Macedonia et al., 1995) (Macedonia et al., 1995b), VERN (Blau et al., 1992) e (Broll, 1997) são exemplos de sistemas virtuais *multicast* projetados para simulações militares.

Pelo fato de *multicasting* não ser ainda disponível em redes para consumo em massa, não há jogos multi-jogador usando esta técnica. Entretanto, alguns jogos *peer-to-peer* utilizam *broadcast* em certas situações (ex: para encontrar outros clientes numa rede local).

2.3 Sistemas cliente-servidor

Certamente a comunicação cliente-servidor (figura 3) constitui a maneira mais versátil de distribuir atualizações para um grande número de usuários. A idéia básica é que cada cliente envia sua mensagem de atualização apenas para o servidor, que terá então a responsabilidade de encaminhar a informação para os demais clientes.

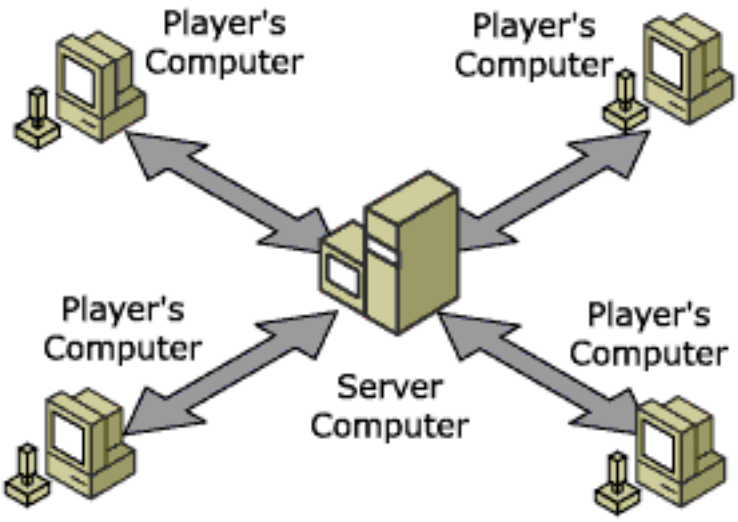

figura 3: arquitetura cliente-servidor típica

Por receber atualização de todos os clientes, o servidor terá uma "visão" completa e permanentemente atualizada, do mundo virtual e tem a responsabilidade de reenviá-la a todos os seus clientes.

Sendo um programa e não um simples roteador, é possível utilizar regras sobre que clientes atualizar e quando seria o momento adequado para fazê-lo de modo a minimizar o tráfego na rede e manter as estações (clientes) o mais atualizadas quanto possível. Por exemplo, um servidor pode

não enviar mensagens de um cliente A referentes a um cliente B (e vice-versa) caso estes clientes estejam em compartimentos fechados e distintos.

Da mesma forma, clientes situados a grandes distâncias um do outro não precisam ter atualizações (referentes ao outro) com tanta freqüência, pois uma pequena alteração de posição do cliente X pode não fazer nenhuma diferença visual para o cliente Y.

Adicionalmente, se o ambiente virtual for espacialmente particionado, múltiplos servidores podem ser usados, cada qual cuidando de clientes de uma parte do mundo. Esta técnica pode aumentar a escalabilidade destes sistemas ainda mais.

Como exemplos de sistemas distribuídos cujos projetos visam uma alta escalabilidade podemos citar RING (Funkhouser, 1995), (Funkhouser, 1996) e NetEffect (Das et al., 1997). Estes sistemas, de acordo com a descrição, podem suportar simulações para centenas ou mesmo milhares de usuários simultâneos. RING é um exemplo de sistema que usa algoritmos de visibilidade para filtrar atualizações de estado entre clientes invisíveis um para o outro.

3. Problemas e soluções típicas adotadas em jogos multiplayer baseados na Internet

Neste item, analisamos problemas e algumas soluções ligados a implementação de jogos baseados na Internet. Estas questões, em adição ao conhecimento das características das arquiteturas descritas no item anterior, permitem justificar a adoção de nossa proposta de arquitetura.

3.1 Inicialização

Um problema crucial de jogos multi-jogador, é semelhante ao fato de que o sucesso de muitos empreendimentos depende da atração de público ou de clientes portanto. Quanto mais clientes são conquistados, mais sucesso e maior popularidade adquire o empreendimento num círculo virtuoso. Em contrapartida, quanto menos clientes ou quanto mais se perde clientes, maiores as chances de insucesso formando um círculo que, ao contrário do anterior, é vicioso.

Este problema, cujos bons exemplos na área de informática são jogos multi-jogador e sistemas de salas de chat na Internet, possivelmente tem implicações ligadas a psicologia humana que não serão tratadas neste trabalho.

No entanto, a facilidade de atrair jogadores e estabelecer a comunicação entre eles é um fator de sucesso do software produto, principalmente levando-se em conta o que foi dito na introdução, referente ao principal objetivo desta sub-área de ambientes virtuais distribuídos que é o entretenimento.

Assim sendo, entregar totalmente ao usuário candidato a jogador a tarefa de encontrar outros jogadores e estabelecer procedimentos técnicos (definir endereços IP, portas, inicializar servidor, cliente ou *peer*) é algo, no mínimo, inadequado do ponto de vista de boa interface com o usuário.

Atualmente, existem duas abordagens ordinariamente usadas: a de jogos mono-usuário que oferecem ao jogador a opção e a responsabilidade organizarem uma sessão do jogo numa arquitetura *peer-to-peer* com poucos jogadores (até 8, 16 ou 32 em geral) e a de jogos *massive multiplayer* usando arquitetura cliente-servidor, cuja conexão com o servidor é algo embutido no código da aplicação.

Jogos mais modernos, costumam optar pelo uso de um *masterserver* que é uma forma semi-automática de localizar servidores na Internet, cabendo ao jogador a escolha de se conectar ou não às sessões oferecidas.

A presença de algum tipo de elemento sincronizador é fundamental para definição de regras da sessão, eventual atualização de software e *download* de alguma geometria comum a ser usada por todos os jogadores.

A respeito disto, cabe aqui apontar um problema adicional de compartilhamento de objetos que ocorre quando algum item é criado ou modificado durante a simulação (Kozovits, 2003).

O objetivo, portanto, de um *masterserver* é prover uma lista de servidores ativos. Quando um servidor é ativado ou se desconecta, deve comunicar ao *masterserver* sua ação a fim de manter a lista atualizada. Cada servidor informa também ao *masterserver* o número de jogadores ativos e o número máximo permitido. A interface usual do *masterserver* é a criação de páginas web dinâmicas para permitir que o jogador escolha com que servidor ele deseja se conectar.

3.2 Problemas de rede

Num ambiente virtual distribuído ideal, o estado da simulação deveria ser exatamente o mesmo em cada estação de trabalho. Isto significa que todas as entidades que fazem parte do mundo virtual deveriam estar no mesmo estado e na mesma posição exibindo as mesmas propriedades não importa qual estação esteja sendo observada.

Para garantir esta consistência existe o que se chama de simulação no lado do servidor (*server-side simulation*) ou seja, um computador executando um programa servidor se encarrega de controlar toda a simulação e repassá-la aos demais. Para alcançar este objetivo, uma enorme quantidade de dados deveria ser transmitida entre os participantes e o servidor.

Dado que, devemos considerar as limitações de um ambiente típico de jogos multi-jogador, é comum encontrarmos o que se chama simulação no lado do cliente (*client-side simulation*), ou seja, transmitir informações básicas e deixar a cargo do cliente, os cálculos simples de atualização. Por exemplo, se uma bala de canhão for disparada, basta enviarmos esta informação (do disparo) que cada cliente deverá ser responsável por calcular a nova posição do projétil após um certo tempo.

Naturalmente, isto é fácil de se fazer apenas para objetos com dinâmica simples, sem intervenção do usuário. Discrepâncias entre o que se vê em uma ou outra estação podem ser bem visíveis e são causadas pelos problemas de latência da rede e de perda de pacotes. Este problema de latência é maior na simulação efetuada no lado do servidor.

No entanto, problemas ligados a fraudes (*cheating*) e arbitragem de resultados dificultam a adoção geral de simulação no lado do cliente. Estes problemas serão abordados a seguir.

3.2.1 Latência

Latência pode ser definida formalmente como o tempo que um pacote infinitesimal de dados leva para para viajar entre sua origem e seu destino (Yu-Shen, 1997). Um dos principais problemas encontrados principalmente nos jogos em tempo real e em rede está relacionado ao conceito de latência.

Dado que temos que assumir que um número elevado de jogadores estão conectados utilizando modems analógicos, o problema de latência acentua-se ainda mais. Isto dá-se pelo fato de estes modems utilizarem esquemas de compressão e correção de erros que adicionam atrasos na transmissão de pacotes UDP.

Jogos utilizando simulação no lado do servidor são mais afetados pela latência pelo fato de o servidor, responsável por todos os cálculos e manutenção de estados, ter que enviar cada posição e estado a todos os participantes da simulação. Isto significa que o jogador, neste tipo de simulação, ficará mais propenso a atrasos (*lags*) que no caso de simulação no lado do cliente.

Estes atrasos, principalmente em jogos de ação, irritam o jogador que perde a sensação imersiva e, por vezes, pontos, caso não possa tomar uma decisão durante a fração de tempo em que o atraso está se manifestando. Simulação feita inteiramente no lado do servidor pode, inclusive, gerar a sensação de que o próprio avatar é "lento", pois a resposta a um comando é sempre validada e retransmitida pelo servidor.

Mesmo em jogos com simulação no lado cliente, é possível que um jogador defina sua mira, lance um míssil, ação esta imediatamente visível localmente e, mesmo assim, não atinja o oponente

pelo fato de, devido a atrasos de transmissão, a posição mirada não corresponder a posição "real" do oponente num dado instante.

Embora jogos com alta latência sejam ruins, pode ser até mesmo pior a flutuação desta latência. A variação da latência deve ser baixa ou, de outra forma, o jogador terá dificuldades em se acostumar com esta intermitência. O cérebro humano tem a capacidade de se adaptar a um moderado atraso entre a ação e a consequência apenas se o tempo de atraso for constante (Blow, 1998).

Uma regra de projeto de jogos multi-jogador é não acumular mudanças de estado e sim enviá-los no momento em que ocorrem sem usar *buffers*. Fazendo isto, pelo menos ficamos limitados a atrasos gerados apenas pela rede.

3.2.2 Perda de pacotes

Um outro problema também importante, principalmente se considerarmos conexões utilizando modems é a perda de pacotes. Pacotes podem ser perdidos em diversos estágios da transmissão.

Como, em geral, utiliza-se UDP para transmitir atualização de estado por sua maior velocidade, temos que considerar o problema de perda de pacotes, pois a comunicação por UDP não é confiável.

Assim, é possível ocorrer, numa implementação sem maior cuidado, ausência de suavidade no movimento, como por exemplo, movimento em pulos dos personagens ou mesmo o "teletransporte" de personagens ou outros componentes dinâmicos da simulação entre pontos não muito próximos.

Como este problema é inerente a tecnologia usada correntemente na Internet, a solução está no uso de *dead-reckoning* que veremos ainda.

3.2.3 Largura de banda

Largura de banda (*network bandwidth*) é a quantidade de dados que a rede pode transferir numa unidade de tempo, representa, portanto, a taxa de fluxo de dados ou o *throughput* da rede.

Considerando que uma boa parte dos usuários da Internet se conectam usando modems de 56,6 kbps, a largura de banda para envio de pacotes para o servidor (ou outros jogadores) é muito limitada.

Vamos analisar uma situação ideal desconsiderando os custos adicionais das camadas dos protocolos UDP e PPP. Nesta situação, seremos capazes de transmitir 56600/10 = 5660 bytes (consideramos 10 para incluir *start* e *stop* bits usados em comunicação serial). Se consideramos atualizações de estado numa taxa de 30Hz (uma atualização a cada *frame* numa animação a 30fps), teremos apenas 5660/30 = 189 bytes possíveis de ser enviados ou recebidos por *frame*.

Se transmitirmos apenas a informação de posição e velocidade vetoriais e definirmos 1 byte para representar o tipo de mensagem ou estado e, se utilizarmos 4 bytes por número real, teremos então 189/(3*4 bytes da posição + 3*4bytes da velocidade + 1tipo de mensagem) = 7,5 pacotes possíveis de serem trocados por frame e por jogador.

Assim sendo, um jogador parado pode receber, no máximo e em condições ideais, a informação de atualização de outros sete jogadores a cada *frame*. Importante lembrar que os modems já realizam compressão de dados.

É evidente portanto que, para lidar com estas limitações de largura de banda temos apenas duas alternativas: diminuir o tamanho do pacote ou a reduzir a taxa de envio destes pacotes. A última alternativa é freqüentemente a escolhida. A tecnologia *dead-reckoning* (Das et al., 1997) (Macedonia et al., 1995b) (Macedonia et al., 1994) também ajuda a reduzir este problema de taxa de atualização.

3.3 *Dead-Reckoning*

O termo dead-reckoning (abreviação de *deduced reckoning*) tem suas origens em uma técnica de navegação utilizada até mesmo por Colombo, que consiste em marcar a posição corrente da embarcação com base nas medidas de direção e tempo gasto no último trecho conhecido percorrido. Ainda hoje, esta técnica pode ser utilizada em navegação, e existem programas feitos para isto, caso haja problemas com equipamentos de GPS.

Na área de ambientes virtuais distribuídos, este conceito foi proposto pela DARPA (*Defense Advanced Research Projects Agency*) para uso em seu protocolo *Advanced Distributed Interactive Simulation*. Neste protocolo, cada nó (veículo ou unidade) envia um pacote de dados com sua corrente localização, nível de dano etc e, também um algoritmo específico de *dead-reckoning* para uso na previsão do movimento daquela unidade.

Cada nó da batalha simulada irá usar esta informação para prever qual a posição dos demais nós em função do tempo, até que uma nova atualização seja recebida. Cada nó irá também, por ter enviado uma cópia do seu algoritmo de *dead-reckoning*, saber o que os outros nós estão estimando acerca de sua posição e enviará uma nova mensagem de atualização, apenas quando ocorrer uma discrepância acima do tolerável.

Isto representa um duplo bônus, pois os nós tem uma visão relativamente acurada de todo o cenário fazendo a simulação como um todo, mais tolerante a atrasos (*lags*) tornando a simulação, portanto, mais imune ao problema de latência.

O segundo benefício de uso desta técnica é o potencial de diminuir o tráfego da rede reduzindo o problema de largura de banda. Todavia é importante considerar a carga de processamento de cada nó devido ao fato de o algoritmo de o *dead-reckoning* de todos os nós ter que ser calculado por cada nó da simulação.

3.3.1 Questões adicionais referentes a *Dead-Reckoning*

Ao definirmos algoritmos para implementação de *dead-reckoning*, devemos objetivar suavidade deste movimento, que é uma contribuição para o realismo e convergência da extrapolação com um ponto proveniente de uma mensagem de atualização (figura 4). Em (Singhal, S.K. & Cheriton, D.R., 1994) foi proposto um algoritmo (*Position History-Based Protocol*) que contempla os aspectos de convergência e suavidade de forma melhor que simples equações da física do movimento utilizadas no *dead-reckoning* original.

Também importante é levar em conta os riscos de uma extrapolação que podem levar a uma situação como na figura 5, onde pode ocorrer um choque que não faz parte da simulação ou mesmo a passagem de um objeto por dentro do outro. O algoritmo proposto por (Szwarcman, 2001) trata melhor estes riscos por incorporar "um poder de decisão" aos objetos que são orientados por metas como veremos a seguir.

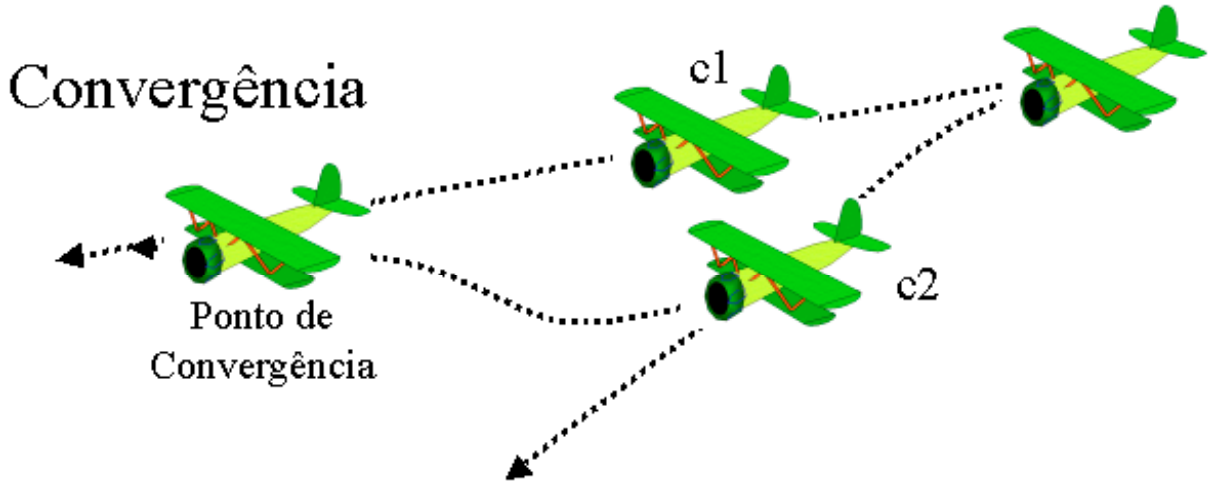

Figura 4: uso do algoritmo proposto por Singhal, S. K. & Cheriton, D.R. (1994), onde se pode observar a suavização do movimento e o caminhamento para um ponto de convergência

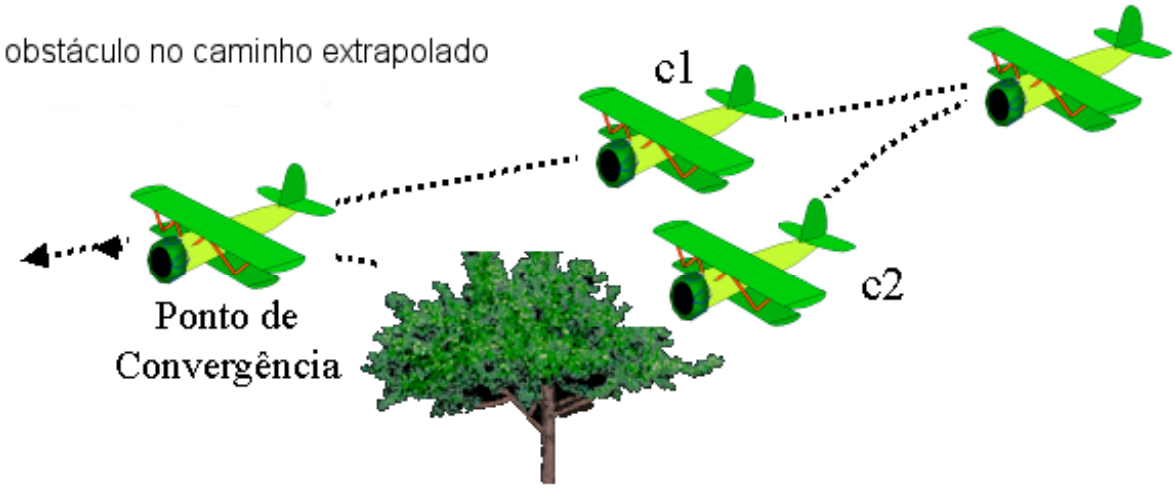

Figura 5: riscos do uso do algoritmo proposto por por Singhal, S. K. & Cheriton, D.R. (1994), onde se pode observar a presença de um obstáculo no caminho suavizado (que não é o caminho real percorrido)

Outra questão importante, crítica no caso de jogos de ação, é o fato de que visões ligeiramente diferentes devido a atrasos (*lags*), podem provocar situações como na figura 6, onde para um jogador, um míssil lançado atinge o alvo e, para outro, isto não ocorre.

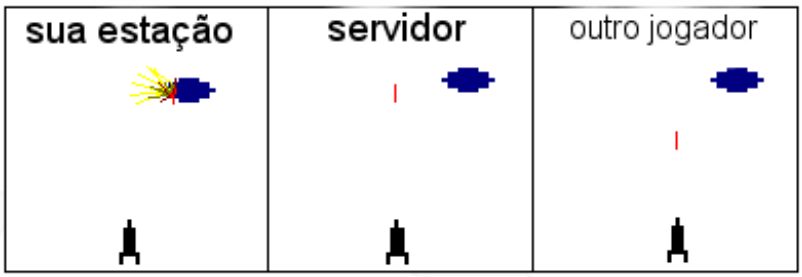

figura 6: visões diferentes da posição de um míssil lançado (e suas conseqüências) em função da latência do ambiente de rede utilizado (Haag, 2001)

Para contornar esta situação, (Singhal, S.K. & Zyda, M. 1999), e (Haag, 2001) propõem uma amostragem prévia para obtenção de um valor médio dos atrasos na rede para cada jogador, de

forma que o envio de pacotes (por exemplo, a posição do míssil na figura 6) já contemple a posição de um objeto não no tempo t de envio, mas sim no tempo (t + lag) corrigido.

3.3.1 Extensão do conceito de *dead-reckoning*

O conceito de dead-reckoning, inicialmente destinado a previsão de posição e velocidade de objetos móveis, foi extendido em diversos trabalhos, como por exemplo, para contemplar movimentos mais complexos como animação de estruturas articuladas de corpos (Capin, T.K., Esmerado J., Thalmann D., 1999).

Outro trabalho (Szwarcman, 2001) que abre muitas possibilidades de uso, é *o dead-reckoning* orientado a metas, onde se usa o conceito de piloto e clones. Piloto é o avatar de um dado usuário em sua estação de trabalho e clones representam os avatares dos usuários remotos.

Neste trabalho, assume-se que o importante são as metas, os estados finais dos clones em todas as estações e não seus passos intermediários. Por exemplo, é possível um determinado personagem receber a missão (meta) de deixar um certo aposento que tem duas portas. Numa estação ele sai pela porta da direita e em outra ele efetua a mesma ação, saindo, no entanto, pela porta da esquerda devido ao fato de, naquela estação, existir uma representação menos atualizada de outro clone impedindo a passagem pela porta da esquerda.

Estes pilotos e seus clones são dotados de comportamentos e têm a capacidade de tomar decisões. Isto, especialmente para jogos multi-jogador, é uma abordagem extremamente interessante, uma vez que nem sempre os passos intermediários de uma determinada ação irão influir negativamente na jogabilidade do ambiente.

3.4 Riscos de fraudes e problemas de arbitragem

A maioria dos esquemas de criptografia assumem que existem dois pontos confiáveis que desejam se comunicar utilizando um canal não confiável. Poderia até parecer absurdo o fato de que um destes pontos, que tem uma cópia da chave de encriptação ou uma autorização para participar da comunicação, possa tentar enganar a outra parte.

Este é exatamente o desafio para o projetista de arquiteturas de jogos multi-jogador online: - além da insegurança da própria rede (na maioria dos casos a Internet), temos contar com a possibilidade de que um dos jogadores possa estar, durante jogo, utilizando "suas próprias regras".

Não se pode assumir (Kirmse, 2000) especialmente em jogos com um número elevado (*massive*) de jogadores, que todos estejam agindo corretamente. Para o jogador, instalado em sua casa, "protegido" da ameaça de olhares de reprovação de terceiros, eventualmente conectado "anonimamente" utilizando um *proxy* para mascarar seu IP verdadeiro ou mesmo em um *cyber-café*, pode parecer tentador obter sucesso fácil nesta vida alternativa proporcionada por ambientes virtuais. De fato, é preciso cuidado no projeto para tentar diminuir a incidência de "vitórias" fraudulentas ou mesmo dificultar ataques contra o empreendimento como um todo.

Usando-se algumas técnicas, pode-se restringir ou desestimular o número de tentativas de violação de modo a se conseguir, pelo menos, lidar com um menor número de hackers. No mínimo, é uma linha de defesa a ser mantida para que o software mantenha-se protegido por algum tempo (Randall, 2002). A idéia é esta: se é impossível uma proteção total, ao menos temos que prover meios de manter o curioso honesto.

Num ambiente *peer-to-peer* não se pode assumir que algum dos jogadores seja confiável, mas pode-se estatisticamente, levar em conta um histórico de problemas relacionados a presença de um determinado usuário que, a princípio, deve fazer algum tipo de *login* para participar do jogo. Esta então seria a primeira regra de uma boa arquitetura defensiva: manter um *log* (Lee, 2003) contendo data e hora de início e, se possível fim de sessão (ou estimativa de fim de sessão por *timeout* pela ausência de notificação regular).

Também deve ser armazenado o endereço IP utilizado e, eventualmente, uma assinatura do computador utilizado para conexão. Uma "semente" semelhante a um *web cookie* poderia ser "plantada" a cada conexão e verificada na sessão subsequente. Obviamente, vamos lembrar que, para quem entende cada instrução em linguagem de máquina do programa e o seu protocolo, pode ser fácil contornar código de proteção num jogo onde toda a tecnologia está nas mãos deste jogador.

Como é relativamente fácil monitorar pacotes, é importante que as senhas de encriptação variem de sessão para sessão, ou mesmo numa mesma sessão. Isto porque um pacote encriptado e com checksum, portanto uma mensagem válida, pode ser gerada externamente e repetida de forma a gerar resultados positivos para o jogador desonesto.

Outra regra importante é não enviar para um jogador considerado suspeito, uma notificação do tipo "você está executando operações suspeitas, por favor suspenda suas atividades!". Na verdade, um servidor do jogo ou mesmo um website do fabricante de um jogo *peer-to-peer,* é que deve receber esta notificação para disparar atividades de auditoria especiais como gravação de atividades suspeitas, chat etc.

Outra possibilidade é disparar este procedimento de forma não determinística, ou seja, utilizar uma certa aleatoriedade no envio destas mensagens de monitoramento para que suscite no hacker, a dúvida sobre a existência ou não de monitoramento nesta e em também em outras situações não monitoradas, dificultando seu "trabalho".

Usando uma arquitetura cliente-servidor, poderíamos pensar que, pelo menos o servidor é confiável facilitando a segurança do ambiente. No entanto, há o grande risco de que, uma vez que o servidor tenha sido invadido ou tenha sua segurança comprometida, milhares de usuários possam ser prejudicados ao contrario de uma abordagem de um sistema composto de centenas de pequenos núcleos *peer-to-peer*.

Em qualquer caso, é importante que toda a comunicação seja encriptada, com exceção talvez , de chat entre os usuários. Como a alteração casual de protocolo é uma das tentativas de um *hacker* em compreender seu funcionamento, torna-se importante acrescentar mecanismos de *checksum* para evitar que a alteração de um byte em uma mensagem provoque o aumento de velocidade de um veículo, amplie o poder de uma arma, ou mesmo altere a pontuação de um jogador.

Pode ser útil o uso de assertivas em todos os programas para permitir a detecção de atividades suspeitas. Por exemplo, se um *hacker* altera sua cópia do programa e seu avião começa a viajar numa velocidade acima da máxima ou mesmo começa a voar em uma altitude acima da permitida para aquela aeronave, os outros participantes devem recusar sua conexão. Também possível é notificar o servidor, de forma automática ou manual, sobre o (mau) comportamento de outro jogador.

Transformar os jogadores em "policiais" do jogo é, portanto, mais uma sugestão de projeto. Evidentemente, deve-se analisar as reclamações de uma forma estatística, pois uma única reclamação pode ser indevida ou mesmo o reclamante pode ser um *hacker* tentando sabotar o ambiente. Aqui, cabe a citação latina: "*testis unus, testis nulus*" (testemunha única, testemunha nula).

Na arquitetura proposta, que também utiliza a tecnologia *peer-to-peer*, adota-se, além dos mecanismos de proteção convencionais descritos aqui, um esquema aleatório de usuário mestre (*host-peer*), para que nem todos os mini-ambientes componentes do grande jogo possam ser comprometidos pela presença de um *hacker*.

4. Arquitetura proposta

A idéia central deste trabalho consiste em criar um sistema capaz de contemplar jogos com um número grande de jogadores (*massive multiplayer*), utilizando a Internet, apresentando uma arquitetura híbrida cliente-servidor e peer-to-peer. Tendo em mente os tópicos discutidos na sessão anterior como limitações da rede utilizada e dos recursos disponíveis, implementou-se uma

arquitetura para contemplar jogos de estratégia integrados a jogos de ação. Podemos ter uma boa idéia da arquitetura proposta observando as figuras 7 e 8.

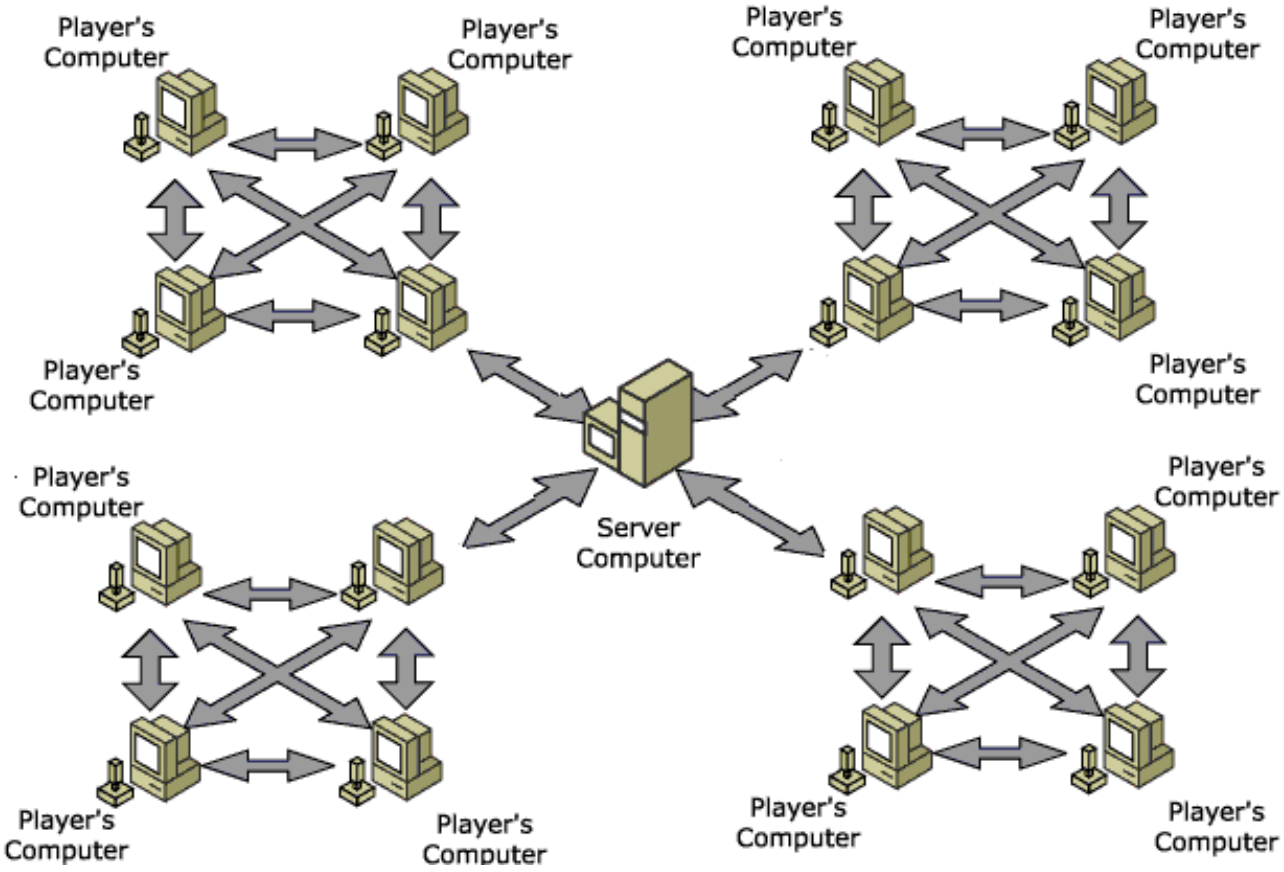

figura 7: salas ou sessões utilizando arquitetura P2P realizando a notificação de seu estado e coordenadas para o servidor web(central)

A classe de jogos de ação é representada pelo paradigma de salas de jogos *peer-to-peer* com um número limitado de participantes. Um participante da sala irá notificar um serviço web de que está ou não disponível para aceitar novas conexões. Este serviço web, além de atuar como *masterserver*, servirá como um repositório central de dados, onde todos os participantes de uma sala ou um em especial, dependendo da largura de banda disponível, irá atualizar a sua e, eventualmente dos demais, posição, velocidade e situação.

Um jogador que tenha o desejo (ou permissão) de participar do jogo de estratégia integrado ao jogo de ação irá conectar-se ao mesmo serviço web (figura 8) para obter as informações de todas as salas ou de um subconjunto de salas com as quais deseja interagir. Este jogador irá colocar situações de bônus e de reveses em cada sala e observar o efeito de sua atuação segundo critérios da instância (do jogo especificamente implementado). Deste modo os dois tipos jogos estarão integrados, o que, junto com a dinâmica de balanceamento de salas (Kozovits, 2003b) compõem uma arquitetura de jogo *massive multiplayer* de baixo custo.

figura 8: jogadores do jogo de estratégia interagindo com o mesmo servidor web da figura 7

Cada sala, ou seria melhor dizer mini-mundo, pois não precisa ser fisicamente uma sala, é independente das demais, assim o número de mensagens trocadas $O(N^2)$ não irá constituir um problema, pois estamos considerando N um número pequeno de jogadores. O número total de jogadores M, distribuídos em S salas pode ser alto, porém como cada sala envia uma amostra de sua situação para apenas um servidor, o número de mensagens é O(S). Como a comunicação com o servidor web não precisa ser freqüente, pois este servidor está suportando um jogo de estratégia, não há maiores problemas relativos a largura de banda.

Uma instância desta arquitetura proposta pode ser jogadores atuando como soldados num determinado cenário de ação de combate limitado a 8 ou 16 combatentes e jogadores atuando como generais, observando a atualização da ação numa vista aérea e alimentando uma ou outra sala com recursos "jogados de pára-quedas" na simulação. Os melhores generais e os melhores combatentes teriam seu desempenho registrado no banco de dados do sistema. A figura 9 ilustra a integração entre os dois jogos formando um grande jogo *massive multiplayer*.

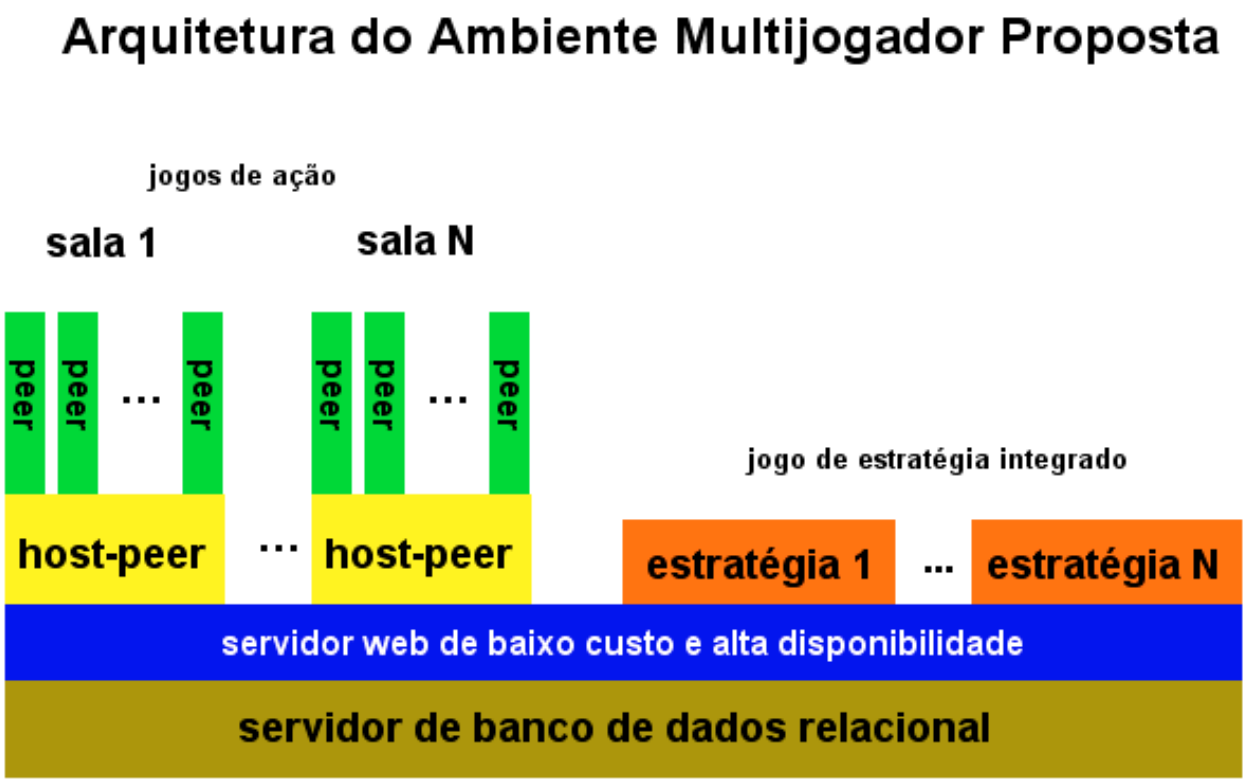

figura 9: diagrama de blocos ilustrando a arquitetura híbrida e integrada

5. Implementação

Foi utilizado Microsoft Visual C/C++ 6.0 para implementar o ambiente gráfico utilizando bibliotecas OpenGL-glut para efetuar o *rendering* dos objetos de cada sala. Os modelos utilizados têm a extensão MD2 e estão abundantemente disponíveis na Internet, assim como bibliotecas *open-source* para sua carga e renderização.

Criou-se uma biblioteca integrada ao glut para encapsular as complexidades de uso do Microsoft DirectPlay responsável pelas conexões peer-to-peer entre os jogadores.

Visando o uso de recursos computacionais de baixo custo, foi utilizado um servidor web .NET com páginas ASPX e um componente em Csharp para tratar da criptografia de mensagens entre clientes e servidor e realizar a conexão com o banco de dados Microsoft Access (ou SQL Server) residente no servidor web.

Para permitir a conexão simultânea do ambiente sem desviar e atrasar o *rendering* (*game loop*), foi implementada uma *thread* que, usando as classes para conexão web da *Microsoft Foundation Classes* MFC, realiza a tarefa de atualização de cada jogador no sistema baseado no servidor web Microsoft IIS .NET citado acima.

Maiores detalhes dos comandos SQL empregados para conexão com o ambiente, assim como a estrutura das tabelas estão disponíveis em (Kozovits, 2003b). Uma visão do ambiente implementado durante a execução é ilustrada pela figura 10.

Alguns problemas como balanceamento automático de salas quanto sua ocupação ótima e o mecanismo de gerenciamento automático de conexões para facilitar o interfaceamento com o usuário são tema de outro trabalho (Kozovits, 2003b).

Outro problema refere-se a transferência de objetos entre salas ou adicionados às salas pelo usuário jogando o jogo de estratégia (que congrega as salas). Quando estes objetos, imagens, malhas poligonais e texturas não estão disponíveis em cada sala no momento de sua inicialização, a

limitação de banda determina o uso de algumas estratégias que também são tema de um outro trabalho original (Kozovits, 2003).

figura 10: Dois jogadores(um *host-peer* e outro *peer*) durante uma sessão, onde se pode observar o *chat* entre os participantes e ter uma visão do ambiente implementado

6. Encerramento e Trabalhos futuros

A implementação de uma instância (um jogo) usando a ferramenta criada com a arquitetura proposta poderá dar um *feedback* quanto a jogabilidade deste tipo de ambiente contemplado e impor novos desafios ou mostrar novos problemas ainda não abordados.

O uso de avatares orientados a metas, seguindo a linha descrita em (Szwarcman, 2001) pode ser extremamente útil para possibilitar o aumento do número de jogadores por sala.

Finalmente, explorar a integração de *chat* (já implementado) com a movimentação de jogadores constitui um desafio para uma boa interface de usuário. Uma possibilidade poderá ser: enquanto o usuário se comunica com outro, o seu avatar segue seu objetivo utilizando IA, o jogador ao terminar a conversa deverá apenas movimentar as setas ou joystick para retomar o controle de seu avatar num mecanismo semelhante a um piloto automático.

7. Referências

Blau, Brian, et al., (1992) Networked Virtual Environments. ACM SIGGRAPH Special Issue on 1992 Symposium on Interactive 3D Graphics, Cambridge MA, pp: 157-164.

Blow, Jonathan, (1998), A Look at Latency in Networked Games. Game Developer issue 7/98, pp: 28-40.

Broll, Wolfgang, (1997), Distributed Virtual Reality for Everyone – a Framework for Networked VR on the Internet. Proceedings of the IEEE Virtual Reality Annual International Symposium, pp: 121-128.

Calvin, J., Dicken, A., et al., (1993), The SIMNET Virtual World Architecture. Proceedings of the IEEE Virtual Reality Annual International Symposium, pp: 450-455.

Capin, T.K., Esmerado J., Thalmann D., (1999), A Dead-Reckoning Technique for Streaming Virtual Human Animation, IEEE Transactions on Circuits and Systems for Video Technology, IEEE Press, Vol.9, No3, pp.411-414.

Das, Tapas K., Singh G., et al., (1997), NetEffect: A Network Architecture for Large-scale Multi-user Virtual Worlds. Proceedings of ACM VRST, pp: 157-163.

Everquest website (2003) [online]. Available from: http://everquest.station.sony.com/ [Accessed 27 September 2003]

Funkhouser, Thomas A., (1995), RING: A Client-Server System for Multi-User Virtual Environments. Symposium on Interactive 3D Graphics, Monterey, CA USA, pp: 85-92, April.

Funkhouser, Thomas A., (1996), Network Topologies for Scalable Multi-User Virtual Environments. Proceedings of the IEEE Virtual Reality Annual International Symposium, pp: 222-228.

Haag, Chris, (2001) Targeting: a variation of dead reckoning, In gamedev.net May 17, [online]. Available from: www.gamedev.net/reference/articles/article1370.asp [Accessed 5 September 2003]

Kirmse, Andrew, (2000), A Network Protocol for Online Games. In Mark DeLoura, ed., Game Programming Gems, Charles River Media.

Kozovits, Lauro E., (2003), Um estudo para visualização de objetos com geometria dinâmica em jogos multi-jogador. Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, (em preparação).

Kozovits, Lauro E., (2003b) Um estudo do uso de SGBDs relacionais em arquiteturas de jogos multi-jogador. Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, (em preparação).

Lee, Jay, (2003), Consideration for Movement and Physics in MMP Games. In Thor Alexander, ed., Massively Multiplayer Game Development, Charles River Media.

Macedonia, Michael R., Zyda, Michael J., et al., (1994) NPSNET: A Network Software Architecture for Large-Scale Virtual Environments. Presence 3(4), pp: 265-287.

Macedonia, Michael R., Zyda, Michael J., et al., (1995) Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments. Proceedings of the IEEE Virtual Reality Annual International Symposium , pp: 2-9.

Macedonia, Michael R., Brutzman, Donald P., et al., (1995b) NPSNET: A Multi-player 3D Virtual Environment over the Internet. Symposium on Interactive 3D Graphics, Monterey, CA USA, pp: 93-94, April.

Randall, Justin, (2002), Scaling Multiplayer Servers. In Mark DeLoura, ed., Game Programming Gems 3, Charles River Media.

Singhal, Sandeep K. & Cheriton, David R. (1994) Using a Position History-Based Protocol for Distributed Object Visualization, Technical Report CS-TR-94-1505. Available from: ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/94/1505/CS-TR-94-1505.pdf, Stanford Univ.

Singhal, Sandeep K. & Zyda, Michael. (1999), Exploiting Temporal Perception, Networked Virtual Environments: Design and Implementation., Addison Wesley, pp: 221-236.

Szwarcman, Dilza de Mattos, (2001), Dead Reckoning Orientado a Metas para Personagens Reativos, Tese de Doutorado, Departamento de Informática, PUC-Rio.

ToonTown (2003) [online]. Available from: http://play.toontown.com/webHome.php [Accessed 27 September 2003]

Ultima Online website (2003) [online]. Available from: http://town.uo.com/screens.html [Accessed 27 September 2003]

Yu-Shen, Ng, (1997), Designing Fast-Action Games For The Internet. (2003) [online]. Available from: http://www.gamasutra.com/features/19970905/ng_01.htm [Accessed 5 September 2003]