PEDRO MONTEIRO KAYATT

SISTEMAS DE REDE OBJETIVANDO PLATAFORMAS DE JOGOS DIGITAIS

São Paulo
2010

PEDRO MONTEIRO KAYATT

SISTEMAS DE REDE OBJETIVANDO PLATAFORMAS DE JOGOS DIGITAIS

Texto apresentado à Escola Politécnica da Universidade de São Paulo como requisito para a conclusão do curso de graduação em Engenharia de Computação, junto ao Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

PEDRO MONTEIRO KAYATT

SISTEMAS DE REDE OBJETIVANDO PLATAFORMAS DE JOGOS DIGITAIS

Texto apresentado à Escola Politécnica da Universidade de São Paulo como requisito para a conclusão do curso de graduação em Engenharia de Computação, junto ao Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

Área de Concentração:
Engenharia de Sistemas Digitais e Computação

Orientador:
Prof. Dr. Ricardo Nakamura

AGRADECIMENTOS

Este trabalho é dedicado a minha família, a qual tem me apoiado desde o começo. Através de longos períodos de tempo longe de casa, em meu diploma duplo, ou mesmo em rotinas incomuns vivendo em madrugadas, e se comunicando através de emails mesmo sob o mesmo teto. Risadas que sempre me fizeram mais forte.

Preciso também agradecer meus amigos os quais jamais me esqueceram. Àqueles antigos que inúmeras vezes me visitaram, na esperança que eu estivesse livre, e aos novos amigos que conheci nesta longa jornada de seis anos que estiveram comigo lutando e vivendo pelos seus sonhos.

Obrigado à minha mãe, Cristina, que sempre me mostrou o certo e o errado e como eu deveria seguir meu coração e não apenas a razão. Ao meu pai, Cesar Augusto, que é meu exemplo de conhecimento e inspiração de ser e superar, com todos os conselhos certos me mostrando que amar é cuidar.

Para meus irmãos; Renato que sempre foi um exemplo de guerreiro, desde o nascimento lutando contra uma deficiência e mostrando ao mundo que ele poderia ir além. Obrigado por me mostrar que nada é impossível. E ao Ricardo, um grande cara com um grande coração, sempre se esforçando ao máximo, servindo no exército e estudando na faculdade, você é doido meu irmão!

Para minha namorada, Keila Keiko que me apoiou e ajudou de inúmeras formas, me dando outros pontos de vista, um novo jeito de pensar e criando em mim uma esperança de incríveis mudanças. Obrigado pelos sonhos.

Aos meus avós, que sempre foram além, superando seus limites e sempre bem humorados, interessados em como minha vida se desenvolvia. Hoje, definitivamente eu sei de quem meus pais puxaram quem eles são.

A todos que conheci na Itália, na PoliMi e na Novecento, aprendendo e estudando, dando o nosso melhor para obtenção dos sonhos. Nas viagens, brincadeiras, guerras e bonecos de neve, noites geladas e feriados longe das nossas famílias, grazie mille ragazzi!

"Ainda que eu ande pelo vale da sombra da morte, não temerei mal nenhum, porque Tu estás comigo: a Tua vara e o Teu cajado me consolam.

Psalm 23:4

In memoriam Carlos Francisco Kayatt.

RESUMO

O campo dos videogames vem evoluindo e tornando-se a cada ano um caso de estudo mais sério. Muitas melhorias técnicas foram elaboradas e desenvolvidas para suportar tanto os negócios como a própria comunidade de *gamers*. Com uma receita que supera o Mercado Hollywoodiano e algoritmos computacionais ganhando espaço em diversas conferências, finalmente o assunto se apresenta com a seriedade que há.

Adicionando ao advento da Internet, somada a crescente evolução de banda de transmissão e a vasta adoção em lares por todo o globo, os serviços on-line relacionados a jogos se tornou uma necessidade. Neste trabalho é apresentada uma análise dos serviços de jogos em rede, em como estes estão organizados e quais diferenças podem ser encontradas entre eles focando em como é proposta uma superação das barreiras entre um jogador comum e o jogador on-line.

Não obstante, são apresentadas partes de um projeto que foi desenvolvido na empresa de jogos italiana Novecento Games, onde os pontos comuns de diferentes serviços de rede foram agrupados visando à construção de uma biblioteca comum.

Concluindo o documento, é demonstrado um projeto em código-livre de uma aplicação que exemplifica as teorias então estudadas e implementa de forma simples um protótipo de protocolo de comunicação de jogos.

Palavras-chave: Biblioteca comum. Engenharia. Jogo. Microsoft. Nintendo. Rede. Sistemas de Jogos. Sony. Vídeo Games. Xbox. On-line.

ABSTRACT

The videogames field has been evolving and becoming each year a more serious case of study. Many technological improvements have been done in order to support the gamer community and business. With revenue overcome the Hollywood market and the computer algorithms getting space on many conferences, finally the subject has presented it as serious it is.

Adding that to the advent of the Internet, the increased evolution of bandwidth and the great adoption of it on most of the homes across the globe, the online service related to gamming has become a must. In this thesis has an analysis of the actual networking gaming services, as how they are organized and which differences can be found in each one of them focusing on how they propose to suppress the barriers that separate the common player to the on-line player.

Nevertheless is presented parts of a project that was developed at Novecento Games, an Italian videogame company, where the common points of these networking services where assemble in the way to build a common library.

A conclusion of this document is, then, presented a project in open-source code of an application. This example shows the theories that had been studied and implements in a simple game communication protocol.

Keywords: Network. Gaming. System. Sony. Nintendo. Microsoft. Xbox. Library. Online. Videogame

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Acrônimo	Significado
ACK	Acknowledge
AP	Access Point
CEO	Chief Executive Officer
COD	Call of Duty
DLC	Downloadable Content
DS	Nintendo DS Handheld
E3	Electronic Entertainment Expo
FPS	First Person Shooter
FSM	Finite State Machine
GFWL	Games for Windows Live
HD	High Definition
ID	Identification
IP	Internet Protocol
ISP	Internet Service Provider
ITU	International Telecommunication Union
LAN	Local Area Network
MMORPG	Massively Multiplayer Online RPG
MW	Modern Warfare
NAT	Network Address Translation
NWC	Nintendo Wi-Fi Connection
P2P	Peer-to-Peer
PC	Personal Computer

PS3	PlayStation 3
PSP	PlayStation Portable
RPG	Role Playing Game
SCE	Sony Computer Entertainment
SDK	Software Developer Kit
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UK	United Kingdom
US	United States
VE	Virtual Environments
VPN	Virtual Private Network
WEP	Wired Equivalent Privacy
Wii	Nintendo Wii Console
WPA	Wireless Application Protocol
XBLM	Xbox Live Marketplace
XMB	Xross Media Bar

SUMÁRIO

1 INTRODUÇÃO

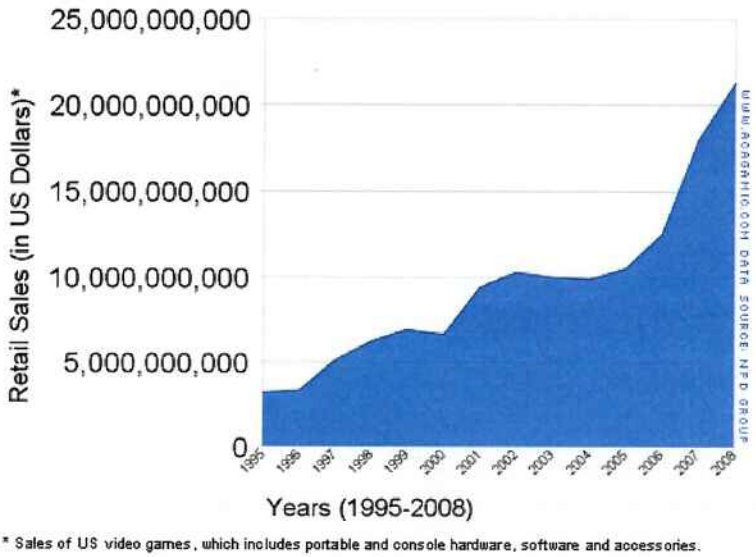
Neste capítulo são introduzidas as motivações do trabalho aqui descrito, assim como o problema a ser atacado e a estratégia de solução sugerida.

1.1 Motivação

Não faz muito tempo a internet não era nem sequer imaginada, uma rede virtual que pode conectar a todos, fazendo possível a troca dos mais variados tipos de mídia. Além disto, logo após o "boom" da Internet [I], rapidamente novas tecnologias começaram a surgir a fim de suprir a necessidade de estar sempre conectado. A rede mundial então tornou-se uma certeza, todo computador precisava estar conectado à Internet ou então ele seria inútil, afinal "para quê servem os computadores se eles não estão conectados?" (Vide Figura 2).

Logo após, outra "Revolução nas Conexões" teve espaço: o Wi-Fi. Onde além de estarem conectados, as pessoas agora podiam (e queriam) estar conectadas sem utilizar fios. Através destas evoluções a mídia vem mudando drasticamente a fim de seguir as violentas inovações.

Enquanto isso a indústria dos jogos veio mudando "da água ao vinho"(como pode ser observado na Figura 1), um pequeno negócio nos anos 80 se transformou em uma indústria de aproximadamente 42 bilhões de dólares (*IbisWorld, 2009*) superando até mesmo o mercado Hollywoodiano e havendo instalações nas casa da maioria das famílias no mundo.



* Sales of US video games, which includes portable and console hardware, software and accessories.

Figura 1 – Receita da Indústria de Vídeo Game (US)

Projetos que estavam acostumados a serem desenvolvidos em garagens agora têm uma média de desenvolvimento de três anos [I] e um orçamento de aproximadamente 5 a 20 mil dólares [II]. Assim esta "mídia" também vem sofrendo mudanças causadas pela "Revolução da Rede", o jogo multi-jogador online começou a se espalhar nos anos de 1995-1996 através do jogo chamado Quake produzido pela iD Software [III].

Deste ponto em diante, os jogos multi-jogador online começaram a ser uma febre, todos os jogos precisavam ter a opção "Jogar Online". Gamers começaram a gastar mais e mais horas jogando por toda a noite. Tendo isso em mente as maiores companhias de jogos desenvolveram sistemas complexos que podem suportar milhares de jogadores assim como diferentes estilos de jogos.

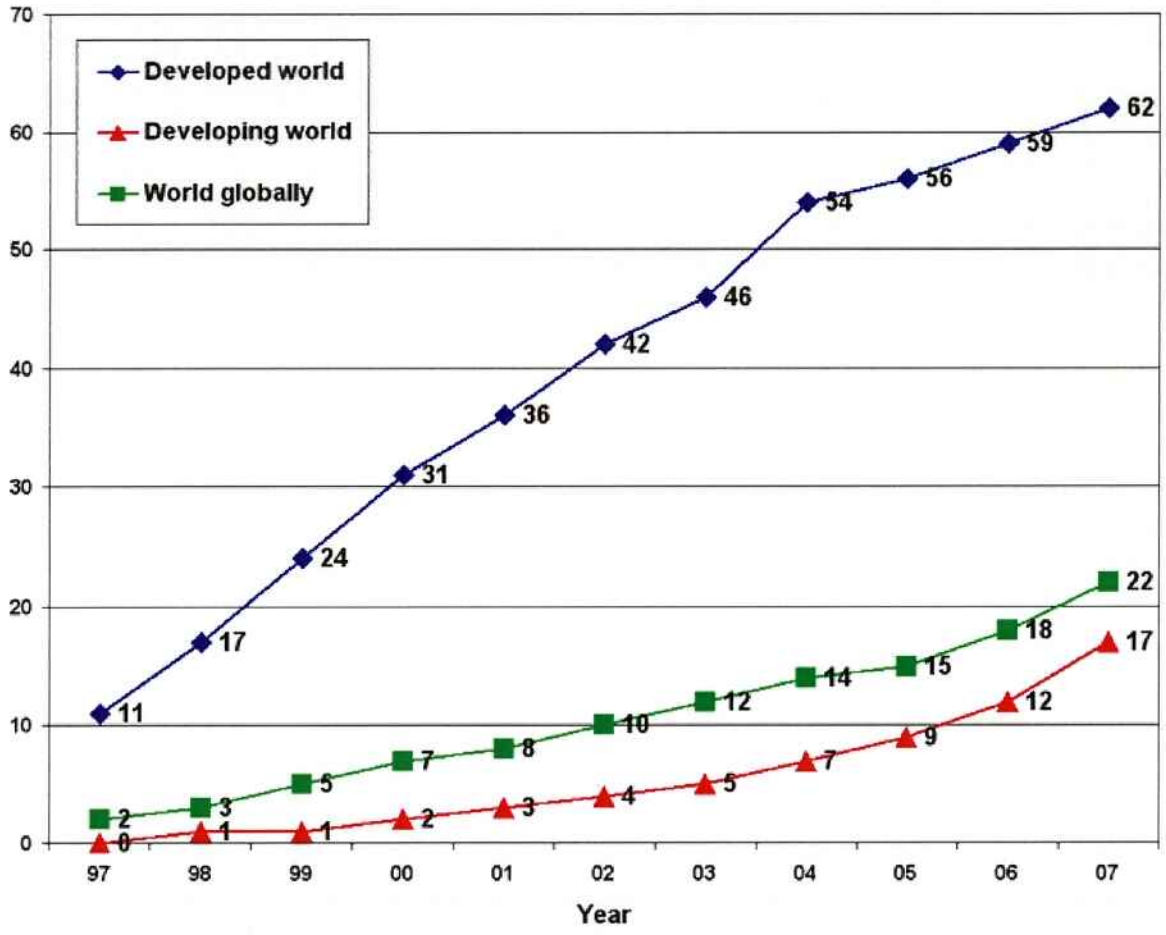


Figura 2 – Usuário de Internet por 100 habitantes (Fonte: ITU)

1.2 Definição do Problema

Tendo dito isto, é óbvio que o mercado online está em expansão, e na maioria das vezes, uma grande fatia é composta de usuários inexperientes que apenas começaram com o básico de comunicações em rede há alguns anos.

O foco do nosso problema será prover um sistema simples que possui uma rápida curva de aprendizagem, sem contudo comprometer a robustez necessária para expandir as experiência online até seus limites. A ideia principal é satisfazer tanto o novo mercado como os gamers padrão.

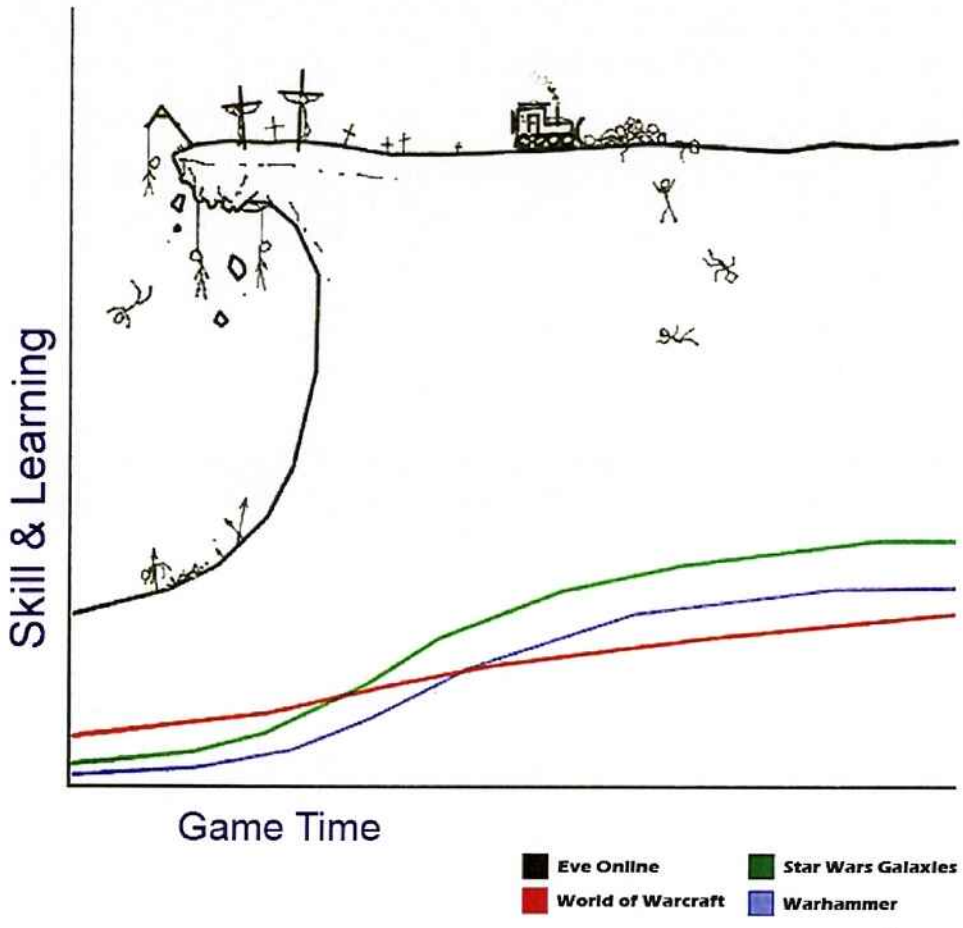


Figura 3 - Exemplo de Curvas de Aprendizado em Jogo

Além disso, um grande problema envolve o desenvolvimento multi-plataforma de uma aplicação on-line, como um jogo. Como será descrito em seções futuras, cada sistema tem seu próprio método de gerenciar os pacotes de rede e as conexões, assim como protocolos (encriptação, *handshanking*, entre outros).

Não obstante, apesar da contínua evolução da internet e do aumento da velocidade e da qualidade dos serviços, as aplicações online estão sempre sujeitas ao problema do atraso, que é ainda pior quando relacionado a jogos, levando o usuário a uma experiência frustrante, e no final das contas, a negação ao serviço.

1.3 Estratégia de Solução

Como solução foi decidida que as ações de rede devem ser expressas da maneira mais simples possível, tornando uma tarefa fácil tanto ao cliente como ao desenvolvedor acessarem essas funções. O primeiro passo do projeto é reconhecer as ações mais básicas nos diferentes sistemas de rede em uso no mercado atual.

O próximo passo será analisar os algoritmos e métodos que cada um provê, e decidir o sistema mais conveniente a um desenvolvimento de uma biblioteca para diferentes usos de rede. Utilizando das melhores práticas destes sistemas de rede uma interface terá de ser definida.

É necessário lembrar que o sistema precisa ser testado em uma aplicação real, tendo isso em mente, um modelo de jogo será desenvolvido (ou adaptado) com a finalidade de invocar a interface. A funcionalidade deve ser testada não só em ambiente fechado, mas também na internet pública, como será detalhado em seções futuras.

1.4 Estrutura

A estrutura deste documento é definida a seguir:

- Capítulo 1 é a Introdução;
- Capítulo 2 é o Estado da Arte, o qual provê informações relevantes e atualizadas das tecnologias disponíveis e sistemas de rede que estão sendo usados;
- Capítulo 3 é a Metodologia, onde passos de desenvolvimento serão tomados para exemplificar como nosso problema pode ser solucionado;
- Capítulo 4 expõe os resultados obtidos. Analisando os dados e as soluções adquiridas dos testes, assim como estes foram estabelecidos;
- Capítulo 5 é a conclusão. Nela é apresentada a conclusão do projeto e como pode ser desenvolvido em projetos futuros.

2 ESTADO DA ARTE

2.1 Xbox Live

Figura 4 - Xbox Live Logo

2.1.1 Introdução

A Xbox Live, também chamada de Live, é o Serviço de Jogos Online mais antigo analisado neste projeto e é o único que incialmente cobrava seus usuários, agora ele possui duas versões: Gold (que é cobrada) e a Silver (grátis) e as suas diferenças serão analisadas mais tarde (Seção 2.1.2). Live foi projetada para o primeiro console da Microsoft, o Xbox, e era uma de suas "killing features". O logo da Live (Figura 4) é estampado na capa dos jogos que a podem acessar.

Tomando partido da crescente adoção da banda larga (em 2001), o console Xbox (Figura 5) foi construído com uma porta Ethernet (10/100) e com um disco rígido interno, já prevendo o lançamento do sistema Live. Como prometido pela Microsoft, o serviço foi demonstrado inicialmente na E3 de 2002 e então em 15 de Novembro estava finalmente disponível aos usuários.

Deste aquele momento a Xbox Live vem recebendo muitas atualizações, e está em constante participação de Beta Testers, que são usuários que aceitam a tarefa de testar o sistema. Em troca do recebimento de informações para o melhoramento do sistema, a Microsoft os recompensa com presentes como camisetas, jogos, cartões de memória, etc.

No momento do lançamento do seu último console, o Xbox 360 (Figura 6), a Live já havia evoluído a outro nível. Ao início ela havia basicamente uma lista de amigos, em contrapartida no momento do lançamento do Xbox360 era possível assistir outros gamers, competindo um contra o outro, o que garantiu a patente de número 5000 à Microsoft.

O número de inscritos na Xbox Live também é um critério a ser considerado, em 6 de Janeiro de 2009, eles possuíam 17 milhões de membros e a Microsoft anunciou que em 5 de Fevereiro de 2010 eles já haviam 23 milhões de membros.

Não deve ser ignorado o fato de que a Microsoft também fornece o sistema Games for Windows – Live, também conhecido como GFWL, que usa um sistema parecido ao do Xbox, porém não cobrado, o qual não estará no escopo deste estudo.

Figura 5 - Microsoft Xbox Console

Figura 6 - Microsoft Xbox 360 Console

2.1.2 Funcionalidades

Gamercards: é basicamente um cartão de apresentação de um jogador, mostrando o apelido do jogador (gamertag), sua *MOTTO (descrição de quem você é)*; a gamer foto (usualmente de algum de seus jogos), uma pequena bibliografia, etc.

Virtual Avatar: um avatar criado pelo usuário que deve representá-lo dentro do mundo virtual.

Game achievements: são algumas conquistas que o jogador precisa obter.

Gamerscores: para cada achievement obtido o jogador irá receber um certo número de pontos (Gamerscore Points) que são somados neste item.

Reputation: é a reputação do jogador, obtido pelo voto de outros jogadores. A reputação padrão é 100% (cinco estrelas) depois que um jogador votou em outro usuário.

Friends list: uma lista de usuários adicionados pelo gamer. Através dela você pode trocar mensagens, iniciar uma vídeo/áudio conversa, convidar para jogos, comparar *achievements* dos jogos, etc.

Recent player list: é uma lista dos 50 últimos jogadores com quem o gamer jogou.

Complaint filing system: Este item permite que um usuário denuncie outro usuário por ter quebrado o Termo de Uso da Xbox Live.

Funções	Live Silver	Live Gold
Preço	Gratis	Mensalidade
Voice chat	Sim	Sim
Party chat	Não	Sim
Video chat	Não	Sim
Avatars	Sim	Sim
Downloadable content	Sim	Sim
Multiplayer gaming	Não	Sim
Parties	Não	Sim
Netflix movie streaming	Não	Sim
Sky Player	Não	Sim
Xbox Live Arcade point results	Sim	Sim
Facebook	Não	Sim
Twitter	Não	Sim
last.FM	Não	Sim
Zune	Não	Sim
1 vs 100	Não	Sim

Tabela 1 – Comparação entre Xbox Live Silver e Xbox Live Gold

Windows Live Messenger integration: integração com o conhecido sistema de mensagens instantâneas da Microsoft.

Access to the Xbox Live Marketplace: loja virtual que oferece jogos, músicas e conteúdo de filmes.

Voice Chat and Video Chat: usuários que tem microfone e Lite Vision Camera podem fazer Video Chat.

Multiplayer Gameplay: O principal foco deste trabalho, o multi-jogador é apenas providenciado pela Live Gold, como demonstrado na Tabela 1.

Enhanced matchmaking: outro de um dos principais focos do estudo, o matchmaking é responsável por encontrar os melhores jogadores disponíveis para jogarem juntos a mesma partida. Este item será posteriormente melhor detalhado (Ver Seção 2.1.4 TrueSkill Matchmaking).

Parental controls: opção muito atrativa que permite com que limitações para a exposição de conteúdos adultos seja controlada.

Inside Xbox: é um boletim informativo sobre as novidades da Xbox Live como eventos, produtos, entrevistas e jogos, totalmente integrado com o Xbox 360 Dashboard. O conteúdo de Inside Xbox é criado pelo time global de marketing do Xbox.

Last.fm: é um conhecido website que contém uma aplicação que permite usuários a ouvir suas músicas favoritas e pesquisar por artistas semelhantes a outros.

Social networking sites: famosas redes sociais também estão disponíveis no Xbox Live, como o Facebook e o Twitter. Usuários com a idade entre 13 e 17 anos podem acessar os aplicativos somente com o consentimento dos pais.

Zune: aplicação que permite usuários a assistir vídeos e shows de TV direto do Marketplace em resoluções de 1080p HD com 5.1 som surround.

Movie Parties: é uma aplicação que permite vídeos e shows de TV ser assistidos simultaneamente por amigos através do Xbox Live.

Sky Player: é disponível apenas no Reino Unido e na Irlanda, oferecendo TV Ao-Vivo e por demanda, incluindo Sky News, Sky Sports e o Disney Channel

Netflix: aplicação que permite seus membros alugar ilimitadamente (através de streaming) shows de TV de seus Netflix Instant Queue. Esta opção é apenas disponível nos EUA.

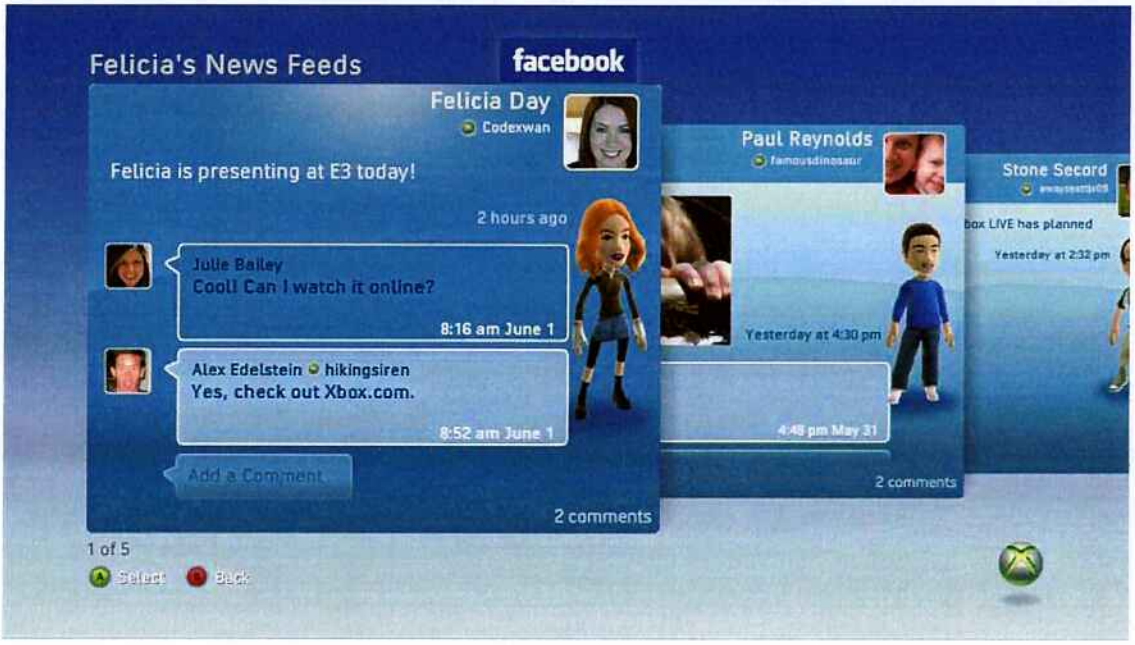


Figura 7 - Facebook application running on XBox Live

Game Room: é um espaço virtual, que oferece uma biblioteca de 30 jogos de consoles e fliperamas clássicos incluindo Centipede, Asteroids Deluxe, e Super Cobra, apresentados com seus designs originais. A compra de títulos de árcade permite que os jogos sejam jogados através do Xbox Live e do Games for Windows Live. Game Room é uma aplicação Xbox 360 e Windows com jogos vendidos separadamente.

2.1.3 TrueSkill Ranking

Na realidade o TrueSkill Ranking foi uma criação do Microsoft Research Laboratory (em Cambridge) e consiste em um algoritmo de classificação de Bayesian utilizado para encontrar o melhor nível aproximado de um jogador. Esse sistema é muito parecido com o sistema de classificação Elo, cuja classificação de habilidade é utilizado para jogadores de Go e de Xadrez.

A introdução do paper de lançamento do TrueSkill diz (em tradução livre): "Sistemas de classificação em jogos competitivos e em esportes servem para três principais funções. Primeiro, eles permitem que jogadores sejam equiparados com outros jogadores de habilidades semelhantes levando a partidas interessantes e equilibradas. Segundo, as classificações podem ser exibidas a jogadores e ao interesse público para então simular o interesse e a competição. Terceiro, classificações podem ser utilizadas como critério de qualificação para torneios."

É impossível medir o desempenho de um jogador, mas utilizando os dados da quantidade de jogos e taxa de vitória, algumas estimativas podem ser efetuadas. Sendo que a classificação de um jogador é atualizada cada vez que ele joga uma partida, assim, a classificação adquire maior confiabilidade sobre a real habilidade do jogador.

Na classificação TrueSkill, a classificação do jogador é datada como uma distribuição Normal (N), que é caracterizada por uma média (μ-mu) e uma variância (σ - sigma, que é a atual confiabilidade no valor de). Tal N(x), então, pode ser observado como sendo a probabilidade de um jogador possuir sua classificação X "correta".

$$p(\mathbf{s}) := \prod_{i=1}^{n} \mathcal{N}(s_i; \mu_i, \sigma_i^2)$$

Equação 1- Performance de um jogador

A Figura 8 a seguir demonstra o processo iterativo de como o TrueSkill funciona. Existem quatro tipos de variáveis: s_i para a habilidade do jogador, p_i para o desempenho de todos, t_i para o desempenho de times e d_j para a diferença de desempenho de times. A primeira linha de fatores codifica a prévia, o produto dos fatores restantes caracterizam a probabilidade do Team 1 > Team 2 = Team 3. A seta indica a mensagem ótima passando a relação. Primeiro, todas as setas claras de mensagem são atualizadas de cima abaixo. Em seguida a relação do desempenho dos nós da diferença dos times são iterados na ordem numérica.

Finalmente a classificação posterior é computada atualizando todas as mensagens das setas escuras de baixo a cima.

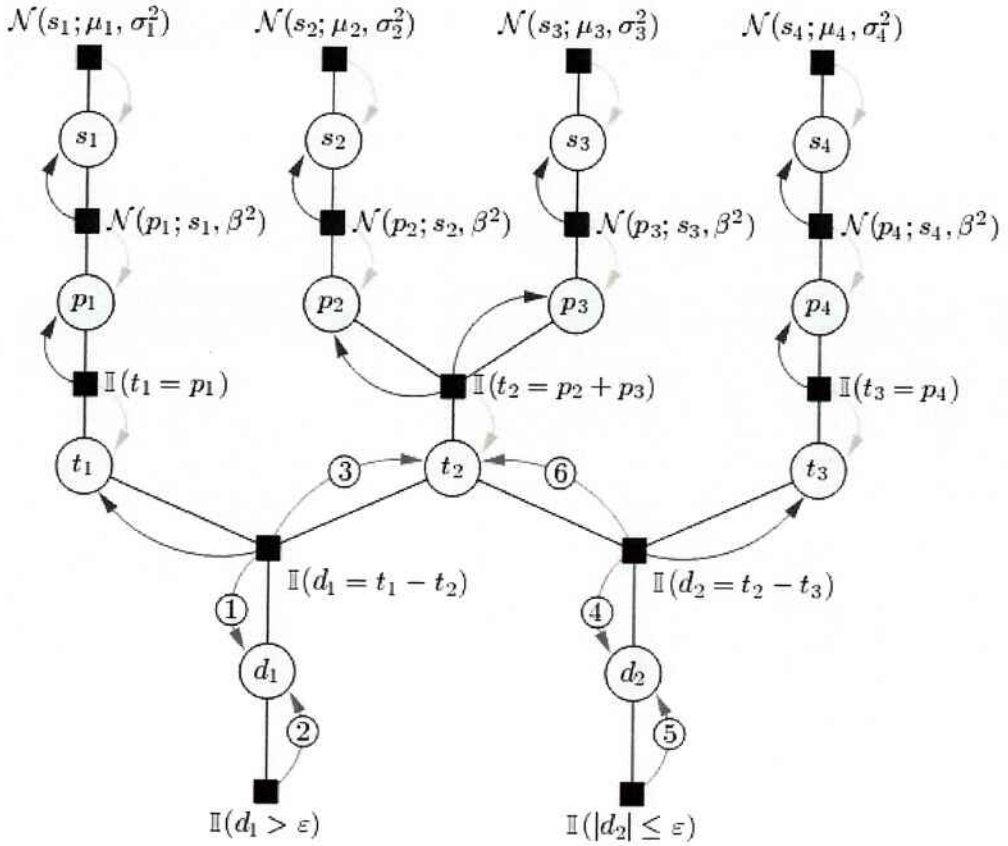


Figura 8 – Um exemplo do gráf0 de fatores da TrueSkill [IV]

Na Xbox Live os valores iniciais são μ = 25 e σ = 25/3, que corresponde à probabilidade de uma classificação positiva de 99%. Através da análise rápida dos dados os pesquisadores da Microsoft conseguiram chegar a três pontos importantes que estão apresentados a seguir:

- Gêneros de jogos diferem no número efetivo de níveis de habilidade. Jogos de sorte (como Gamão ou Uno) tem uma distribuição de habilidade estreita, por outro lado, jogos mais realistas (como simuladores de corrida ou *shooters*) tem uma larga distribuição de habilidades.
- Ao exibir o resultado da habilidade como resposta, para jogadores que normalmente checam seus estados, conseqüentemente eles vêem isso como recompensa ou punição. Para melhorar sua classificação jogadores começam, então, a selecionar seus oponentes mais cuidadosamente ou até a trapacear.

- A distribuição total de habilidade é deslocada para menor que a distribuição inicial quando os jogadores perdem seus primeiros poucos jogos (o que é esperado).

2.1.4 TrueSkill Matchmaking

Toda a complexidade da TrueSkill é sem valor se não colocada em prática. O propósito dela em nosso escopo é tornar possível encontrar a melhor solução de criação de partida, para isso é necessário enfatizar quais são as principais características de um jogador online e como alcançar sua satisfação.

Jogadores podem ser simploriamente divididos em duas categorias principais:

- *Play for Fun:* São jogadores que apenas querem conhecer novas pessoas ou então jogar partidas rápidas com seus amigos. Eles estão principalmente interessados em ter uma partida simples e agradável, não se preocupando com recompensas ou competição.
- *Play for Keeps:* Também conhecidos como jogadores *hardcore*, eles gostam de saber como jogaram, guardar as pontuações, obter todas as recompensas e bônus dos níveis. De fato eles querem competir para descobrir quem é o melhor.

Para satisfazer ambos os tipos de jogadores a Xbox Live tem dois tipos de jogatina online (respectivamente):

- *Player Match:* É projetado para recreação, este modo não afeta a classificação do jogador, ao invés disto apenas guarda os últimos jogadores e os jogadores mais frequentes que têm sido confrontados e os coloca com prioridade.
- *Ranked Match:* Este é o modo que utiliza o TrueSkill Ranking, tentando obter a habilidade do jogador e o colocar em confronto contra outros jogadores com habilidade semelhante.

Houve um tempo em que as experiências de jogo multiplayer eram estritamente projetadas para redes locais (LAN), entretanto agora o foco foi completamente alterado para experiências na Internet, o que gerou uma mudança de diferentes fatores que precisam ser revistos, um exemplo destes pode ser acompanhado na Tabela 2 e também na Figura 9.

	LAN	Internet
Número de jogadores simultâneo	8 – 32	1,000 – 1,000,000
Alcance de encontro informal	Sim	Não
Trapacear	Não	Sim
Anonimato	Não	Sim
Adequada Seção do Browser	Sim	Não
Latência	Conhecida	Desconhecido

Tabela 2 – conexões de LAN versus Internet

Um destes fatores mais importantes é certamente o número de jogadores simultâneos, para evitar que isto se torne um grande problema algumas recomendações podem ser seguidas, tentando obter o melhor matchmaking possível:

- *Usar mecanismos de filtros para partidas* (mapa, tipo, classe, etc.); isso irá minimizar os resultados e acelerar a pesquisa pela melhor solução.
- *Evitar Criar/Entrar seção* na interface do jogador; o gamer apenas quer jogar, os pormenores dos detalhes da configuração podem ser gerenciados pelo sistema de rede.
- *Nenhum jogador deve ter mais poder* na seleção dos parâmetros de jogo ou permissão para excluir outros jogadores.
- *Mostrar* informação sobre *o progresso do processo de matchmaking*; exibindo quantos jogadores são necessários ou a diferença de níveis atuais ajudam o jogador a estimar quanto tempo ele terá de esperar.

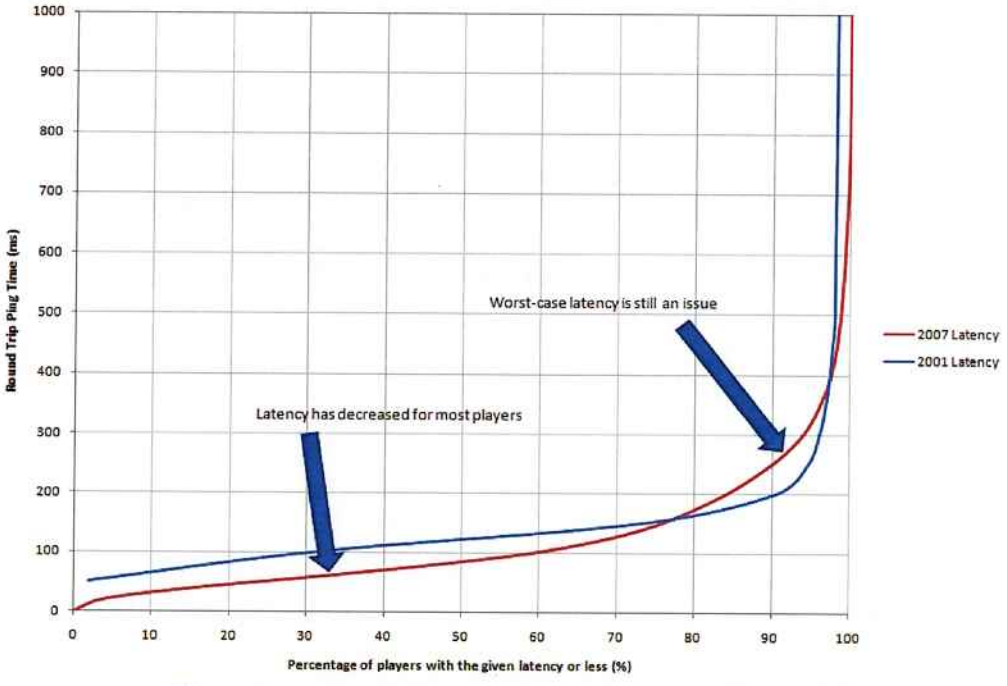


Figura 9 – Latência "Round Trip" do Console a Xbox LIVE

Para jogos ranqueados algumas outras características são enfatizadas,

- *Focar em um gameplay balanceado;* sempre se certifique que os parâmetros da partida são tais que ambos os times possuem iguais chances de ganhar.
- Não dê informações ao gamer sobre jogadores adversários ou parâmetros da partida antes da partida iniciar. Assim, o TrueSkill matchmaking não é comprometido, já que não existem penalidades quando jogadores abandonam a partida antes do começo do jogo.
- Desencorajar jogadores de deixar a partida depois que o jogo inicie, informando-os então que isto irá causar uma penalidade no seu nível de habilidade.

Não obstante há sempre a relação entre o tempo de espera e a qualidade da partida; em outras palavras, a partida precisa ser bem definida ao mesmo tempo em que o jogador não deseja esperar muito pelo início do jogo. No TrueSkill matchmaking o tempo de espera pode ser calculado por:

$$TempoEspera = \frac{GamesMode * MatchTime * PlayersPerMatch * SkillBins}{PlayersOnline}$$

Em que:

- *GameModes*; é o total de modos que o jogo possui. Modos de jogo são os diferentes jeitos que o jogo pode ser jogado.
- *MatchTime;* é a media de duração de uma partida (em minutos).
- *PlayersPerMatch;* é a média de jogadores por partida.
- *SkillBins;* é o número da variação admissível na habilidade.
- *PlayersOnLine;* é o número de jogadores online naquele momento.
- *TempoEspera;* é o tempo de espera para uma partida (em minutos).

Por este motivo, para jogos multiplayer na Internet os desenvolvedores de jogos precisam estreitar as possibilidades focando acelerar o processo de matchmaking. Algumas das idéias principais a serem consideradas como maneiras inteligentes de diminuir o tempo de espera podem ser as seguintes:

- *Mantenha o número de modos de jogo baixo:* Um número alto de modos de jogo faz com que a base de jogadores esteja dividida em grupos menores. Modos de jogo devem ser utilizados somente para definir estilos de jogo que necessitam de grupos de habilidades diferentes para ser bem sucedido. Na Xbox Live, modos de jogos são utilizados em conjunto com tipos de jogos para criar uma tabela de líderes para cada jogo ou par de tipos de jogo.
- *A equação precedente calcula a média de tempo de espera.* Para um skill bins bem populado, o tempo de espera pode ser menor mesmo se o número de skill bins é alto (isto é, o tamanho da diferença de níveis para o matchmaking é pequena). Se você escolher adicionar filtros por habilidade, você deve manter um limite de diferença (gap) de níveis adaptativo que aumenta através do tempo (de espera).
- *Projete partidas relativamente curtas para minimizar o tempo de espera no lobby do matchmaking.* Uma partida de duas horas pode soar divertida, mas não é apenas um compromisso de tempo extenso por parte do jogador, mas é também um aumento significativo de tempo de espera para outros jogadores buscando partidas.

2.1.5 Xbox Live Market Place

A Xbox Live Marketplace é um Mercado virtual projetado para o console Xbox 360 da Microsoft que permite que membros da Xbox Live efetuem o download de DLC (Downloadable Content) e conteúdo promocional.

O serviço provê filmes e trailers de títulos, uma loja de vídeos, demonstração de jogos, jogos do Xbox Live Arcade, jogos do Xbox Live Indie (antigamente conhecido como Community Games), Games on Demand (títulos originais do Xbox 260 e Xbox), temas do Xbox360 Dashboard, etc.

Em 2010 os serviços do Market Place foram estendidos a fim de suportar o WP7 (Windows Phone 7) caracterizando então o serviço a níveis mais elevados e colocando-o em ponta de competição com a Apple Store (dominante no mercado de fornecimento de produtos On-Demand, principalmente para o iPhone).

2.2 PlayStation Network (PSN)

Figura 10 - PlayStation Network Logo

2.2.1 Introdução

Em 15 de Maio de 2006, foi a vez da Sony. A maior companhia japonesa, que detinha a maior quantidade de consoles vendidos da última geração (PlayStation 2), começou a seguir os passos da Microsoft e anunciou seu próprio sistema unificado de jogos online. Inicialmente chamado de PlayStation Network Plataform, ela foi aclamada diferentemente da Xbox Live principalmente por não cobrar pelo serviço e focar em dois consoles: PlayStation 3 (PS3) e PlayStation Portable (PSP).

No inicio do serviço usuários podiam apenas se registrar através da interface de sistema de um PS3 ou de um PSP, atualmente as inscrições no sistema também podem ser feitas através do acesso ao web site da PlayStation Network [V]. Em 21 de setembro de 2006, Sony revelou na Tokyo Game Show que seria possível fazer aquisições de títulos através de download utilizando o sistema, eles iriam fornecer inicialmente jogos com pequenas quantidades de dados (pouco tamanho em dados).

O registro de uma conta na PlayStation network não é associada a número de série de um console; entretanto nesta situação o jogador não pode comprar títulos, tornando desta forma o registro de um console necessário para compra de jogos. Múltiplos consoles pode utilizar a mesma conta, até um número de cinco, mas o usuário não pode ser excluído, do contrário seus dados serão bloqueados.

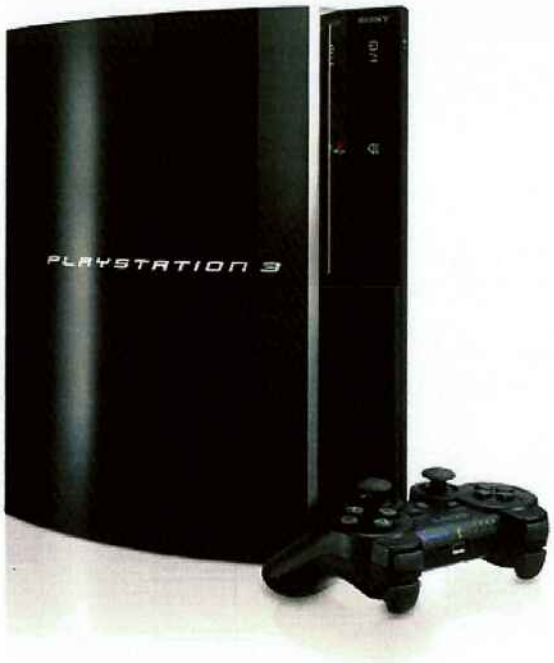


Figura 11 - PlayStation 3 Console

Figura 12 - PlayStation Portable (PSP) Handheld

2.2.2 Funcionalidades

Assim como a Xbox Live, a PSN tem um grande número de funcionalidades. Algumas destas estão citadas a seguir:

Profiles: este conceito não é muito diferente daquele do Gamercard (ver Xbox Live Funcionalidades na Seção 2.1.2). Ele possui o apelido do jogador, seu jogo favorito, uma citação, e seus Trophies (similar aos Xbox Live Achievements).

Sign-in ID/ Online-ID: basicamente é o e-mail utilizado no processo de registro. É através deste ID que o usuário efetua o login e também recebe informações e novidades da Sony.

Friend List: assim como no serviço do Xbox, ele fornece uma lista de até cem jogadores, os quais devem ser adicionados através da interface do sistema dos amigos.

Instant Messaging: é integrado no XMB (Xross Media Bar) como uma simples opção de envio de mensagens para jogadores amigos.

Lobbies/Matchmaking: Fornece uma solução interna para criação de Mundos, Lobbies e Salas visando fornecer um sistema de matchmaking (ver 2.2.3.).

Multiplayer gameplay: O jogo online da PSN suporta até sete jogadores locais e até 64 jogadores na mesma seção. Assim como o matchmaking este item será detalhado em futuras explicações.

Scores/Ranking: Diferentemente do sistema TrueSkill Ranking do Xbox (Seção 2.1.3) o rankeamento na PSN é calculada através da soma dos Trophies de um jogador e do nível obtido através de pontos gerados com os Trophies, as tabelas abaixo detalham a obtenção destes pontos.

Level	Pontos Necessários
Level 1	0
Level 2	210
Level 3	600
Level 4	1,200
Level 5	2,400
Level 6	4,000
Level 7	6,000
Level 8	8,000
Level 9	10,000
Level 10	12,000
Level 11	14,000
Level 12	16,000
Level 13	24,000
Level 14	32,000
Level 15	40,000
Level 16	48,000
Level 17	56,000
Level 18	64,000

Tabela 3 – Número de pontos para obter um Level

Trophy	Valor do Ponto
Bronze	15
Silver	30
Gold	90
Platinum	180

Tabela 4 – Número de Pontos adicionados por cada Trophy

Trophies: como já foi citado são recompensas dadas a jogadores quando estes completam determinados objetivos dentro dos jogos. Os Trophies podem ser: Platinum, Gold, Silver ou Bronze como descritos na imagem abaixo (Figura 13).

Avatars: Uma imagem virtual que o jogador tem como sua representação no mundo virtual do console. Estes avatares podem ser utilizados na rede social chamada PlayStation Home.

Voice/Video Chat: Usuários que possuem câmeras e microfone podem se comunicar entre si através de um chat em tempo real. Esta opção, porém, não é disponível quando em jogo.

Cross Game Chat Room: Mensagens de texto que podem ser trocadas ao mesmo tempo em que se está jogando.

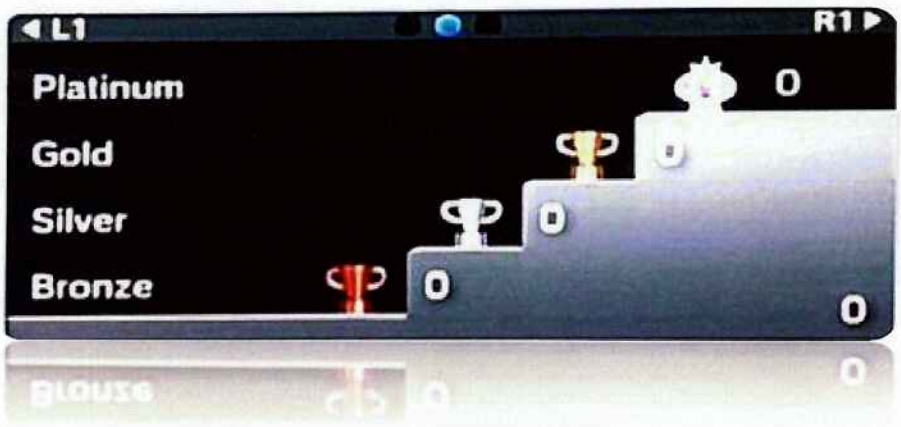


Figura 13 - Trophies disponíveis na PSN

Ad-hoc Party: Uma opção exclusiva para o PSP, que torna possível conectar um usuário a outro sem a necessidade do uso de um Access Point (AP).

Internet Browser: Utilizando o Google Search Engine, um modo simples de acessar páginas da Internet é o escopo desta funcionalidade.

PlayStation Home: É um jogo social e focado em comunidades em rede, similar ao famoso Second Life [VI], ele cria a experiência de um mundo virtual onde jogadores possuem casas, roupas, itens, etc.

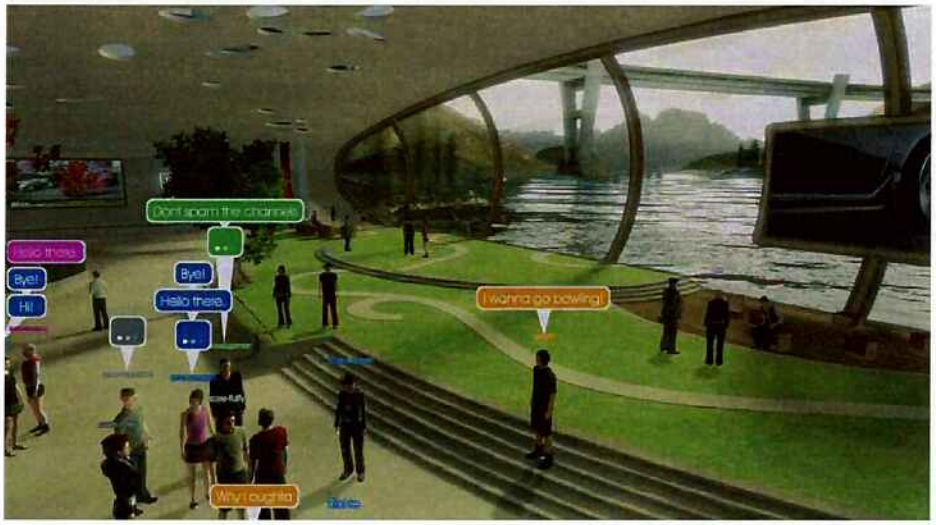


Figura 14 - PlayStation Home Central's Plaza

Parental Control: limita o acesso a BD, DVD e do Navegador de Internet para controle de visualização de crianças.

System Update: Tanto PS3 quanto PSP podem atualizar seus firmwares diretamente através da PSN.

What's New: É como um blog RSS que mantém o usuário atualizado com as últimas noticias sobre consoles e jogos.

Entertainment Third-Party Applications: PlayStation Network tem muitos contraltos com diferentes provedores de serviços e estes mudam de região em região. Os serviços online até agora são: FirstPlay, Qore, VidZone, ABC iView, BBC iPlayer, MLB.tv, Netflix, now TV, RTE Player, TVNZ ondemand.

PlayStation Store: Similar ao Xbox Live Market Place (ver Seção 2.1.5) é um provedor de download de conteúdo. O usuário pode fazer o download de arquivos, desde vídeos a jogos inteiros, incluindo jogos demonstração e trailers.

2.2.3 Sistema de Matchmaking da PSN

Assim como o Xbox tem seu TrueSkill Matchmaking, a PlayStation Network também possui um sistema de matching. Mas diferentemente do Xbox Live, o sistema de partidas da PSN é focado na organização de seções de jogadores, preocupado mais em grupos de jogadores do que no confronto jogador-jogador. Isso significa partidas em grupos, sempre maiores do que de dois jogadores.

Um detalhe que é preciso ficar esclarecido neste momento é que nós não teremos um sistema de classificação preciso como o TrueSkill, mas ao invés os jogadores irão selecionar aqueles contra quem irão jogar. O método é baseado em uma forma bem estabelecida: um jogador cria o jogo e outro jogador acessa este jogo criado.

Infelizmente não existem mais informações que possam ser compartilhadas, já que estas seriam parte de documentos confidenciais da SCE, impedindo a explicação de detalhes sobre este assunto neste ponto. Mais detalhes podem ser requisitados diretamente aos desenvolvedores oficiais ou a própria Sony.

2.2.4 PlayStation Store

A *Wikipédia* define: "O PlayStation Store é um mercado virtual online disponível para usuários dos consoles PlayStation 3 e PlayStation Portable, da Sony, via PlayStation Network. O mercado oferece uma gama de conteúdo disponível para compra e de graça. Conteúdo disponível inclui jogos completos, conteúdo adicional a jogos, demos jogáveis, temas e trailers de filmes e jogos. O serviço é acessível através de um ícone no XMB no PS3 e PSP. O mercado do PS3 pode ser acessado no PSP via uma conexão Remote Play com um PS3. Uma versão para Internet para o PSP também se faz disponível via Sony Media Go através de um PC. No dia 16 de maio de 2008, existiam 170 milhões de downloads disponíveis mundialmente na PlayStation Store."

2.3 Nintendo Wi-Fi Connection

Figura 15 - Nintendo Wi-Fi Connection

2.3.1 Introdução

Mesmo com a estratégia revolucionária que a Nintendo adotou com o Wii, último console lançado, eles também foram muito cuidadosos com os sistemas de conexão para melhorar a experiência de jogo. Antes de começar a falar sobre os serviços online é preciso ser dito que o Wii é um console, assim como o DS, que foi fabricado sobre aspectos inovativos e focado em um *market share* de não-jogadores, ao invés dos jogadores comuns.

O serviço foi lançado em 14 de Novembro de 2005 nos Estados Unidos e em 21 de Novembro na Europa, sendo lançado junto com os jogos Mario Kart DS e Tony Hawk's American Sk8land, dois títulos do Nintendo DS. A NWC (Nintendo Wi-Fi Connection) se tornou bem popular e em 30 de Maio de 2007 já possuía aproximadamente cinco milhões de membros e por volta de 200 milhões de acessos.

O serviço que estreou o uso da NWC no Wii foi Pokémon Battle Revolution, e utilizava exatamente a mesma tecnologia utilizada no DS, e os jogos que faziam utilização do serviço tinham o logo da Figura 15 em suas capas. Ao invés de uma conexão a cabo Ethernet (como o Xbox ou PS3, ver Seção 2.1.1 e 2.2.1 respectivamente) tanto Wii quanto DS utilizam conexões sem-fio (utilizando o protocolo de rede 802.11) e eles conectam a HotSpots públicos ou a APs. Entretanto a rede integrada do DS não suporta protocolos de segurança WPA, somente WEP.

Diferentemente de outros serviços de jogos online, a NWC não possui uma loja digital, ao invés disto a Nintendo tem Wii Shop Channel ou o DS Shop. Desta forma a distribuição digital de cada console é gerenciada por aplicações diferenciadas.

Figura 16 - Nintendo Wii Console

Figura 17 - Nintendo DS Handheld

2.3.2 Funcionalidades

Multiplayer match: Até dezesseis jogadores no Nintendo DS e até trinta e dois no Wii podem jogar juntos em modo online de graça.

Worldwide Matchmaking: A NWC cria uma infraestrutura que possibilita aos jogadores criar e procurar partidas com diferentes opções.

Leaderboards: Cada jogo pode ter sua própria lista de líderes para exibir qual jogador obteve mais pontos em uma partida deste jogo.

Tournaments: Uma funcionalidade que permite a organização de torneios dentro de jogos para competir com amigos possuindo partidas no modo knock-out (o perdedor sai).

Friend Codes: Esta é uma das funcionalidades unicast do NWC. Cada jogo (tanto no Wii quanto no DS) possui um número de identificação único (esteja atento que é cada jogo produto, não título), assim como cada jogador tem um ID único. Desta forma os dois podem ser combinados para criar este Friend Code. Este código possibilita adicionar outro amigo, se ambos em concordância, criando assim novas funcionalidades que seguem:

Friend List: Uma lista onde os jogadores podem ver se outros amigos estão online e interagir com eles.

Cooperative Play: Jogadores que adicionaram um ao outro com Friend Codes podem jogar juntos os desafios do jogo.

Text Chats: Um mensageiro instantâneo com o qual amigos podem trocar mensagens.

Voice Chats: Se o usuário possui os equipamentos necessários, ele pode falar com seu amigo registrado no Friend Codes.

Rivals: É uma funcionalidade similar ao Friend Codes, porém possibilita que um jogador memorize outro como seu rival (ao fim de uma partida) para jogar revanches.

Pay & Play: Através desta opção é possível ter acesso aos DLC (Downloadable content) pagando o conteúdo com Nintendo Points. Funcionalidade que foi lançada em 2008.

WiiConnect24: é a possibilidade de permanecer conectado a Internet até mesmo quando o console está em modo stand-by.

2.3.3 Matchmaking

O matchmaking da Nintendo Wi-Fi Connection tem duas principais características, que são: é suportado entre amigos (ver Friend Code em Seção 2.3.2) ou pode ser resultante de um processo com um jogador aleatório.

Para a segunda opção, entretanto, possui um sistema de nivelamento de habilidade que tenta juntar jogadores que possuem níveis de experiência e habilidade em jogo; este é efetuado através dos servidores da Nintendo utilizando um sistema de mapeamento de habilidade criando o ambiente mais facilitado possível para usuários de quaisquer idades ou habilidades.

Infelizmente especificações mais detalhadas sobre o funcionamento do sistema são confidenciais, não podendo ser expostas aqui. Apenas desenvolvedores oficiais da Nintendo possuem acesso aos manuais oficiais e a documentação que podem suprir esta informação.

3 METODOLOGIA

Essa seção é dividida em quatro partes principais: Estruturas de Rede, Exemplos de Conexões Socket, Organização de Seções de Rede e Processing (Desenvolvimento da demonstração). Os passos e estudos aqui apresentados são parte de um processo de contínuo aprendizado e pesquisa dos sistemas de rede atuais e da aplicação destes nos sistemas de rede descritos na Seção 2 – Estado da Arte.

3.1 Estruturas de Rede

Para projetar uma aplicação de rede é necessário a priori entender o quão diferente o processo de rede pode ser dependendo de como cada tipo de aplicação é desenvolvida. A Figura a seguir demonstra as diferenças entre variados ambientes virtuais (VEs) e o relacionamento destes com diferentes gêneros de jogo.

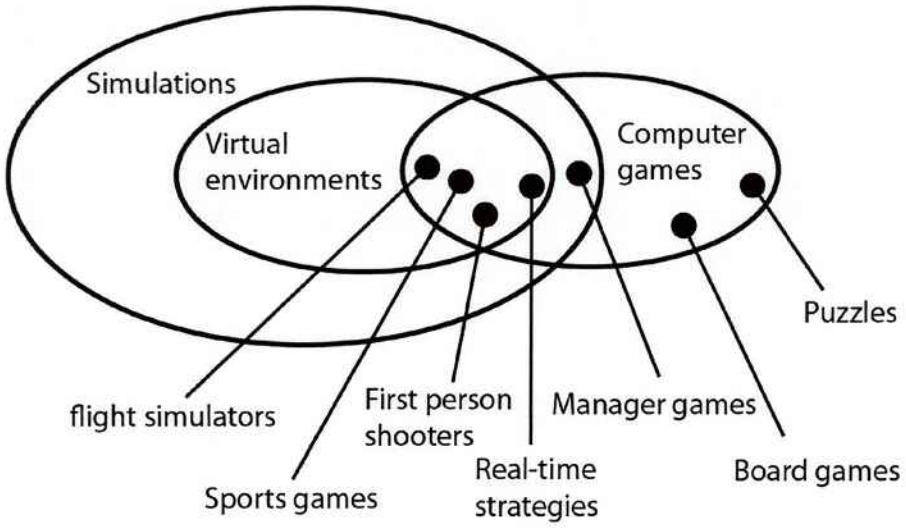


Figura 18 – Relacionamento de simulações, VEs e jogos digitais com o genero do jogo.

O significado desse processo de clusterização é tal que quanto mais um jogo está à esquerda (no campo de simulações e ambientes virtuais) mais importante é ter um tempo de resposta muito baixo, enquanto quebra-cabeças e jogos de tabuleiro, que estão localizados mais na porção direita do gráfico, podem sofrer atrasos maiores (também conhecido como *lag* pelos jogadores de rede).

A falta de um bom tempo de resposta em um jogo influencia drasticamente o desempenho do jogador, na Figura 19 é possível analisar as curvas de performance relacionadas diretamente a latência do pacote de rede. Com a finalidade de melhorar a latência e reduzir os requerimentos de banda existem inúmeras técnicas (as quais podem ser encontradas em [VII]).

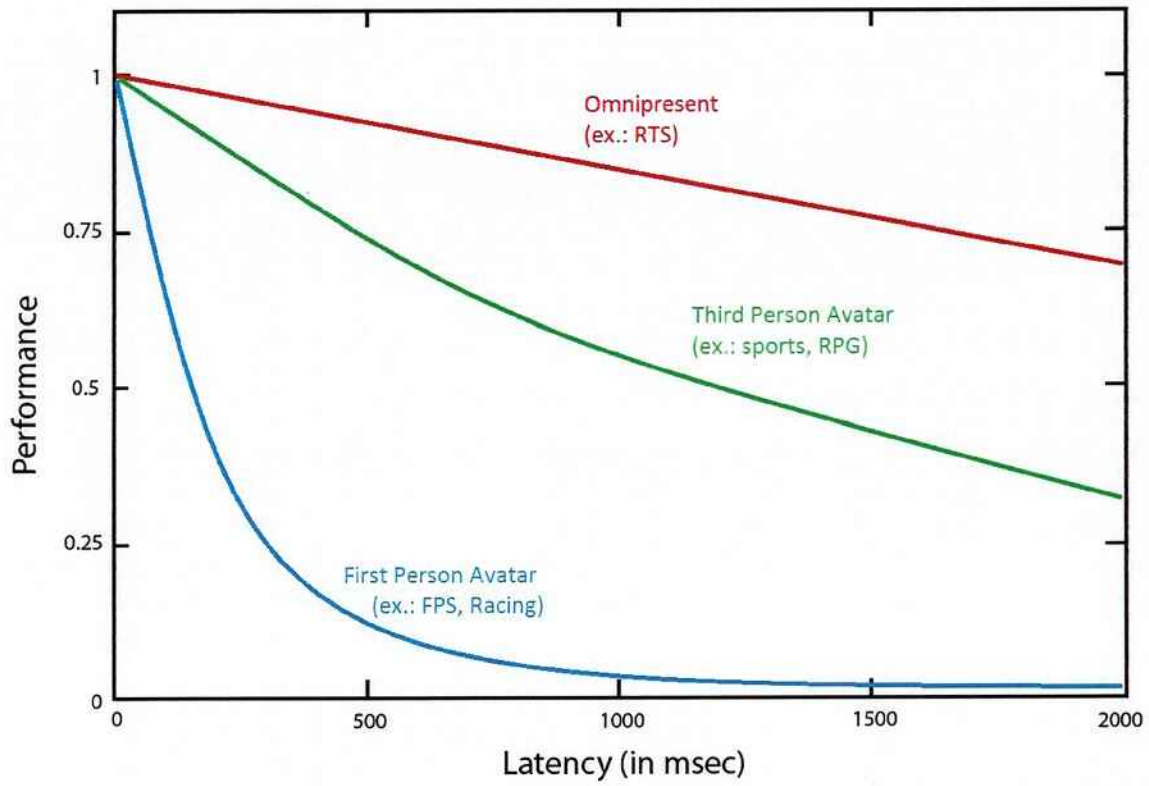


Figura 19 – Desempenho do jogador sob Diferentes Latências induzidas e para Diferentes Gêneros de Jogos.

Não obstante disto, não apenas a troca da informação é importante, mas também onde esta é armazenada. Existem diversas arquiteturas para gerenciar mensagens; assim como outras diversas para gerenciar o armazenamento dos dados. A importância dessas arquiteturas muda principalmente dependendo da importância de cada jogador no jogo e as diferentes configurações de hardware (ou conexão) que podem ser encontradas.

Por exemplo, a maioria das arquiteturas segue o modelo cliente-servidor. Neste modelo existe um jogador, referenciado como Host, que abre o jogo e espera que outros jogadores se unam a ele em jogo. Os outros jogadores que se unem a essa Seção serão administrados e terão seus dados geridos pelo Host.

Neste caso, quando cada jogador entra no jogo, o Host precisará efetuar uma operação de *handshake*. Esse processo troca primeiramente dados sobre o novo jogador, informando-o da situação atual do jogo com informações necessárias para o jogador visualizar o jogo como o Host. Neste mesmo momento o Host precisa atualizar a informação de todos os outros clientes, devido à entrada de um novo jogador.

Não apenas ao entrar em um jogo, mas também quaisquer ações que qualquer jogador tomar na partida terá de ser reenviado ao servidor (Host) e ele terá a necessidade de atualizar a informação a todos outros jogadores. Seguindo esta ideia fica claro que o número de dados que precisa ser enviado é sempre maior do que poderia ser, mas o processo sempre é sincronizado e controlado pelo servidor.

Também se um jogador que é considerado o servidor tem um problema e consequentemente perde a conexão, a partida então irá certamente parar sendo que os outros jogadores não receberão nenhum tipo de resposta um do outro.

Além disto, há outras arquiteturas, apresentadas na Figura 20, e cada uma delas tem seus pontos fortes e fracos. A melhor opção é sempre analisar cuidadosamente o tipo de jogo (gênero do jogo) para o qual a função de rede será desenvolvida.

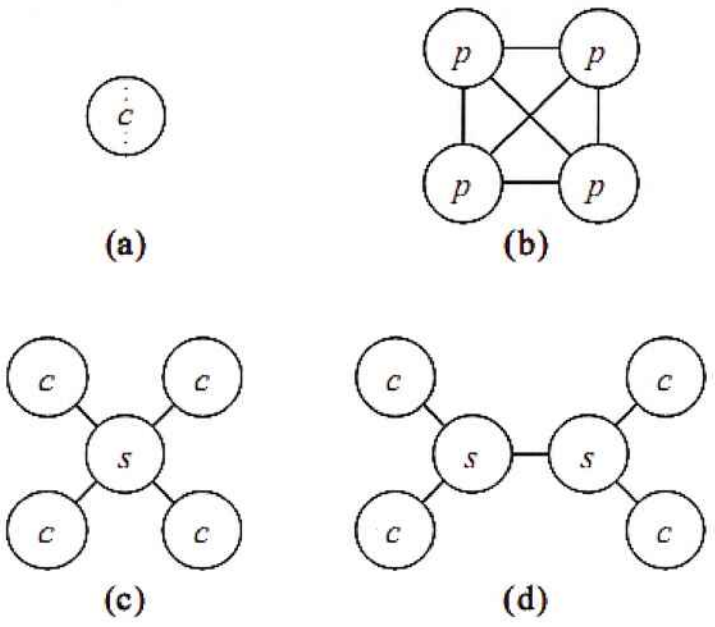


Figura 20 – Níveis de desenvolvimento: (a) split-screen, (b) uma arquitetura peer-to-peer, (c) uma arquitetura cliente-servidor, e (d) uma arquitetura rede de servidores.

Por outro lado existe a arquitetura peer-to-peer (ponto-a-ponto) em que cada jogador troca informações diretamente com os outros jogadores, havendo a necessidade de uma banda realmente maior, já que cada jogador irá enviar a mensagem N-1 vezes (sendo N o número de jogadores). Entretanto nenhum dos jogadores é essencial para a continuação da partida, excluindo então os problemas gerados por jogadores que perdem a conexão.

Por exemplo, quando o gênero do jogo é um tiro em primeira pessoa (em inglês FPS) é necessário que toda a informação seja concisa e o tempo de resposta o menor possível. Neste caso geralmente é escolhida a plataforma cliente-servidor, levando em conta que a experiência do jogador seria decrescida enormemente se fosse utilizada a configuração de peer-to-peer, devido ao fato da grande quantidade de dados na rede e que para alguns jogadores esta informação poderia chegar em tempos diversos, não levando em conta a possível perda de pacotes.

Não é complicado perceber que conexões de peer-to-peer seriam melhores se tratando de números mínimos de jogadores, e servidores para números maiores, apesar disso é necessário ser considerado que no caso o servidor tem um trabalho maior a executar, assim como necessita de uma banda maior para lidar com todo o câmbio de mensagens.

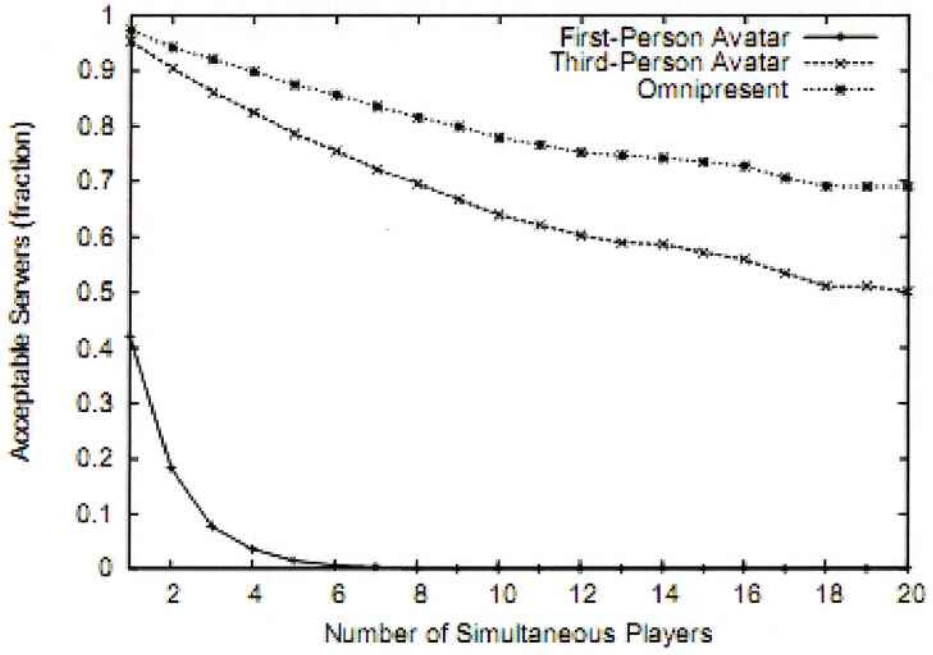


Figura 21 – Fração de Servidores Aceitáveis versus Número de Clientes Jogando Simultaneamente para Diferentes Gêneros de Jogos

Entretanto a arquitetura rede-de-servidores pode sustentar diversos tipos de jogos, sendo que um servidor único não pode suportar o processamento de dados e o tempo de resposta de um jogo em questão. A Figura 21 exibe um gráfico de [VIII] que avalia a capacidade de um número crescente de jogadores em um servidor mantendo uma experiência positiva de jogo.

Não muito distante destas arquiteturas também encontramos as arquiteturas de dados da partida. Na Figura 22 é possível observar três diferentes tipos de arquiteturas: Centralizada, Distribuída e Replicadas. O porquê de estas serem separadas da organização de redes é para possibilitar a combinação entre elas de diversas formas.

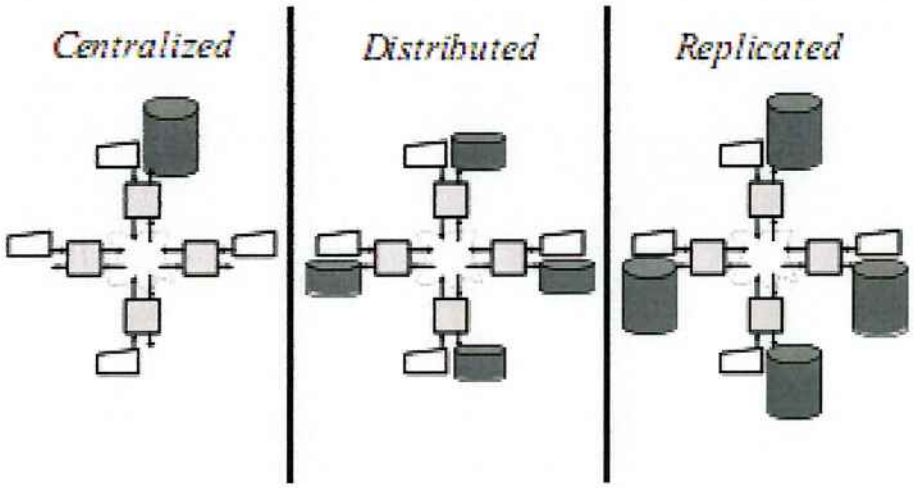


Figura 22 – Na arquitetura de Centralizada o servidor de dados (nó) registra toda a informação. Na Replicada, cada nó gerencia uma replica de todos os dados. Na Distribuída, a informação é distribuída entre os nós.

Não é incomum combinar diferentes designs, como a arquitetura cliente-servidor com a base de dados Replicada, desta maneira o sistema pode verificar perda de conexão de jogadores e notificar se um jogador da partida perde sua conexão subitamente. Então o sistema começa um processo para alterar o host para outro jogador, já que todos eles dividem a mesma informação.

Entretanto, o desenvolvedor precisa ser cuidadoso porque a consistência dos dados será controlada através de modelos de software, e o processo supracitado terá um grande impacto na largura de banda, adicionando muitos dados a serem enviados todo o tempo. Na verdade estes dois fatores, consistência e resposta, definem os modelos de arquitetura de controle e de dados.

Visando obter uma alta consistência, a arquitetura precisa garantir que todos os processos estejam rodando notoriamente sincronizados. Normalmente isso requer uma largura grande de banda, baixa latência e um pequeno número de nós remotos. Por outro lado, para obter uma alta responsividade (ou oportunidade [VII]), os dados precisam ser processados o mais rápido possível, o que leva a baixa sincronia. Não apenas, mas para manter a responsividade alta diversos tipos de compressão de dados e outros algoritmos (para diminuir o envio de dados) são utilizados, levando a um peso computacional maior.

A Figura 23 demonstra o problema. Um jogo está em execução em um nó local, ele envia uma mensagem de controle através do *relay* e obtém mensagens dele. Em troca o *relay* troca mensagem com outros nós através da rede (no caso especifico a Internet). Aqui o *relay* é um conceito lógico que expressa como o controle pode afetar os dados.

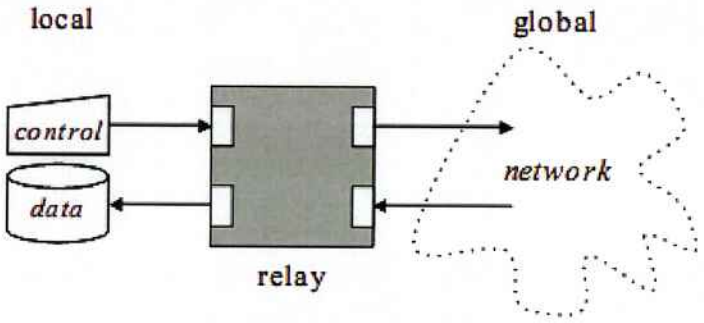


Figura 23 – Arquiteturas definem como as mensagens são chaveadas entre nós locais e remotos.

Além disto, a quantidade de banda também é medida dependendo da técnica de transmissão, como vemos na Figura 24. Antigamente, nos tempos de implementação em LAN, era comum utilizar de técnicas de Broadcast (Figura 24 (c)), enviando mensagens a todos os usuários da rede [IX]. É claro que isto nos leva a problemas maiores quando um número de participantes aumenta. Entretanto as mensagens geralmente tinham mais de apenas um destinatário, sendo então um desperdício enviar em Unicast (Figura 24 (a))para diversos usuários.

Então nos anos 90 a arquitetura Multicast (Figura 24 (b)) [X],que é um meio-termo entre Broadcast e Unicast, surgiu, permitindo que usuários se unissem a grupos de interesse. O usuário então envia uma mensagem a um grupo, semelhante ao Unicast, e o grupo recebe a informação, da mesma forma que em Broadcast.

Assim o Multicast se tornou fortemente aplicado nos sistemas de rede para jogos digitais.

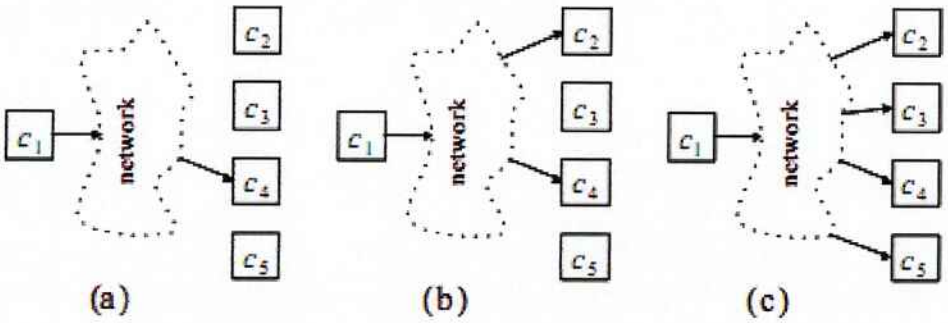


Figura 24 – Técnicas de transmissão

Adicionado, um problema muito comum que é encontrado quando conexões de Internet são gerenciadas, é o endereçamento de um jogador em específico para o envio da mensagem. O problema é gerado devido ao fato de muitos usuários utilizarem roteadores ou servidores *Proxy* que dividem suas conexões, método ainda mais popularizado para conexão de dispositivos Wi-Fi (APs). A Figura 25 expõe um exemplo de jogadores diferentes utilizando (e não) servidores *Proxy*.

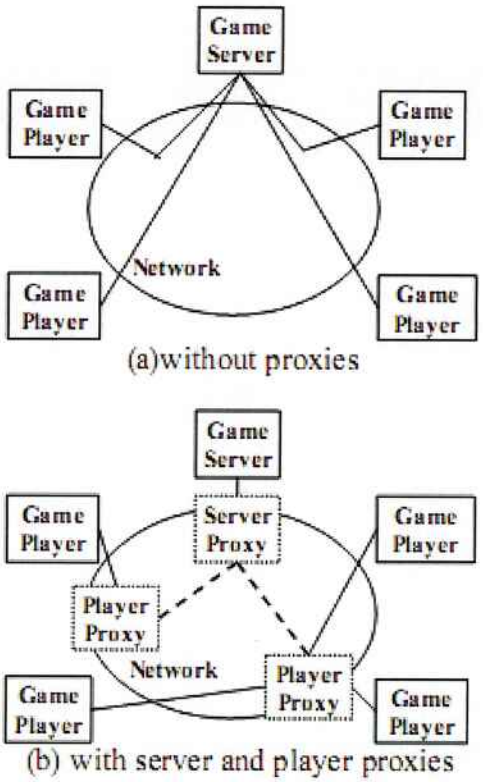


Figura 25 – Distribuição de um ambiente de jogo

O método que governa, na Internet, sobre a interpretação de endereços privados de rede para endereços IP é o NAT. Como definido em [XI]: "Network Address Translation é um método pelo qual endereços IP são mapeados de um reino a outro, na tentativa de fornecer rotas transparentes a *hosts*. Tradicionalmente, dispositivos NAT são utilizados para conectar endereços isolados com endereços privados não registrados a um único e global endereço externo. (Tradução livre)."

Aplicações de rede precisam de um IP externo para se comunicar, com este propósito foi criado o STUN (Simple traversal UDP over NATs) que classifica as implementações NAT como: Cone Completo (Figura 26), Cone Restrito (Figura 27), Cone de Porta Restrita (Figura 28), e Simétrico (Figura 29).

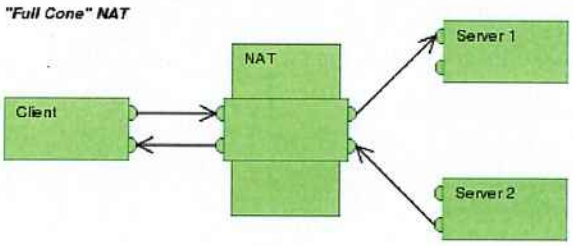


Figura 26 - Full Cone NAT

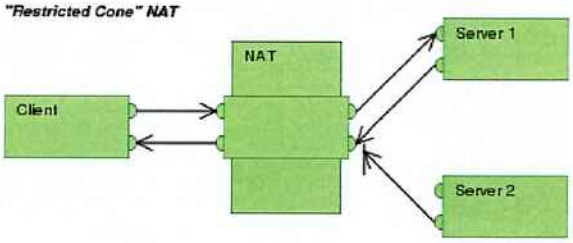


Figura 27 - Restricted Cone NAT

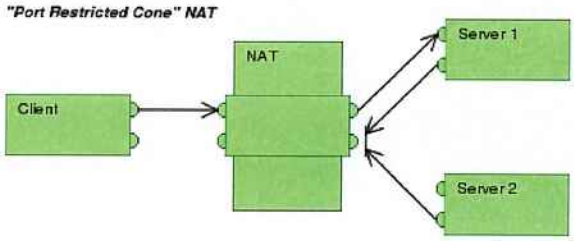


Figura 28 - Port Restricted Cone NAT

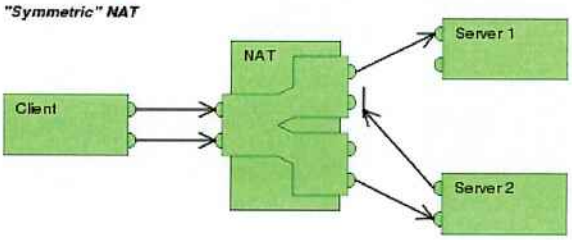


Figura 29 - Symmetric NAT

Entretanto estes procedimentos foram reprovados de um estado padrão, desde que muitos dos métodos se provaram ineficientes ou inadequados para classificar diversos dispositivos. Um novo padrão foi formalizado na RFC 5389 em Outubro de 2008 e agora o acrônimo STUN significa: Session Traversal Utilities for NAT.

Por sorte os sistemas de rede em questão neste trabalho conseguem trabalhar nos mais comuns usos de NAT e possibilitam a comunicação unicast para consoles. Apesar de que em algumas situações o direcionamento de portas (Port Fowarding) deva ser configurado.

3.2 Exemplos de Conexão de Socket

Nesta seção é explicado o básico para se estabeler uma conexão entre dois dispositivos utilizando um Socket. Como definido em "Um par de socket para conexões TCP é uma quadrupla-tupla que é definida por dois pontos finais de conexão: O endereço IP local, porta local, endereço IP externo e porta externa. Um par socket identifica exclusivamente toda conexão TCP de uma rede." (Tradução Livre).

Antes de tudo, todos os conceitos sobre sockets em TCP podem ser estendidos ao UDP, como citado também em [IX], e esse processo é comum sendo que um dos pontos principais em jogos é diminuir o pacote de dados e minimizar o uso da banda, aparte que perder pacotes não seria tão problemático já que o fluxo de dados é continuo, e sobrescreveria os pacotes antigos com novos em um curto espaço de tempo. Entretanto uma mensagem de ACK pode ser utilizada para mensagens essencialmente importantes.

O exemplo aqui apresentado demonstra o uso básico de sockets para construir uma aplicação simples de Cliente-Servidor que será necessária para entender as comunicações de baixo nível que acontecem em Sistemas de Rede. Mais detalhes são descritos na Seção 4 – Resultados.

Primeiro, nós devemos entender como o servidor socket deve funcionar. Para isso utilizaremos de estruturas socket que são definidas em um ambiente C/Unix, as quais podem variar em outras plataformas, não alterando o conceito aqui visto. A conexão por parte do servidor deve ser aberta e então esperar por uma conexão do cliente, logo, uma porta deve ser especificada, conforme o exemplo da figura abaixo:

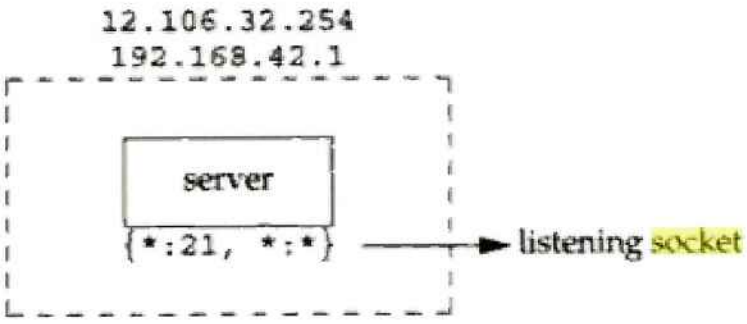


Figura 30 – Servidor Socket com uma abertura passive na porta 21.

Para elaborar o processo de "escuta" o socket precisa ser criado e depois alterado seus parâmetros, o método *BIND* deve ser chamado, este método na verdade liga um socket à porta, como podemos ver no código abaixo [XII].

```
hints.ai_family = AF_UNSPEC; // set to AF_INET to force IPv4
hints.ai_socktype = SOCK_DGRAM;
hints.ai_flags = AI_PASSIVE; // use my IP

if ((rv = getaddrinfo(NULL, MYPORT, &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    return 1;
}


// loop through all the results and bind to the first we can
for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
            p->ai_protocol)) == -1) {
        perror("listener: socket");
        continue;
    }

    if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sockfd);
        perror("listener: bind");
        continue;
    }

    break;
}
if ((numbytes = recvfrom(sockfd, buf, MAXBUFLEN-1 , 0,
    (struct sockaddr *)&their_addr, &addr_len)) == -1) {
    perror("recvfrom");
    exit(1);
}
```

Neste caso foram criados um socket na primeira função destacada, e logo depois disto nós conectamos o socket a um endereço, isto é feito através da define MYPORT, que é a porta de escuta da conexão. O próximo passo é estudar como o cliente irá se juntar a este servidor, através dos parâmetros de conexão do IP e a porta bem definidos. No código destacado em terceiro lugar, vemos o servidor esperando por uma conexão no socket recém-criado.

Na Figura 31 nós temos uma simulação do que aconteceria quando um cliente requisita a conexão ao servidor. Para tal o cliente precisa saber a priori o endereço do servidor e a porta que fora aberta. Depois disto ele também precisa abrir um socket, com toda a informação sincronizada do servidor, neste momento o cliente conecta seu socket ao do Server, criando um par de sockets.

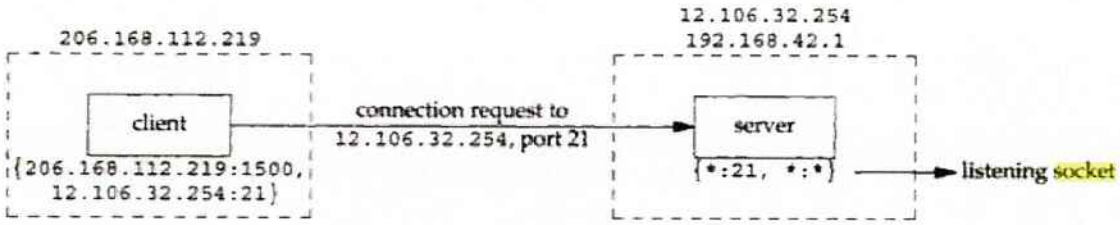


Figura 31 – Solicitação de Conexão de um cliente à um servidor.

O código a seguir é uma implementação simples de um cliente que pode nos fornecer os requerimentos necessários. Mantenha em mente que estes códigos estão focados em conexões UDP, portanto utilizando *datagramas* ao invés de um modelo *streaming* de TCP cliente-servidor.

```
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;

    if ((rv = getaddrinfo(SERVERIP, SERVERPORT, &hints, &servinfo)) != 0)
{
        return 1;
    }

    // loop through all the results and make a socket
    for(p = servinfo; p != NULL; p = p->ai_next) {
        if ((sockfd = socket(p->ai_family, p->ai_socktype,
                p->ai_protocol)) == -1) {
            perror("talker: socket");
            continue;
        }

        break;
    }

    if ((numbytes = sendto(sockfd, argv[2], strlen(argv[2]), 0,
             p->ai_addr, p->ai_addrlen)) == -1) {
        perror("talker: sendto");
        exit(1);
    }
```

De fato é preciso estar muito claro que a solicitação não necessariamente é feita ao mesmo Server que irá "fechar" a conexão. É comum que quando um servidor reconhece uma solicitação, ele invoca um *fork*, criando outro processo que então ira gerir a conexão (como podemos evidenciar na Figura 32 a seguir).

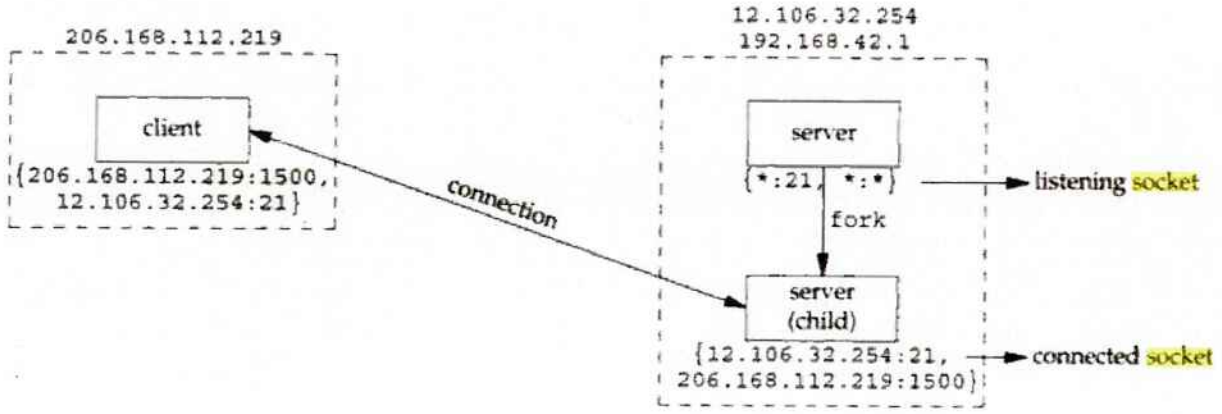


Figura 32 – Servidor repassa a gerenciar da conexão para um "filho"

Este método é então repetido toda vez quem um novo cliente tenta criar uma nova conexão com o servidor (Figura 33). Desta forma o servidor pode responder individualmente para cada cliente e tem o controle de todos os envios de mensagens e recebimento. É fortemente sugerido o uso de design patterns [XIII] como Singleton para o gerenciamento de múltiplas threads na aplicação.

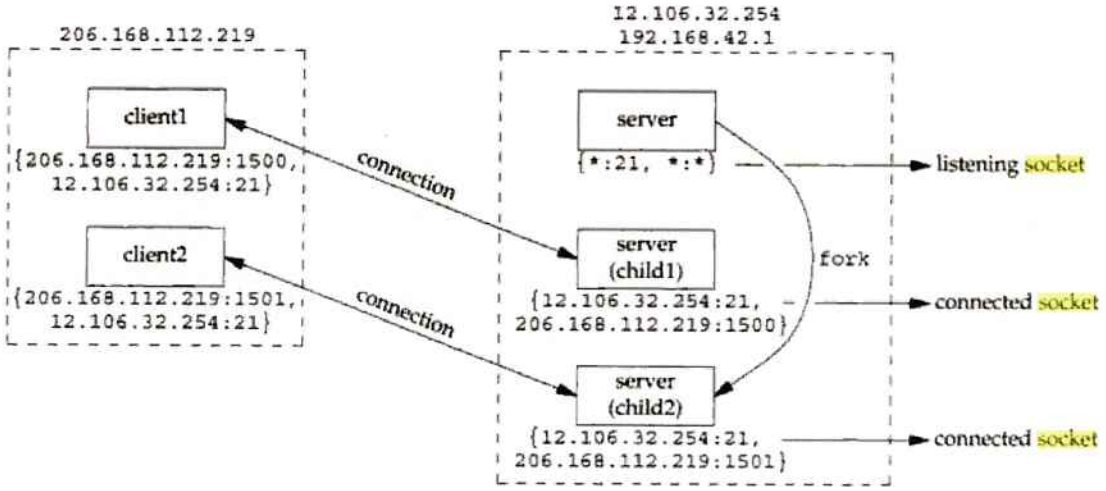


Figura 33 – Uma segunda solicitação de conexão com o servidor.

3.3 Organizações de Seções de Rede

As seções são organizadas em quatro níveis: Servidor, Mundo, Lobby e Salas, conforme o esquema a seguir:

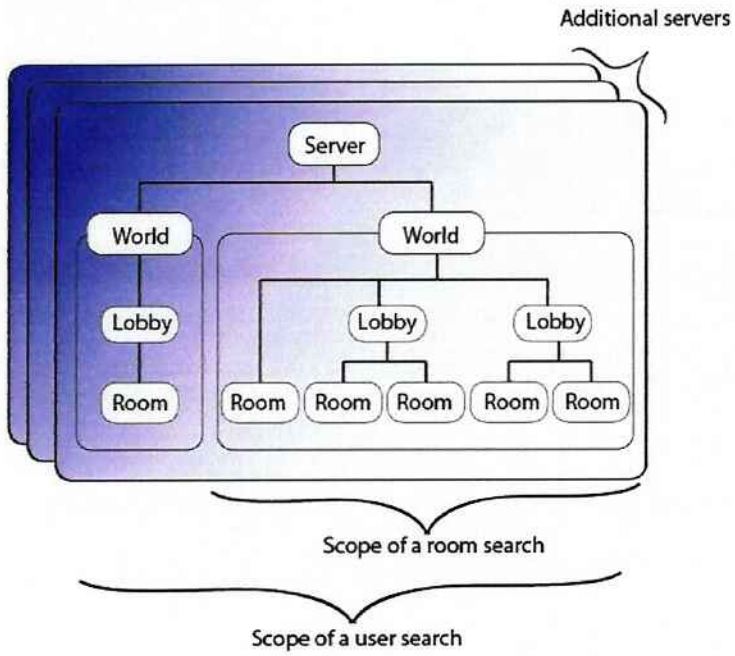


Figura 34 – Componentes de um Sistema de Redes

User: Refere-se ao usuário do sistema de rede.

Server: O servidor compreende o nível mais alto de um sistema de rede. Ele fornece as funções de *Matchmaking* para ser utilizadas nos aplicativos. Um ou mais servidores podem ser alocados para cada aplicação.

Server Context: O contexto de um servidor de um sistema de rede precisa ser criado por um usuário quando este acessa o sistema. Um usuário que criou um contexto será consequentemente apto a acessar o sistema de rede.

World: Um mundo representa o espaço de um M*atchmaking*. Funções de sistemas de rede (como obter Lobbies ou procurar Salas) são executadas por lobbies e salas, respectivamente, quando pertencentes a um mundo.

Session: Uma seção é o termo coletivo para lobbies e salas.

Lobby and Lobby Rooms: Um lobby pertence ao mundo. É um espaço que o usuário pode entrar, comunicar-se com outros membros interno, e fazer uso de funções de *matchmaking.* Um lobby fornece funções de comunicação entre os membros, podendo ter mais membros que salas.

Estes diversos níveis permitem o desenvolvimento de diferentes configurações para oferecer a melhor experiência para um *gamer* especificando-se de acordo com o gênero do jogo. Por exemplo, podem ser utilizadas três configurações básicas: "Procurando por uma Sala", "Utilizando Lobbies" ou "Utilizando Todos os Níveis", como será visto em seguida.

3.3.1.1 Procurando por uma Sala

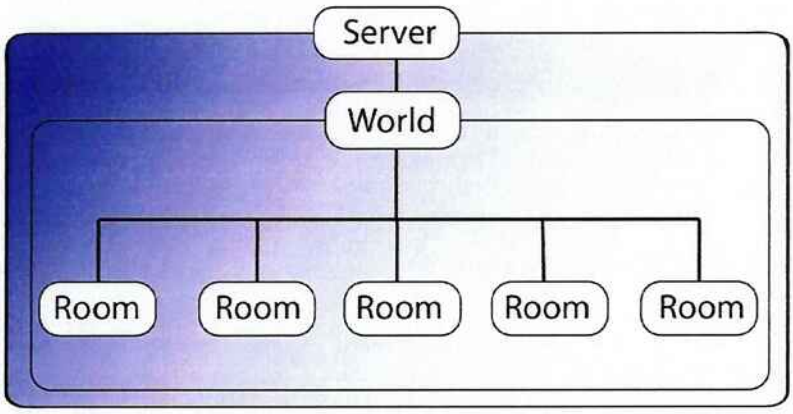


Figura 35 – Configuração quando usando Funções de Procura de Sala

Nesta configuração o usuário não é ciente de quantos mundos ou servidores são utilizados. É a configuração mais próxima do *Matchmaking* feito automaticamente pelo TrueSkill (ver Seção 2.1.4.). Basicamente o usuário precisa criar um certo número de atributos para então procurar uma sala compatível com estes atributos.

O sistema então responde com uma lista de quantas salas satisfazem o critério estabelecido na busca, dando a resposta final para o jogador selecionar a sala desejada. Neste ponto, informações relevantes, além daquelas fornecidas na busca, são exibidas para uma escolha de sala mais adequada.

Do ponto onde um usuário é conectado a uma Sala, é possível estabelecer uma conexão peer-to-peer com outros usuários, e então trocar dados relativos ao jogo. Na Figura 36 abaixo é demonstrado o esquema que deve se assimilar à operação básica de procura por uma Sala.

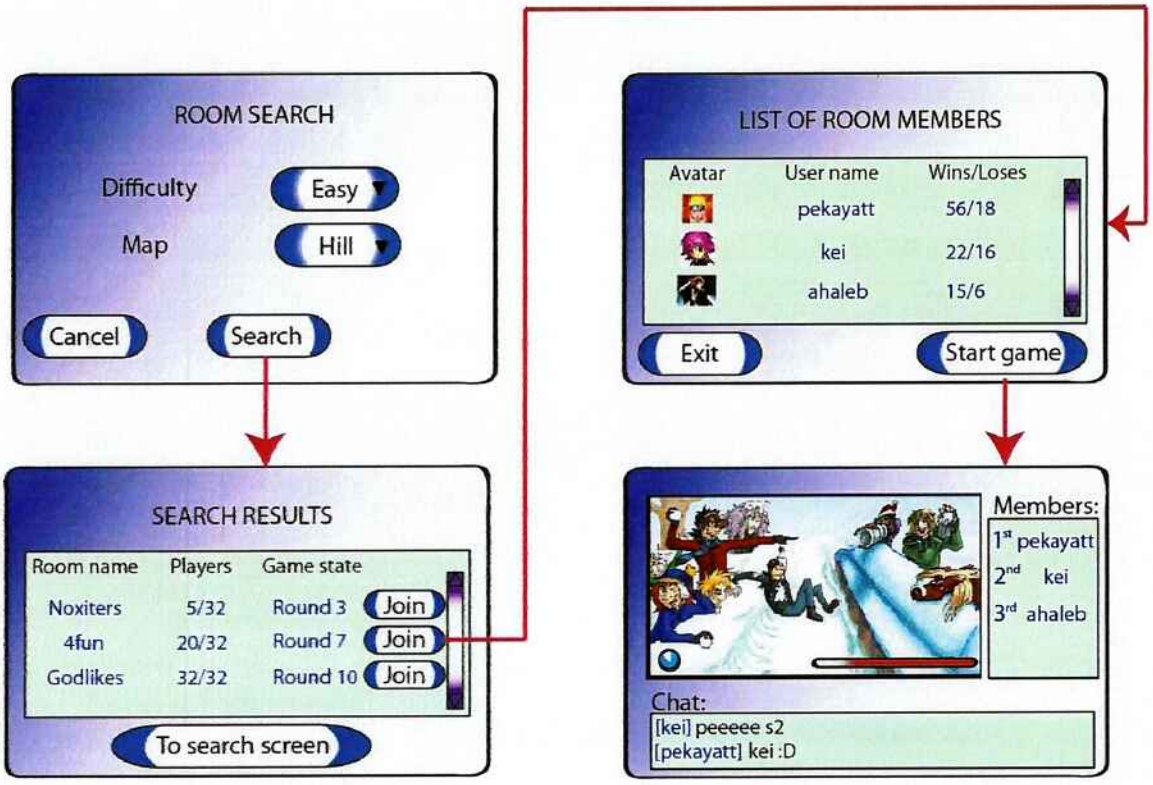


Figura 36 - Design de uma Aplicação que Utiliza a função de Busca por uma Sala

3.3.1.2 Utilizando Lobbies

Os *Loobies* estão em um nível acima daquele das Salas, isto é, um usuário primeiro se conecta a um *Looby* e então poderá entrar em uma Sala. Um exemplo muito popular desta configuração pode ser encontrado no sistema Battle.net [XIV], que foi criado pela empresa Blizzard para suportar a rede de jogadores de um dos mais populares jogos de estratégia.

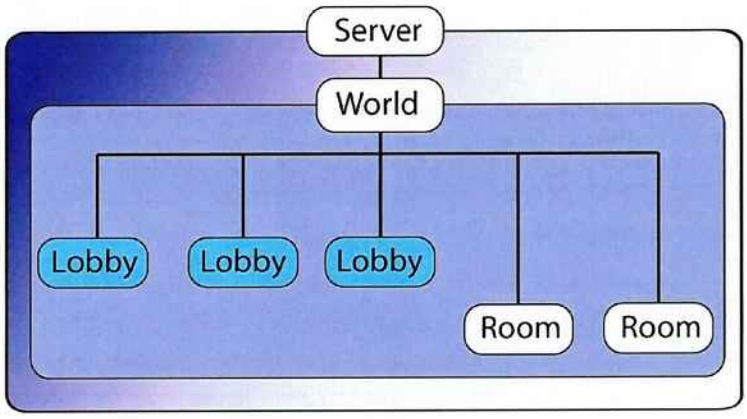


Figura 37 – Configuração quando utilizando Lobbies

Depois da seleção de um *Lobby*, o usuário poderá ser capaz de ver outros usuários que estiverem conectados aquele *Lobby* e então interagir entre eles, conversando e obtendo informações de cada um. A opção de criar Salas deve estar disponível, e da interface de criação de Salas outro usuários podem ser convidados à nova sala.

Deste ponto em diante as negociações acontecem de forma semelhante às anteriores (Seção Procurando por uma Sala) e os jogadores na mesma sala conseguem iniciar conexões P2P e iniciar a troca de dados. A Figura abaixo exemplifica com um esquema de procedimentos esperados.

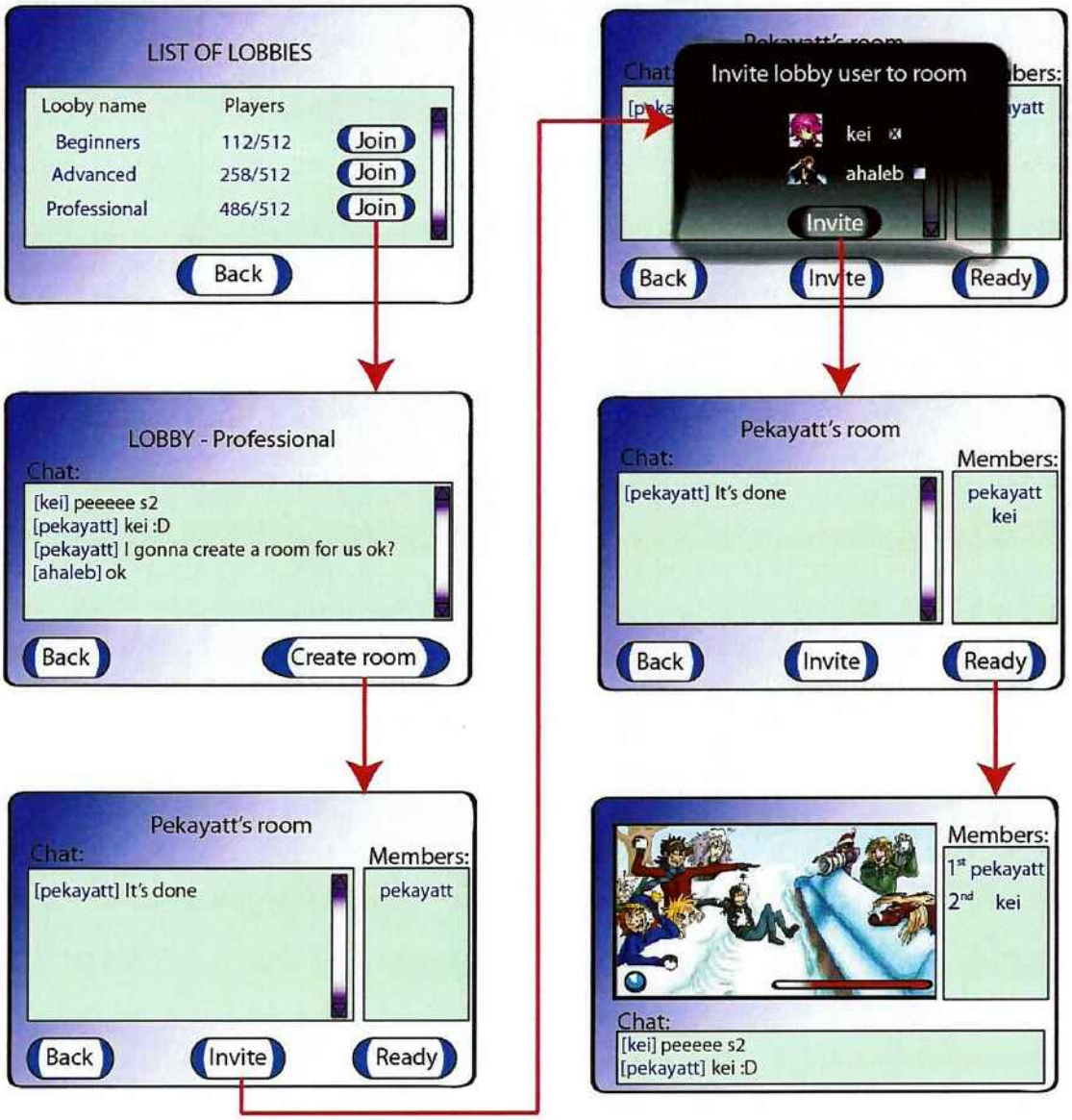


Figura 38 – Design de uma Aplicação Utilizando *Lobbies*

3.3.1.3 Utilizando Todos os Níveis

Este tipo de configuração é dificilmente recomendado (demonstrado na Figura 39), devendo ser utilizado somente se o jogo está sendo projetado para um número enorme de jogadores online. Por exemplo, MMORPG (Massively Multiplayer Online RPG) o qual gerencia milhares de jogadores ou até jogos de tiro muito populares, requerem uma latência muito baixa para jogar.

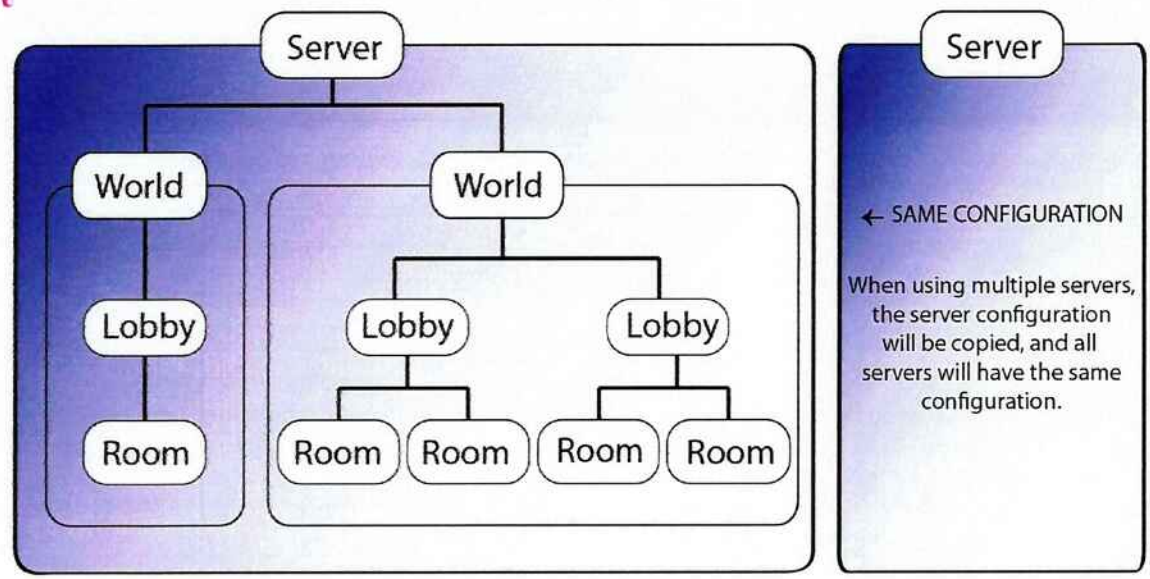


Figura 39 – Configuração quando Utilizando Todos os Níveis.

Desta forma o servidor pode ser criado para distinguir diferentes localidades, como pode ser observado na Figura 40, com a finalidade de organizar jogadores com baixas latências próximos um do outro. Além disto, *Loobies* que são nomeados pelos níveis dos jogadores (iniciantes ou profissionais) ou diferentes lugares visuais podem aprimorar a experiência do usuário.

O jogador então deve selecionar um servidor, um mundo e uma seção para jogar, nesta ordem:

- Selecionar um servidor da lista de servidores
- Selecionar um Mundo da lista de Mundos daquele servidor
- Selecionar um *Looby* da lista de *Lobbies* daquele Mundo
- Criar uma Sala que pertença ao *Lobby*, ou selecionar e entrar em uma Sala através de uma pesquisa das Salas daquele *Looby*.

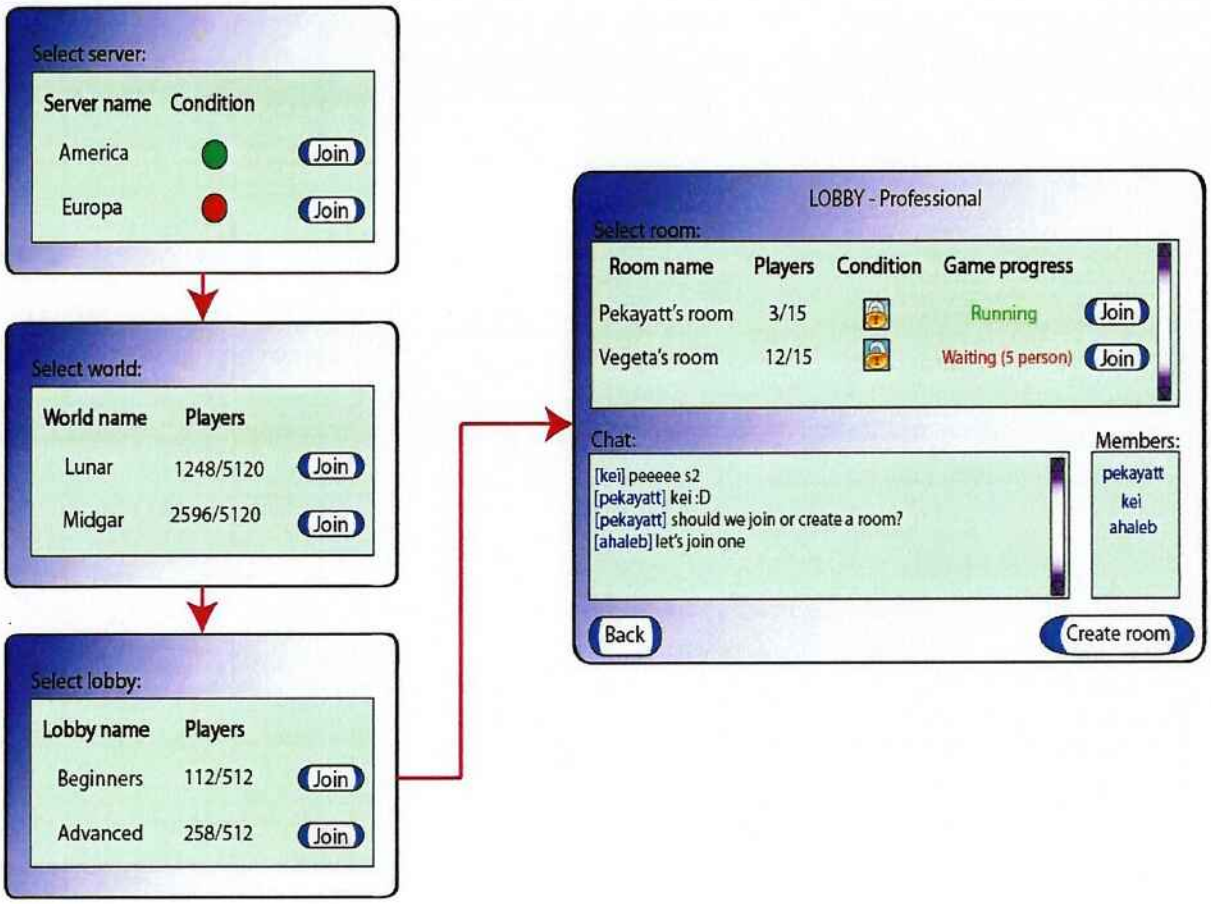


Figura 40 - Design de uma Aplicação – Utilizando Todos os Níveis

3.4 Processing

Para o desenvolvimento do nosso cliente, no exemplo de comunicações partimos então para o estudo de diversas APIs que se demonstravam facilitadoras no processo de desenvolvimento de um rápido protótipo. Entre elas algumas se destacavam como o Microsoft XNA [XV], o Adobe Flash [XVI] e o Processing [XVII].

Tendo em conta o cunho do desenvolvimento em código-livre deste exemplo, optamos pelo Processing já que este detém de todas as ferramentas de desenvolvimento gratuitas e livres, além de ter sido sujeito de práticas em um dos cursos ministrados na universidade pelo Prof. Dr. Ricardo Nakamura, criando então uma familiarização maior com a ferramenta por si só.

O processo de instalação do Processing é extremamente simples e eficaz, apesar de contar com a disponibilidade de ser utilizado em diversas plataformas, como Windows, Linux e Mac, neste trabalho apenas utilizamos a versão para Windows 32bits. Foi importante também notar que o Processing possui uma IDE de edição de códigos própria (como visto na Figura 41), o que facilita desenvolvimentos curtos de rápida prototipagem, entretanto a API também permite ser adicionada a projetos Java na IDE Eclipse [XVIII] possibilitando uma maior organização e desenvolvimento de projetos de maior escala.

Em meio ao desenvolvimento de uma aplicação para exemplificar as características de um jogo em rede foi possível encontrar uma biblioteca interna ao Processing, chamada Vanilla, a qual em teoria supre as necessidades de um desenvolvimento multi-jogador, através de processos semelhantes aos descritos neste trabalho.

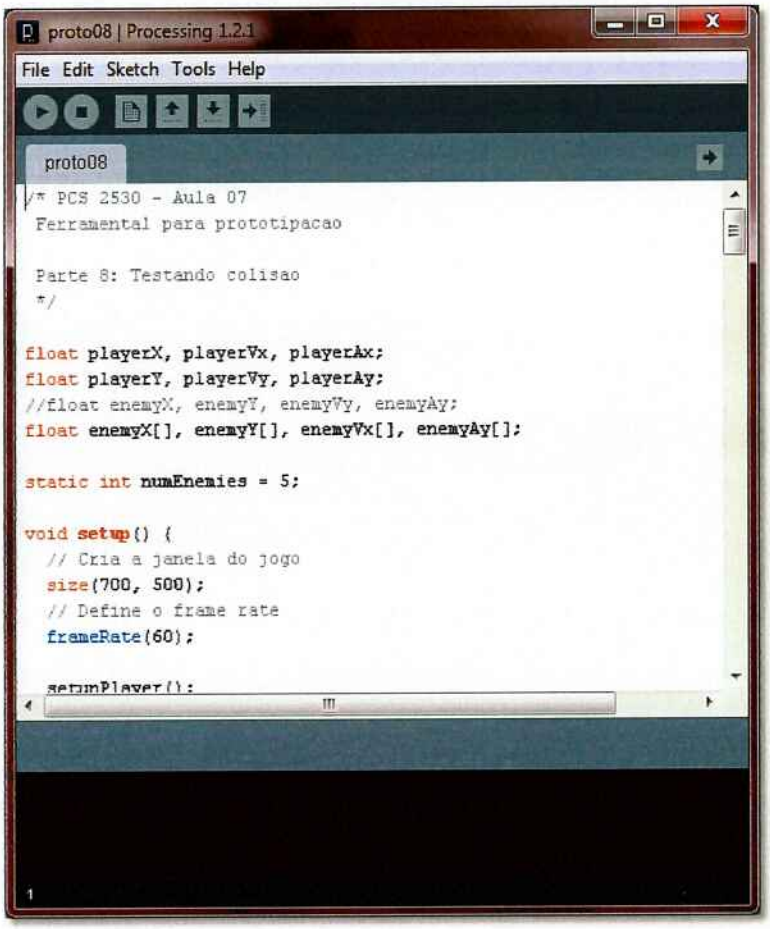


Figura 41 - IDE fornecida pela API Processing

Então foi decidido que uma extensão do protótipo seria elaborada de modo que se permitisse utilizar da biblioteca padrão do Processing, então a avaliando e verificando se os conceitos estudados neste trabalho podem ser aplicados a ela (a biblioteca). Não excluindo o desenvolvimento de meios suficientes e necessários para fazermos as conexões acessando as opções de criação de socket de baixo nível.

Nos Apêndices A e B um código em Java de um protótipo desenvolvido com o uso do Processing e de sua biblioteca de rede pode ser encontrado. Como pode ser visto o processo é simples e bem estruturado, porém notamos diversos problemas quanto ao desempenho da biblioteca de rede. Nosso protótipo é um jogo simples onde os jogadores podem colidir utilizando um objeto controlável com outros objetos que são automaticamente criados.

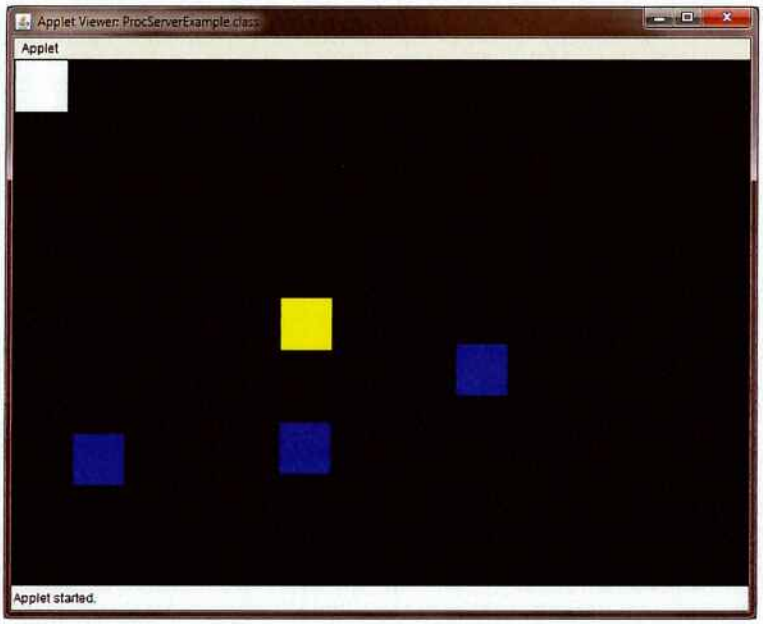


Figura 42 - Protótipo de Jogo: O quadrado amarelo é o personagem controlável, o branco o remoto, e os azuis automaticamente criados.

Para este projeto então foram definidas duas aplicações, uma primeira que serviria como o servidor, criando os objetos automaticamente e abrindo um servidor em uma porta desejada, esperando a conexão do cliente. Por sua vez a aplicação cliente tem a estrutura semelhante a do servidor, porém ela lê por meio da conexão de rede a posição dos objetos criados pelo servidor, que é enviado em uma *string* através da rede, e respondendo com a posição de seu jogador (o objeto que pode ser movido).

Por exemplo, se o jogo está rodando com uma taxa de quadros por segundo superior a 15 *fps* (quadros por segundo), as mensagens começam a se perder e consequentemente ambas as aplicações não conseguem interagir, acarretando em um erro fatal que fecha a conexão e o cliente trava. Partimos então para a elaboração de nosso sistema implementando as conexões em *socket* de baixo-nível.

Para tal foram utilizados de vários exemplos disponíveis na internet, como os quais já estudados na Seção 3.2 (Exemplos de conexão Socket), e foi programado por meio das bibliotecas padrão do Java uma conexão Socket seguindo a topologia cliente-servidor. É importante deixar claro que o servidor é indicado de ser utilizado em uma máquina mais potente, justo, pois a implementação mais adequada é tomando uso de múltiplas threads, para poder carregar as informações de rede sem a interrupção do ciclo de funcionamento normal do jogo.

4 RESULTADOS

Nesta seção alguns resultados obtidos pelos estudos aqui apresentados e das implementações desenvolvidas são especificados.

4.1 A Biblioteca de Rede

Nesta seção será explicado o trabalho desenvolvido na empresa Novecento Games, entretanto alguns detalhes não podem ser revelados, tanto devido a Propriedade Intelectual da companhia quanto pela informação fazer parte de documentos oficiais da SCE (Sony Computer Entertainment) ou da Microsoft.

Avançando, neste projeto foi criada uma biblioteca de rede que pudesse estender as funcionalidades de uma versão modificada da PhyreEngine. PhyreEngine é um motor de jogos de alta qualidade distribuído pela SCE para seus desenvolvedores oficiais e é aberta a modificações, como pode ser visto em [XIX]. Alguns dos jogos que foram construídos através deste motor que valem a pena serem mencionados são: Flower, Race Driver GRID e DiRT (sendo que os dois últimos utilizaram uma versão modificada chamada EGO Engine).

Figura 43 – O jogo Flower da thegamecompany® utilize a PhyreEngine

Sem entrar em mais detalhes, pode ser dito que esta *engine* é projetada de uma maneira muito modular, em especial ela utiliza um método para incluir diferentes funcionalidades em sua forma padrão de atuação, tomando vantagem de uma biblioteca de *Utilities* (Utilidades).

Cada *Utility* estende as funcionalidades da *engine* com a finalidade de fornecer funções necessárias e métodos que satisfaçam o desenvolvimento de um jogo. No caso deste projeto a biblioteca de Rede foi desenvolvida como uma *Utility* do PhryreEngine, tendo como foco primário o PS3 e posteriormente o Xbox 360.

Primeiro deixemos claro que o PlayStation 3 é uma das plataformas mais complicadas para se programar. O CEO da SCE Kaz Hirai disse em 2009: "Nós não fornecemos o 'fácil de programar com' console que os [desenvolvedores] querem, porque 'fácil de programar com' significa que qualquer um pode tirar proveito de tudo que o hardware pode fazer, então a questão seria o que fazer pelos próximos nove anos e meio?". (Tradução Livre)

Hirai explica na entrevista, que foi feita na edição de Fevereiro de 2009 da Official PlayStation Magazine, que a razão de ser tão complicado era porque o console tinha muito potencial: "É algo como – I não falarei espada de dois-gumes – mas é difícil de programar para, e muitas pessoas vêm o pontos negativos disto, mas se você inverter, isso significa que o hardware tem muito a oferecer". (Tradução Livre).

Entretanto, felizmente a programação de rede no PlayStation 3 é muito similar aos exemplos de Socket que foram explicados na Seção 3.2, por outro lado o sistema de Matchingmaking (Seção 2.2.3) é bem específico e complicadamente baseado em chamadas de sistema (*callbacks)* e foi projetada para funcionar como um conjunto de chamadas em forma de máquina de estados (semelhante a Figura 42) [XX].

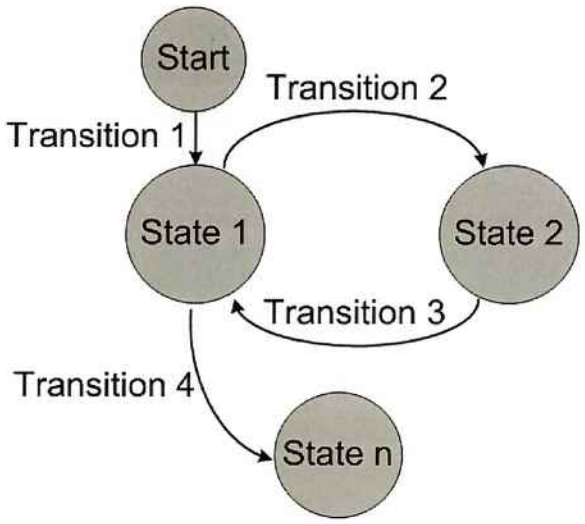


Figura 44 – Um Exemplo de uma Máquina de Estados Finita (FSM)

Isto significa que para conseguir unir as funcionalidades do *Matchingmaking* mais aquelas de *signaling* (troca de mensagem por si mesma) em uma *Utility* seria necessário incluir uma máquina de estados finita dentro desta, ou eliminar algumas funcionalidades para automatizar a troca dos estados da máquina.

A escolha feita foi a segunda, o desenvolvimento da biblioteca seguiu de uma forma a apresentar dois níveis principais: uma interface, a qual administraria a chamada desejada do jogo; e uma máquina de estados em segundo plano, que estaria lidando com a mudança de estados e as *callbacks* específicos da plataforma. Mais tarde, esta escolha demonstrou ser ainda mais conveniente para ser aplicada à arquitetura da Xbox Live sem muitos problemas.

A *Utility* foi então escrita se baseando em *stream* (fluxo de mensagens), de forma com que funcionaria como uma conexão de um fluxo de rede, quando o desenvolvedor do jogo desejasse abrir uma conexão ele simplesmente invocaria a função "*open*", esta função por sua vez se encarregaria do processo de *login* e de estabelecer uma conexão com um servidor da plataforma.

Deste modo, os passos seguintes são necessários para se conectar a outro jogador: buscar ou juntar-se a uma sala (a configuração de rede é tal que o servidor utiliza a Sala exatamente abaixo do Mundo como detalhado na Seção 3.3); procurar um jogador; tornar-se pronto para trocar mensagens com ele. O método *open,* então, recebe uma *flag* que permite com que o desenvolvedor peça para automaticamente juntar-se/criar Sala, desta forma, logo após o *login,* a *Utility* procuraria por uma Sala e se conectaria à mais apta, do contrário se nenhuma Sala fosse encontrada, uma nova seria criada esperando o próximo jogador se conectar.

Não obstante, neste exato ponto é importante ser capaz de obter informação do atual estado do sistema, em outras palavras obter o número de salas e os atributos de cada uma delas. Para esta específica função, o método *listRoom* foi criado, assim como uma nova classe interface que pudesse organizar os atributos principais de cada Sala encontrada.

Além disso, dentro da classe *Room* uma lista dos jogadores é exposta, sendo que a informação sobre os jogadores conectados à Sala pode ser muito útil ao desenvolvedor do jogo, desde obter o nível de habilidade médio até a simples razão de exibir o nome do jogador ao usuário.

Semelhantemente, outra classe foi criada com a finalidade de obter a informação do jogador e seus atributos principais e o método *listPlayers* foi adicionado a classe da sala. Como resultado, o desenvolvedor de jogo tem a capacidade de conectar ao servidor da plataforma, listar as salas disponíveis, e ao selecionar uma sala ele pode exibir uma lista de jogadores daquela sala e obter seus atributos.

Depois do programador do jogo completar as ações necessárias para entrar na Sala, os métodos *send* e *read* estão disponíveis, enviando uma mensagem Broadcast para todos os jogadores que compartilham aquela sala. O método recebe como parâmetros um ponteiro *void* e o tamanho do tipo de dado a ser enviado, isto torna possível ao desenvolvedor do jogo trocar quaisquer que sejam as estruturas de dados desejadas.

Um exemplo desta funcionalidade é tal que o programador do jogo poderia construir uma estrutura de dados que possui uma variável ID com o tipo *byte*, e outra *message* como um vetor limitado de *chars*. Então estes dados seriam enviados especificando o tamanho da mensagem através do *sizeof* da estrutura, definindo a quantidade exata de bits que seria enviada pelo pacote da mensagem.

Outra funcionalidade que foi implementada na *Utility* foi a possibilidade de enviar mensagens diretamente a usuários, utilizando o tipo *Player* obtido através do comando *playerList*. Desta forma, outra arquitetura de mensagens pode ser utilizada, como visto na Seção 3.1, gerando maior liberdade ao designer do jogo. É importante manter em mente que estas são interpretações lógicas e não necessariamente a mensagem por si mesma poderá ser entregue de outras maneiras, já que a *Utility* está funcionando em um nível abstrato acima da implementação de rede da plataforma.

Ademais, a leitura de uma mensagem pode ser efetuada especificando um usuário desejado para o recebimento, adicionando opções avançadas de recolhimento de informação que pode ser essencial principalmente em processos de *hadshanking* quando um usuário acessa uma Sala.

Depois do término da biblioteca, um exemplo foi adicionado com as modificações necessárias para que um jogador conectasse a outro PlayStation 3, com um usuário em uma mesma Sala, e pudesse tomar controle de uma animação.

A rede em questão estava sob a configuração de uma mesma máscara de sub-rede, em outras palavras em Rede Local e, mesmo com a PlayStation Network sendo necessária para o acesso do *Matchmaking* e, portanto, a Internet, tudo ocorreu conforme o planejado.

Então o cenário foi alterado para um mais próximo da situação em vida-real, um dos consoles estava conectado a uma VPN localizada a mais de 100 km e utilizando outro ISP (Provedor de Internet), a latência entre os dois IPs era de em torno 70 a 100ms, com picos de 300ms. Quando o teste foi submetido a este cenário, a *Utility* começou a ter problemas de reconexão, após jogar uma partida o usuário dificilmente conseguia se conectar para uma nova partida com o mesmo jogador.

Foi então encontrado um problema na *Utility* quando esta fechava a aplicação (ou reiniciava o console). A questão era o não fechamento da conexão *socket* que não é feita automaticamente, e foi simplesmente resolvida através da adição de um método na destruição (*destructor)* para fechar a conexão *socket.*

Também, para estar seguro que nenhuma mensagem seria perdida em caso do remetente não ser capaz de processar a mensagem antes da chegada de uma nova (chamada do método *read*, antes da mudança do vetor de leitura de dados), uma lista circular foi implementada utilizando um vetor *void* de tamanho limitado, mantendo em memória o número de dados diferente de zero definido por um *define* (padrão eram dez). Desta forma uma lista seria atualizada a cada nova mensagem e seria limpa quando a função *read* da *Utility* fosse chamada.

Além do mais, a *Utility* uma vez completa e devidamente testada foi adaptada para funcionar também com a biblioteca de rede do Xbox. Diferentemente do SDK do PlayStation3, esta possuía um grande número de bibliotecas internas que possibilitaram um progresso mais suave, incluindo até mesmo uma biblioteca de rede de alto nível (*high level language*).

Desta forma a *Utility* pode ser utilizada com a mesma interface do PlayStation3, apesar de adicionado uma nova camada para o Xbox entre a conexão socket de baixo nível e a *Utility.* Esta interface teve como propósito principal simplificar ainda mais o conjunto de ações necessárias pela *Utility* de rede e se tornou o *middle-ware* entre as bibliotecas do Xbox e nosso motor de jogos (*engine).*

Em adição, a *Utility* para o Xbox teve de alterar algumas propriedades das classes *Room* e *Player*, já que os atributos principais e a interface continuam sendo os mesmos, por óbvias razões de compatibilidade, mas conceitos como o Id do Jogador foram mudados do *login* da PSN para o equivalente na Xbox Live (ver Seção 2.1 e 2.2).

O projeto em questão foi iniciado analisando também a possibilidade de portar a *Utility* uma vez mais para o sistema do Wii, o qual se demonstrou apto a receber tais modificações assim como as duas implementações atuais demonstraram. Infelizmente devido a problemas de ordem temporal este processo não pôde ser terminado, pausando o desenvolvimento no estudo da rede da NWC.

4.2 Demo em Processing

O Processing se mostrou uma API muito boa e prática, sendo de grande facilidade a construção de protótipos de jogos principalmente para mensurar a qualidade de resposta de comandos de entrada e de design de níveis. Funções geralmente mais elaboradas como definir a taxa de quadros por segundo (*framerate*) ou desenhar formas geométricas são simplificadas a chamadas de um método, agilizando muito o desenvolvimento.

Infelizmente a biblioteca Vanilla [XXI], inclusa na linguagem Processing não se mostrou eficiente o necessário para nosso protótipo simples, pecando na responsabilidade e eficiência. Demonstrando mais uma vez a importância de elaboração de boas bibliotecas de rede.

A escolha da criação de rotinas de conexão através das bibliotecas padrão Java se demonstrou interessante, sendo que um maior trabalho para o funcionamento adequado foi gasto, entretanto uma melhoria de performance, e a capacidade de alteração de pormenores esclareceu as vantagens e desvantagens desta opção.

Como podemos ver na Figura 45, as aplicações funcionaram satisfatoriamente conforme previsto. Neste teste a aplicação de servidor era executada localmente, enquanto a aplicação cliente também, de maneira que as conexões foram feitas apenas na mesma máquina, a critério de testes. No entanto o protocolo estabelecido se mostrou suficiente, necessário e aplicável a redes externas, com as devidas configurações (como *Port Forwarding*).

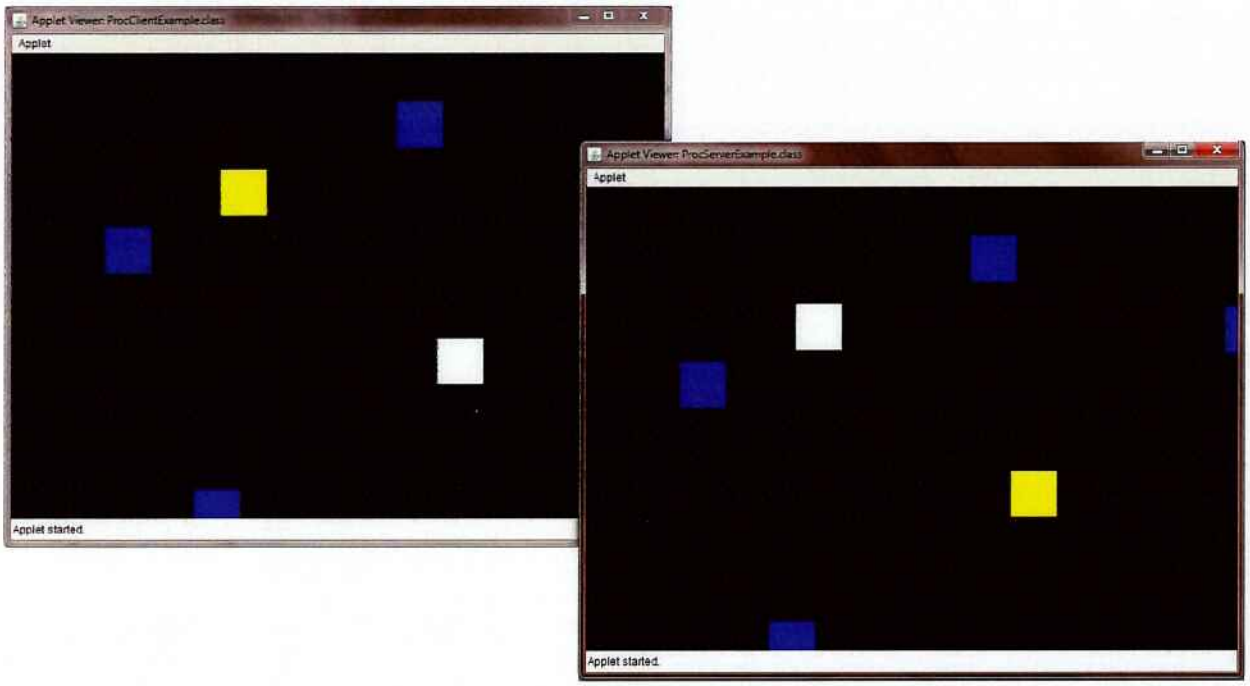


Figura 45 - Protótipo Simples de Cliente e Servidor sendo executados simultaneamente

5 CONCLUSÃO

De nossa análise inicial, não foi muito complicado selecionar as principais características dos sistemas que se sobrepunham. De um lado havíamos um sofisticado *Matchmaking* (e classificação de habilidades) algoritmos rodando no sistema da Live, e do outro lado um número bem definido de configurações de salas e *layouts* definidos pela PlayStation Network, característica muito interessante para nossa aplicação.

Mesmo quando ambos os sistemas são comparados em seu grupo total de funcionalidades, eles continuam similares, como podemos ver na Tabela 5. Cada um tenta realçar uma característica "única", como o PSN Home ou a comunicação por voz in-game do Xbox Live.

	Feature	XBOX LIVE	PLAYSTATION.Network
	Price	About $50/yr	Free
Communications	In-Game Menu	Yes (Mini Dashboard)	Yes (XMB)
	In-Game Friends List	Yes (Mini Dashboard)	Yes (XMB)
	In Game Private Voice Chat	Yes	No
	Text Chat (anywhere)	Yes	No
	Private Messaging	Yes	Yes
	Friends List	Yes	Yes
	Chat Sessions utilizing a headset and webcam?	Yes	Yes
Purchases and Downloads	Downloads Venue	Yes (Marketplace)	Yes (PS Store)
	Addons and Demos?	Yes	Yes
	Music (MP3's, **NOT Rock Band/GH**)	Yes	Limited
	Video/Trailers/Movies/TV?	Yes	Yes
	Photos/Wallpaper/Themes?	Yes	Yes
Enhanced Community	3D Avatar System	Upcoming (XBL Experience)	Upcoming (HOME)
	Full 3D Customized Spaces?	No	Upcoming (HOME)
	Take characters into Games?	Upcoming (XBL Experience)	Unknown
Other	Accomplishment Tracking	Yes (Achievements)	Yes (Trophies)
	Leveling System?	No	Yes

Tabela 5 - PSN vs. Xbox Live comparação de funcionalidades (Leimeisel 2008)

Infelizmente outros sistemas não puderam ser diretamente comparados devido ao propósito e foco diversos. Por exemplo, a Nintendo Wi-Fi Connection não é responsável por muitas das funcionalidades que tanto PSN quanto Live afirmam ser essenciais, ao mesmo tempo Nintendo desenvolveu outros sistemas para suportar seus clientes, como a NintendoWare ou o Virtual Console.

Outro bom ponto a ser ressaltado é que mesmo com os consoles tendo uma arquitetura de hardware muito diferenciada, eles não são máquinas de propósito genérico como PCs (computadores pessoais); eles ainda seguem os mesmo princípios básicos das redes. Isto é devido ao fato da Internet ser organizada de acordo com os mesmos protocolos seja onde for que ela seja acessada.

Então, todas as arquiteturas e estudos alcançados em sistemas de computadores podem ser aplicados no desenvolvimento de uma biblioteca de rede para consoles. Todavia a escolha de um gênero de jogo é realmente importante a ser considerado na biblioteca dado o *trade-of* entre as latências e a concisão dos dados bem definidos para cada escolha.

Mesmo assim se demonstrou possível criar um *middleware* bem balanceado em uma camada de abstração acima daquela especificada pelas bibliotecas dos consoles que pode gerenciar os básicos de rede. O desenvolvimento bem estruturado do *middleware* pode aprimorar o processo de desenvolvimento de um jogo, em troca de algum desempenho.

No cenário atual é claro que os jogos online não podem continuar sendo ignorados. Não apenas os maiores nomes nos consoles estão investindo neste campo, mas também muitos sistemas começam a aparecer em outras plataformas de jogos. Destes, alguns merecem ser citados como: o Steam [XXII] (Windows e Mac OS) ou até mesmo a Apple Store (iPhone). Assim como alguns novo esforços também têm sido tomados no sistema de redes na nuvem [XXIII], como o sistema OnLive e o Gaikai, os quais oferecem a possibilidade de jogar qualquer jogo em qualquer lugar e em qualquer plataforma.

Enfim se concluem que o desenvolvimento de protótipo para exemplificação de conexões de Rede foram processos realmente efetivos, aonde as teorias apresentadas neste trabalho foram aplicadas de forma concisa e direta.

As ferramentas oferecidas pela linguagem Processing são de extrema valia, porém claramente ainda um trabalho em desenvolvimento, sendo altamente indicado a utilização de IDEs mais maduras como o Eclipse. Assim como é indicada a utilização (ou desenvolvimento) de uma biblioteca aparte daquela fornecida pela linguagem, primordialmente devido à simplicidade e falta de desempenho oferecidos.

Assim, como futuro desenvolvimento deste trabalho, se propõe o desenvolvimento de uma biblioteca, que não só possa elaborar a troca de mensagens entre os usuários, mas também criar um nível de organização de rede (assim como supracitado na Seção 3.3) para a criação de Salas e *Loobies*. Sendo indicada principalmente a configuração "Utilizando Lobbies", devido à experiência obtida no trabalho executado na empresa Novecento Games.

REFERÊNCIAS

[I] - Doom 3 Team Development Story - http://www.youtube.com/watch?v=X8xvTJ6msqY – Acesso em Dezembro de 2010

[II] - "Cost of making games set to soar" - *BBC News*. November 17, 2005.

[III] - Id Softare. Disponível em:< www.idsoftware.com> - Acesso em: Dezembro de 2010.

[IV] - Ralf Herbrich, Tom Minka, and Thore Graepel - TrueSkill TM: A Bayesian Skill Rating System, 2007

[V] - "Official PlayStation Website: PlayStation Network - PSN" - http://uk.playstation.com/psn/ - Acesso em Dezembro de 2010

[VI] - Ondrejka, Cory R. - A Piece of Place: Modeling the Digital on the Real in Second Life, 2004

[VII] - S. Singhal and M. Zyda - Networked Virtual Environments: Design and Implementation. Addison Wesley, 1999

[VIII] - Mark Claypool - Network Characteristics for Server Selection in Online Games

[IX] - Michael R. Macedonia and Michael J. Zyda - "A taxonomy for networked virtual environments," IEEE Multimedia, vol. 4, no1, pp. 48–56, 1997.

[X] - Steve Deering - "Host extensions for IP multicasting," - Internet RFC 1112, Aug. 1989, http://www.ietf.org/rfc/rfc1112.txt - Acessado em Dezembro de 2010

[XI] - P. Srisuresh & M. Holdrege - IP Network Address Translator (NAT) Terminology and Considerations – Internet RFC 2663, Aug. 1999

[XII] - Brian Hall - Beej's Guide to Network Programming, 2009

[XIII] - Stephen D. Huston, James C. E. Johnson, Umar Syyid - The ACE programmer's guide: practical design patterns for network and Systems Programming, 2004

[XIV] - Blizzard Entertainment. *Battle.net* - http://www.battle.net, 2010

[XV] - Aaron Reed - Learning XNA 3.0, 2009

[XVI] - Annesa Hartman - Exploring Adobe Flash CS4 ,2009

[XVII] - Reas, Casey; Fry, Ben - Getting Started with Processing, 2010

[XVIII] - Ed Burnette - Eclipse IDE, 2005

[XIX] - "SCEI March 2009 (GDC) PhyreEngine Press Release" – SCEE - Mar 24, 2009.

[XX] - Minsky, Marvin - Computation: Finite and Infinite Machines, 1967

[XXI] - Net \ Library \ Processing.org -http://processing.org/reference/libraries/net/, Acessado em Dezembro de 2010.

[XXII] - Valve, Inc. - "Steam," http://www.steampowered.com/, 2005.

[XXIII] - A. Shaikh, S. Sahu, M.-C. Rosu, M. Shea, D. Saha - On demand platform for online games - IBM Systems Journal, 2006 – Citeseer

APÊNDICE A – EXEMPLO DE CÓDIGO DE UM CLIENTE DE REDE EM PROCESSING

```
import processing.core.*;
import processing.net.*;

@SuppressWarnings("serial")
public class ProcClientExample extends PApplet {

    int playerX, playerVx, playerAx;
    int playerY, playerVy, playerAy;

    int outPlayerX;
    int outPlayerY;
    int maxOutPlayers = 4;

    Server mServer;
    Client mClient;
    int enemyX[], enemyY[], enemyVx[], enemyAy[];

    static int numEnemies = 5;

    public void setup() {
      // Cria a janela do jogo
      size(700, 500);
      // Define o frame rate
      frameRate(8);

      //Set server
      //mServer = new Server(this, 54321); //App, PORT
      mClient = new Client(this, "127.0.0.1", 54321); //App, IP, PORT

      setupPlayer();
      setupEnemies();
    }

    void setupPlayer() {
      // Jogador
      playerX = (width)/2;
      playerY = height;
      playerVx = 0;
    }

    void setupEnemies() {  //Inicializa Inimigos
      // Inimigo
      enemyX = new int[numEnemies];
      enemyY = new int[numEnemies];
      enemyVx = new int[numEnemies];
```

```
    enemyAy = new int[numEnemies];

    for (int i=0; i<numEnemies; i++) {
      enemyY[i] = (int) random(0, height + 50);
      enemyX[i] = width;
      enemyVx[i] = (int) random(5,20);
    }
  }

  void checkColision() {
    // Testa colisao
    boolean hit = false;

    for (int i=0; i<numEnemies; i++) {
      if ((abs(playerX-enemyX[i]) < 50)  && (abs(enemyY[i] - playerY) <
50))
        hit = true;
    }
    // Preenche a tela
    if (hit)
      background(255, 0, 0);
    else
      background(0);
  }

  void updatePlayer() {
    // Jogador

      //X
    playerVx = (int) (playerVx + playerAx - 0.2*playerVx);
    if (abs(playerVx) < 0.2) playerVx = 0;
    else if (playerVx < -10) playerVx = -10;
    else if (playerVx > 10) playerVx = 10;


    playerX = playerX + playerVx;
    if (playerX < 0) {
      playerX = 0;
      playerVx = 0;
      playerAx = 0;
    }
    else if (playerX > width - 50) {
      playerX = width-50;
      playerVx = 0;
      playerAx = 0;
    }

    //Y
    playerVy = (int) (playerVy + playerAy - 0.2*playerVy);
    if (abs(playerVy) < 0.2) playerVy = 0;
    else if (playerVy < -10) playerVy = -10;
    else if (playerVy > 10) playerVy = 10;
    playerY = playerY + playerVy;
```

```
  if (playerY < 0) {
    playerY = 0;
    playerVy = 0;
    playerAy = 0;
  }
  else if (playerY > height - 50) {
    playerY = height-50;
    playerVy = 0;
    playerAy = 0;
  }
}

void drawPlayer() {

  // Desenha um retangulo amarelo
  fill(255, 255, 0);
  rect(playerX, playerY, 50, 50);
}

void drawEnemies() {
  // Desenha um retangulo azul

  for (int i=0; i<numEnemies; i++) {
    fill(0, 0, 255);
    rect(enemyX[i], enemyY[i], 50, 50);
  }
}

public void draw() {
  // Atualiza nossa simulacao
  updatePlayer();

  updateNetwork();

  checkColision();

  drawEnemies();

  drawPlayer();

  drawOutPlayer();
}


private void drawOutPlayer() {

      // Desenha um retangulo amarelo
      fill(255, 255, 255);
      rect(outPlayerX, outPlayerY, 50, 50);

}

private void updateNetwork() {
```

```
//      Recebe dados do Servidor
        mClient.write(playerX + " " + playerY + " " + "\n");
          if (mClient.available() > 0) {
            String input = mClient.readString();
            input = input.substring(0, input.indexOf("\n"));

            String[] data = (split(input, ' ')); // Parse na mensagem
            // Define o tipo de dado

            if (data[0].contentEquals("player")){
                outPlayerX = Integer.parseInt(data[1]);
                outPlayerY = Integer.parseInt(data[2]);
            }
            for (int i = 0; i < numEnemies; i++) {
                if (data[3*i +3].contentEquals("enemy")){
                    enemyX[i] = Integer.parseInt(data[3*i + 4]);
                    enemyY[i] = Integer.parseInt(data[3*i + 5]);
                }
            }

          }

    }

    public void keyPressed() {
      if (key == CODED) {
        if (keyCode == LEFT) {
          playerAx = -3;
        }
        if (keyCode == RIGHT) {
          playerAx = 3;
        }
        if (keyCode == DOWN) {
          playerAy = 3;
        }
        if (keyCode == UP) {
          playerAy = -3;
        }
      }
    }

    public void keyReleased() {
      if (key == CODED) {
        if (keyCode == LEFT || keyCode == RIGHT) {
          playerAx = 0;
        }
        if (keyCode == UP || keyCode == DOWN) {
          playerAy = 0;
        }
      }
    }
}
```

APÊNDICE B - EXEMPLO DE CÓDIGO DE UM SERVIDOR DE REDE EM PROCESSING

```
import processing.core.*;
import processing.net.*;

@SuppressWarnings("serial")
public class ProcServerExample extends PApplet {

    int playerX, playerVx, playerAx;
    int playerY, playerVy, playerAy;

    int outPlayerX;
    int outPlayerY;
    int maxOutPlayers = 4;

    Server mServer;
    Client mClient;
    int enemyX[], enemyY[], enemyVx[], enemyAy[];

    static int numEnemies = 5;

    public void setup() {
      // Cria a janela do jogo
      size(700, 500);
      // Define o frame rate
      frameRate(8);

      //Set server
      mServer = new Server(this, 54321); //IP, PORT

      setupPlayer();
      setupEnemies();
    }

    void setupPlayer() {
      // Jogador
      playerX = (width)/2;
      playerY = height;
      playerVx = 0;
    }

    void setupEnemies() {
      // Inimigo
      enemyX = new int[numEnemies];
      enemyY = new int[numEnemies];
      enemyVx = new int[numEnemies];
      enemyAy = new int[numEnemies];
```

```
      for (int i=0; i<numEnemies; i++) {
        enemyY[i] = (int) random(0, height + 50);
        enemyX[i] = width;
        enemyVx[i] = (int) random(5,20);
      }
    }

    void checkColision() {
      // Testa colisao
      boolean hit = false;

      for (int i=0; i<numEnemies; i++) {
        if ((abs(playerX-enemyX[i]) < 50) && (abs(enemyY[i] -
playerY) < 50))
          hit = true;
      }
      // Preenche a tela
      if (hit)
        background(255, 0, 0);
      else
        background(0);
    }

    void updatePlayer() {
      // Jogador
        //X
      playerVx = (int) (playerVx + playerAx - 0.2*playerVx);
      if (abs(playerVx) < 0.2) playerVx = 0;
      else if (playerVx < -10) playerVx = -10;
      else if (playerVx > 10) playerVx = 10;

      playerX = playerX + playerVx;
      if (playerX < 0) {
        playerX = 0;
        playerVx = 0;
        playerAx = 0;
      }
      else if (playerX > width - 50) {
        playerX = width-50;
        playerVx = 0;
        playerAx = 0;
      }

      //Y
      playerVy = (int) (playerVy + playerAy - 0.2*playerVy);
      if (abs(playerVy) < 0.2) playerVy = 0;
      else if (playerVy < -10) playerVy = -10;
      else if (playerVy > 10) playerVy = 10;
      playerY = playerY + playerVy;
      if (playerY < 0) {
        playerY = 0;
        playerVy = 0;
        playerAy = 0;
```

```
    }
    else if (playerY > height - 50) {
      playerY = height-50;
      playerVy = 0;
      playerAy = 0;
    }
  }
  void drawPlayer() {

    // Desenha um retangulo amarelo
    fill(255, 255, 0);
    rect(playerX, playerY, 50, 50);
  }

  void updateEnemies() {
    // Inimigo
    for (int i=0; i<numEnemies; i++) {
      ;
      //Limitando velocidade
      if (enemyVx[i] > 20) enemyVx[i] =20;
      enemyX[i] = enemyX[i] - enemyVx[i];
      if (enemyX[i] < 0) {
        enemyY[i] = (int) random(0, width + 50);
        enemyX[i] = width;
        enemyVx[i] = (int) random(5,10);
      }
    }
  }

  void drawEnemies() {
    // Desenha um retangulo azul
    for (int i=0; i<numEnemies; i++) {
      fill(0, 0, 255);
      rect(enemyX[i], enemyY[i], 50, 50);
    }
  }

  public void draw() {
    // Atualiza nossa simulacao
    updatePlayer();
    updateEnemies();

    updateNetwork();

    checkColision();

    drawEnemies();

    drawPlayer();

    drawOutPlayer();
  }
  private void drawOutPlayer() {
```

```
            // Desenha um retangulo amarelo
            fill(255, 255, 255);
            rect(outPlayerX, outPlayerY, 50, 50);
        }

        private void updateNetwork() {
            mServer.write("player" + " " + playerX + " " + playerY + "
");
            for (int i=0; i<numEnemies; i++) {
                mServer.write("enemy" + " " + enemyX[i] + " " +
enemyY[i]+ " ");
              }
            mServer.write("\n");//Cambio DESLIGO!

            mClient = mServer.available();
              if (mClient != null) {
                String input = mClient.readString();
                input = input.substring(0, input.indexOf("\n"));
                String[] data = (split(input, ' ')); // Parse da mensagem
                // Sets outPlayer coords
                outPlayerX = Integer.parseInt(data[0]);
                outPlayerY = Integer.parseInt(data[1]);
              }
        }

        public void keyPressed() {
          if (key == CODED) {
            if (keyCode == LEFT) {
              playerAx = -3;
            }
            if (keyCode == RIGHT) {
              playerAx = 3;
            }
            if (keyCode == DOWN) {
              playerAy = 3;
            }
            if (keyCode == UP) {
              playerAy = -3;
            }
          }
        }

        public void keyReleased() {
          if (key == CODED) {
            if (keyCode == LEFT || keyCode == RIGHT) {
              playerAx = 0;
            }
            if (keyCode == UP || keyCode == DOWN) {
              playerAy = 0;
            }
          }
        }
    }
```