Dennis Akihiko Tanaka

Felipe de Aguiar Kamakura

Luiz Eduardo Freire Emmerich

Rodrigo Carvalhedo Petriche

Desenvolvimento do módulo de redes de um Engine didático

Orientadores: João Luiz Bernardes Jr.
Prof. Dr. Romero Tori

Escola Politécnica da Universidade de São Paulo

São Paulo – 2006

Dennis Akihiko Tanaka

Felipe de Aguiar Kamakura

Luiz Eduardo Freire Emmerich

Rodrigo Carvalhedo Petriche

Desenvolvimento do módulo de redes de um Engine didático

PCS 2040 – Projeto de Formatura I

Orientadores: João Luiz Bernardes Jr.
Prof. Dr. Romero Tori

Escola Politécnica da Universidade de São Paulo

São Paulo – 2006

FICHA CATALOGRÁFICA

Tanaka, Dennis Akihiko
Desenvolvimento do módulo de redes de um Engine didático
/ D.A. Tanaka, F. de A. Kamakura, L.E.F. Emmerich, R.C. Petriche. -- São Paulo, 2006.
132 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Jogos de computadores 2.Redes de computadores I.Emmerich, Luiz Eduardo Freire II.Kamakura, Felipe de Aguiar III.Petriche, Rodrigo Carvalhedo IV.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais V.t.

RESUMO

Este trabalho apresenta o projeto, implementação, integração e testes do módulo de redes para um *engine* didático de criação de jogos eletrônicos desenvolvido pelo Laboratório de Tecnologias Interativas da Escola Politécnica da Universidade de São Paulo (Interlab) e o desenvolvimento de um protótipo de um jogo multiusuário utilizando este *engine*, aplicando conceitos do Processo Unificado (UP) como metodologia de Engenharia de Software adaptada para este projeto em particular.

O projeto foi desenvolvido com a utilização da tecnologia NIO, introduzida na plataforma de desenvolvimento Java J2SE a partir de sua versão 1.4, devido ao suporte a entrada/saída assíncrona e ao ganho de desempenho declarado pelos desenvolvedores em relação ao seu antecessor Java IO.

Dentre os requisitos desejados, a criação de uma arquitetura com foco na simplicidade de uso foi o aspecto mais valorizado, já que se trata de uma ferramenta cujo principal objetivo é o uso no ensino de computação, situação em que os usuários têm pouco tempo para aprender a usar a ferramenta e não são especialistas no assunto.

Além disso, este projeto apresenta um caráter multidisciplinar, envolvendo conhecimentos na área de redes de computadores, engenharia de software, computação gráfica e programação além da interação com o mercado de tecnologia na área de entretenimento. Juntamente a esse aspecto multidisciplinar existente no projeto outros dois grandes motivadores devem ser ressaltados: o crescimento do mercado de desenvolvimento e consumo de jogos e também a popularização dos jogos em rede, especialmente os "jogos maciçamente multiusuário" (MMG ou

massively multiplayer game) onde milhares de jogadores interagem, em tempo real, em mundos virtuais de estado persistente.

Como protótipo de jogo multiplayer foi implementado o "Lego Adventures Online", que teve como objetivos demonstrar as funcionalidades do módulo de redes e também servir como exemplo para o desenvolvimento de trabalhos futuros utilizando as características de rede do enJine.

ABSTRACT

This work presents the project, implementation, integration and tests for the creation of the network module of a didactic game engine developed by the Interactive Technologies Laboratory (INTERLAB) at University of Sao Paulo, and the development of a multiplayer game prototype using that engine. Unified Process was chosen as the software engineering methodology and was adapted to suit this project needs.

Java NIO, introduced in the Java J2SE release 1.4 development platform, was used in the development of the project due to its asynchronous IN/OUT support and the performance advantages declared by the developers against its antecessor Java IO.

Among the desired requirements, the creation of an architecture focused on the simplicity of use was the most important aspect, once that engine's main objective is to be used in the teaching of some computation concepts, a situation .where users have little time to learn how to use the tool and are not specialists in the subject.

Besides that this project presents a multidisciplinary character, dealing with computer networking, software engineering, computer graphics, programming and the interaction with the entertaining technology market. There are two other motivators that are the growing of development and consumption of games market and the popularization of network based games, especially the *massively multiplayer games* where thousands of players interact, in real time, in virtual persistent state worlds.

A demonstration game called "Lego Adventures" was developed as well, to show and prove the network module features and to be used as an example for further development using the network characteristics of enJine.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

- INTERLAB
  - Laboratório de Tecnologias interativas da Universidade de São Paulo
- MMG
  - Massively Multiplayer Game
- UP
  - Unified Process = Processo Unificado
- CASE
  - Computer Aided Software Engineering
- XP
  - Extreme Programming
- ABRAGAMES
  - Associação Brasileira de Desenvolvedores de Jogos
- MMORPG
  - Massively Multiplayer Online Role Playing Games = Jogos de interpretação de personagem maciçamente multiusuário
- P2P
  - Peer to peer
- UML
  - Unified Modeling Language
- API
  - Application Programming Interface
- NIO
  - New I/O
- CVS

    - Concurrent Versions System
- ERP
    - Enterprise Resource Planning
- OMG
    - Object Management Group
- ESA
    - Entertainment Software Association

Sumário

1 INTRODUÇÃO

1.1 Objetivo

O objetivo deste trabalho é projetar e implementar um módulo de redes para um *engine* didático de criação de jogos eletrônicos desenvolvido pelo Laboratório de Tecnologias Interativas da Escola Politécnica da Universidade de São Paulo (Interlab) e um protótipo de um jogo multiusuário utilizando este *engine*, aplicando conceitos do Processo Unificado (UP) como metodologia de Engenharia de Software adaptada para este projeto em particular. Tal adaptação visa selecionar, das soluções propostas pelo UP, aquelas mais adequadas a esse projeto em detrimento de outras que não adicionariam valor concreto a ele.

Em outras palavras busca-se desenvolver uma ferramenta e utilizá-la para a criação de um produto como prova de conceito.

O trabalho não tem como escopo desenvolver nem incrementar outras funcionalidades deste *engine* que não estejam relacionadas ao módulo de redes. Isto vale para a integração com banco de dados e de camadas que implementem segurança e encriptação de mensagens, por exemplo. No entanto ao longo do projeto é descrito como poderia haver a integração destes módulos com o *engine* ou como poderiam ser utilizados no desenvolvimento de jogos multiusuário, visto que essas são características importantes desse tipo de jogos, principalmente para os MMGs (*Massively Mutliplayer Games*).

Quanto ao protótipo, o escopo se restringe a uma versão de um jogo demonstrativo que permita interação entre jogadores através da rede.

Escolheu-se este tema para o projeto de formatura pelo seu caráter multidisciplinar, envolvendo conhecimentos na área de redes de computadores,

engenharia de software, computação gráfica, programação, além da interação com um tema relacionado com o mercado de tecnologia na área de entretenimento que é uma área com alto potencial de crescimento no Brasil.

O projeto do módulo de redes é apresentado ao longo deste documento e a descrição do jogo demonstrativo, suas regras e objetivos são detalhados no documento de *Game Design* (apêndice C).

1.2 Motivação

Como citado no item 1.1 além do caráter multidisciplinar e todo o caráter de aprendizado envolvidos nesse trabalho, a equipe foi motivada a desenvolver este projeto também pela importância que o mercado de jogos eletrônicos tem hoje no Brasil e no mundo. Busca-se nesta seção apresentar o cenário deste mercado.

1.2.1 O mercado de jogos eletrônicos

Há alguns anos comentou-se muito que o mercado mundial de jogos vinha crescendo tanto nos últimos tempos que já havia ultrapassado até mesmo a indústria cinematográfica [VEJA, 2003]. Podemos perceber esse crescimento analisando também as empresas que se encontram envolvidas nesse mercado. Abaixo, temos dados sobre algumas dessas empresas obtidos em Março/Abril de 2005.

Esses dados nos mostram o grande volume de capital que as principais empresas de jogos vêm movimentando nos últimos anos.

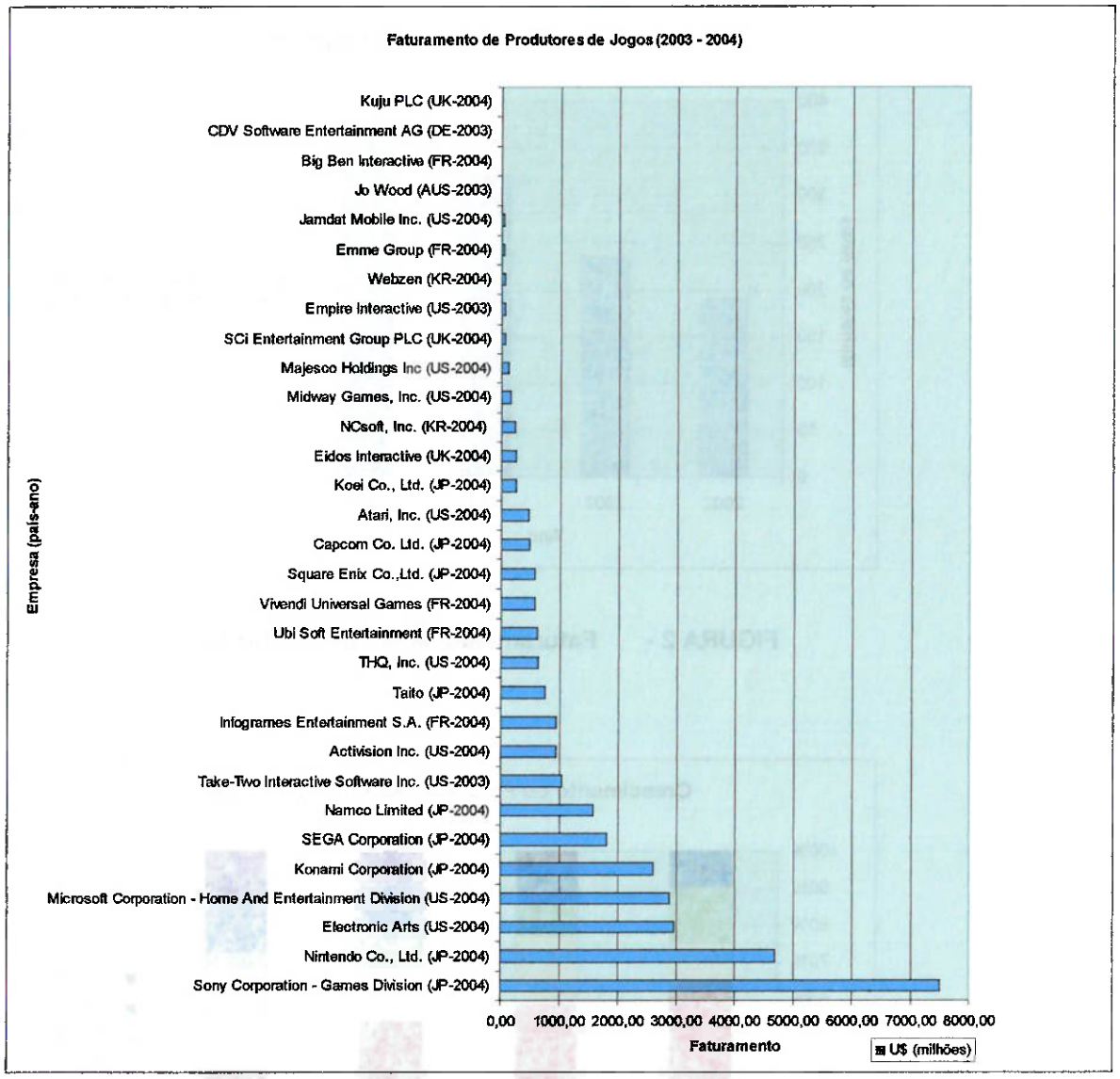

FIGURA 1 - Faturamento de produtores de jogos 2003/2004 [GAMELOCALIZATION, 2005]
(Retirado e adaptado de http://www.gamelocalization.net/facts_revenue.html)

Em documento intitulado "Pesquisa – A indústria de Desenvolvimento de Jogos Eletrônicos no Brasil", publicado pela Associação Brasileira de desenvolvedores de jogos (Abragames) em primeiro de maio de 2005, são apresentados dados sobre o mercado brasileiro que demonstram claramente o crescimento do setor: As figuras 2, 3 e 4 apresentam alguns dados desta pesquisa:

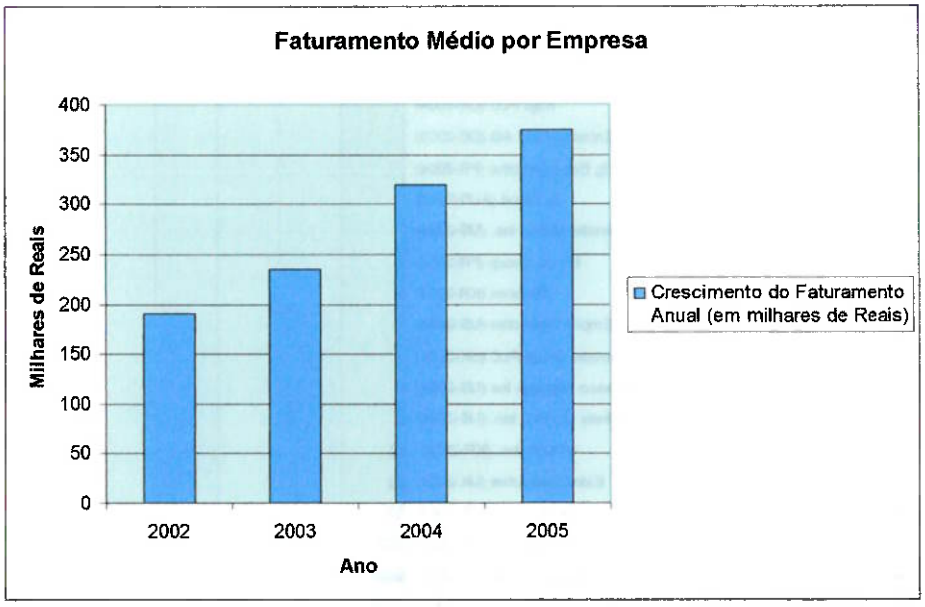

FIGURA 2 - Faturamento Médio por Empresa

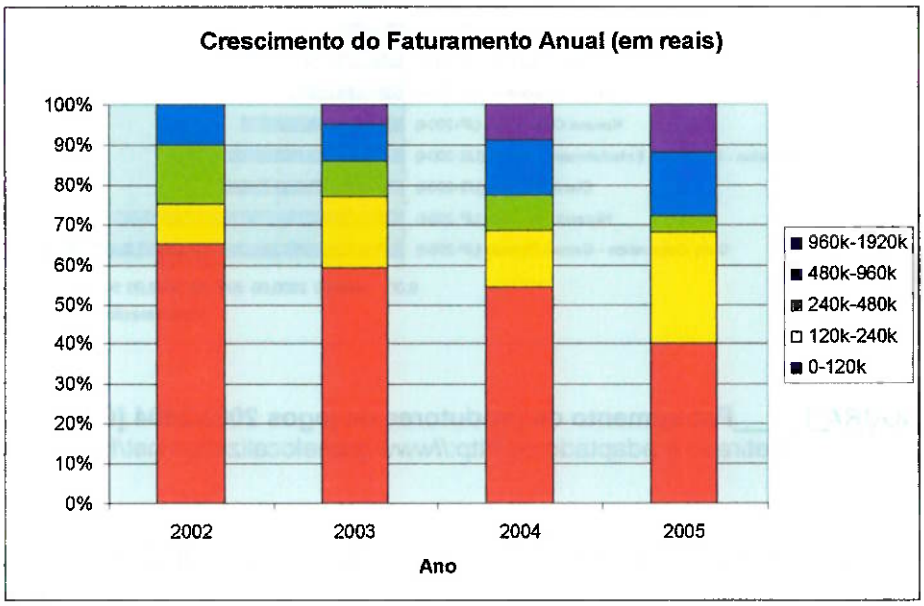

FIGURA 3 - Crescimento do Faturamento Anual por faixa

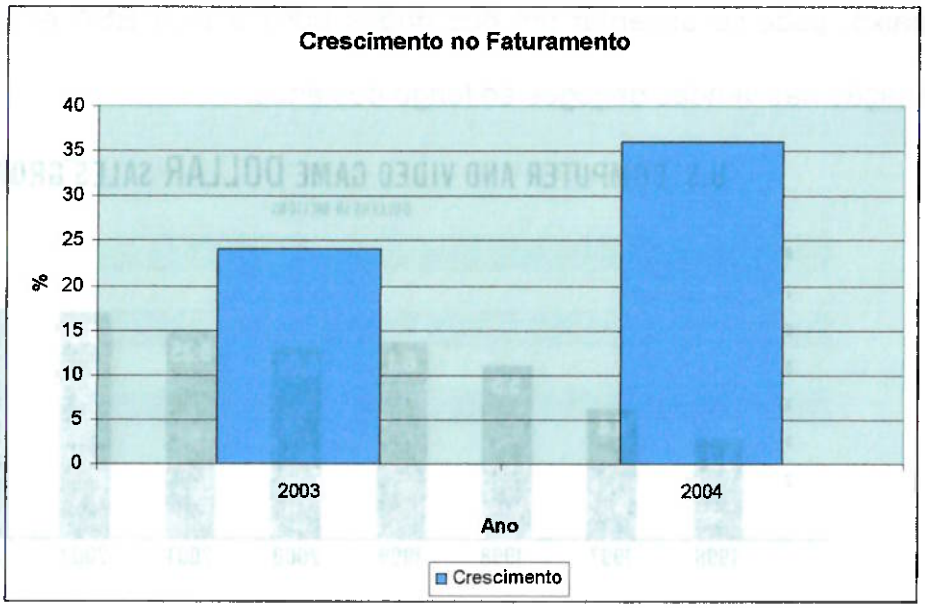

FIGURA 4 - Crescimento do Faturamento Anual

Da análise das figuras, notamos um crescimento do faturamento de cerca de 40% ao ano, o que demonstra o potencial do mercado brasileiro e a tendência de seu crescimento.

Um reflexo dessa tendência no Brasil são as iniciativas criadas pelo Ministério da Cultura para o fomento de projetos de desenvolvimento de jogos. O projeto JogosBr, criado pelo Ministério da Cultura em 2004, é um portal aberto a toda comunidade que incentiva a criação, discussão e colaboração de projetos de jogos eletrônicos. Anualmente, esse portal realiza um concurso nacional de desenvolvimento de jogos estimulando a produção nacional de jogos através de apoio financeiro que varia de R$30.000,00 a R$80.000.00. O portal pode ser acessado em http://www.jogosbr.org.br. [JOGOSBR]

A tendência de crescimento nacional acompanha o cenário observado mundialmente, como comprovam as pesquisas realizadas por associações como a ESA (Entertainment Software Association), que realiza pesquisas anuais a respeito do mercado do jogos e do perfil dos seus consumidores nos Estados Unidos.

Abaixo, pode-se observar um dos dados obtidos pela ESA em 2005 referentes à variação nas vendas de jogos ao longo dos anos:

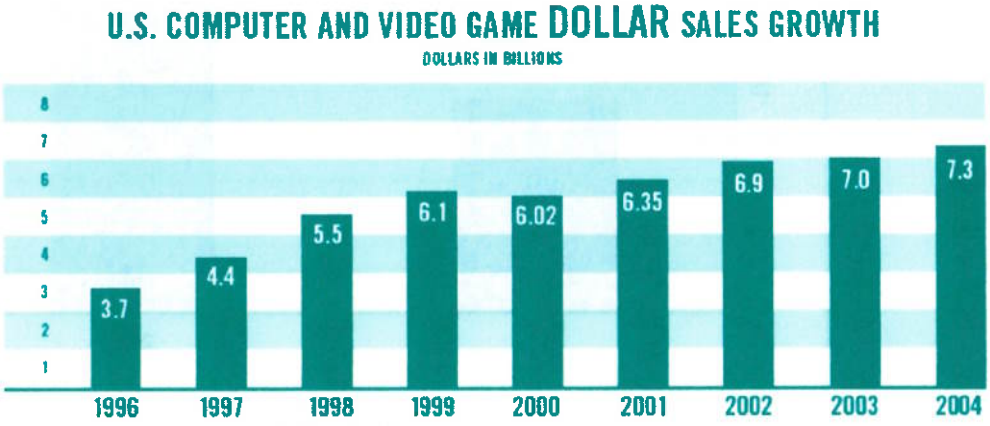

FIGURA 5 - Vendas de jogos de 1996 a 2004 nos EUA [ESA, 2005]

Com relação ao mercado mundial de jogos MMG, de acordo com uma análise realizada por uma consultoria especializada na área de jogos de mundos persistentes, a renda com jogos nesse estilo poderiam chegar a faixa de um bilhão de dólares em 2004 [IGDA, 2004]. Os jogos de mundos persistentes são, em geral, jogados em rede e suas principais características são o fato do mundo estar sempre disponível e dos acontecimentos ocorrerem continuamente, mesmo que os jogadores não estejam conectados.

Hoje, baseado em dados divulgados no início de 2005 pela Blizzard Entertainment, estima-se que o faturamento de somente um título de MMG seja de aproximadamente US$297 milhões [HARDCODEDGAMES, 2005].

1.2.2 Redes de computadores aplicadas a jogos eletrônicos

O desenvolvimento de tecnologias que proporcionam acesso à Internet de forma cada vez mais rápida e barata contribui também para o crescimento do mercado de jogos multiusuário. Como exemplo dessa tendência basta analisar a ultima geração de consoles lançados. Todos proporcionam meios de se jogar em

rede e todos têm enfatizado esse fato. Não só isso, mas também a febre de jogos maciçamente multiusuário, como Ragnarok Online [GRAVITY, 2002], World of Warcraft [BLIZZARD, 2004], entre outros, que permitem que milhares de jogadores interajam entre si e vem cada vez mais atraindo jogadores pelo mundo inteiro como se pode ver no gráfico abaixo:

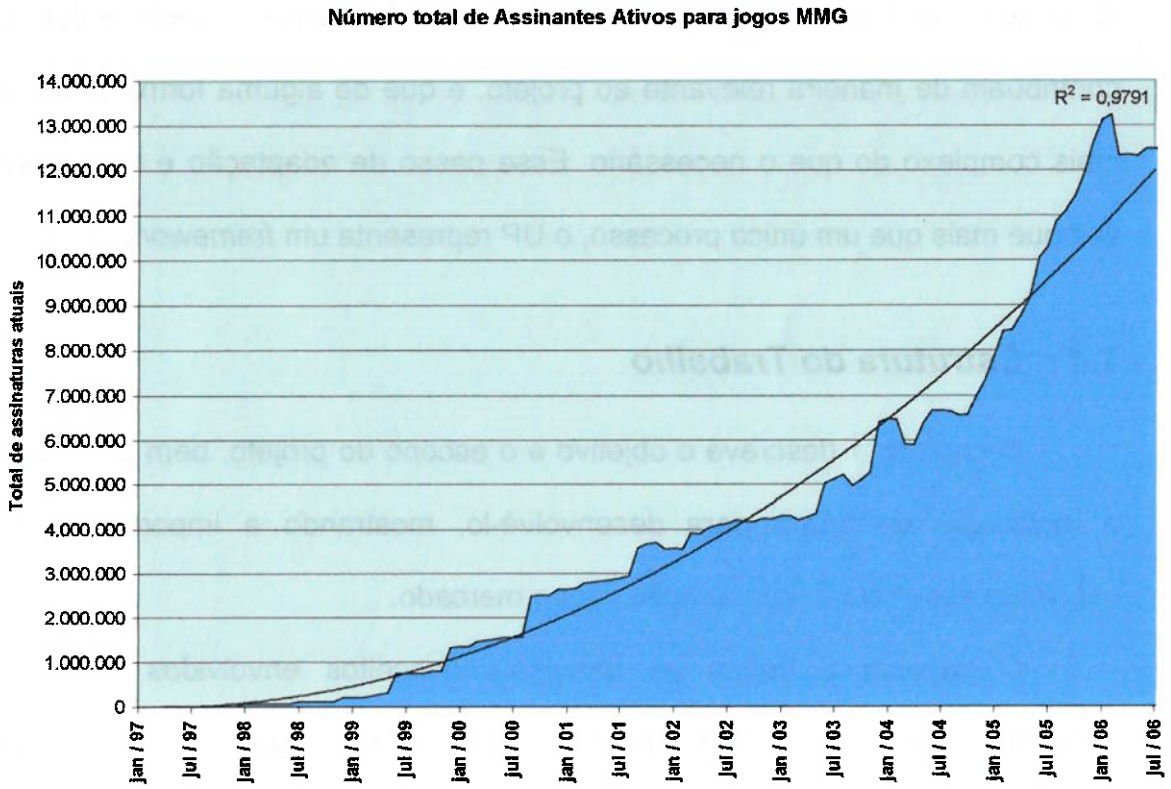

FIGURA 6 - Crescimento do número de assinantes ativos de jogos maciçamente multiusuário.
(Dados retirados de http://mmogchart.com/ [MMOGCHART])

Dessa tendência, o grupo entende que conhecer as tecnologias relacionadas ao desenvolvimento de jogos com suporte a múltiplos jogadores é um diferencial importante no mercado, não só para o mundo do desenvolvimento de jogos, mas para o desenvolvimento de outras aplicações *online*, de natureza similar.

1.3 Metodologia

A metodologia adotada para o projeto foi o Processo Unificado, cuja teoria é descrita brevemente no item 2.1 em Fundamentos Teóricos e de maneira mais detalhada no Apêndice A, além de existir uma descrição da implementação presente ao longo do capítulo 4. Vale ressaltar que o UP não é adaptado às particularidades do projeto, com o intuito de se evitar que sejam adotados procedimentos que não contribuam de maneira relevante ao projeto, e que de alguma forma possa torná-lo mais complexo do que o necessário. Esse passo de adaptação é necessário, uma vez que mais que um único processo, o UP representa um *framework*.

1.4 Estrutura do Trabalho

O capítulo 1 descreve o objetivo e o escopo do projeto, bem como apresenta a motivação do grupo para desenvolvê-lo, mostrando a importância do tema escolhido assim como sua relação com o mercado.

O capitulo 2 define os principais conceitos envolvidos nesse projeto, detalhando a metodologia utilizada, definindo o que é um *engine* de desenvolvimento de jogos bem como o conceito de jogos maciçamente multiusuário.

No capítulo 3, são apresentados argumentos para as escolhas de projeto que serão utilizadas na etapa de implementação.

O capítulo 4 trata da etapa de implementação, como foi realizado o desenvolvimento e a integração do módulo de redes com o *engine* e as etapas de criação do protótipo do jogo.

O capítulo 5 apresenta os resultados principais de nossa implementação, e o capítulo 6 busca utilizar estes resultados para apresentar conclusões sobre o projeto e possíveis trabalhos futuros.

2 FUNDAMENTOS TEÓRICOS

Nesta seção são discutidos os fundamentos teóricos utilizados para o desenvolvimento do projeto através de itens que abordam a seleção da metodologia adotada para o projeto, a definição do conceito de *engine*, as características de um jogo maciçamente multiusuário e outros conceitos comuns em jogos *online*.

2.1 Metodologia do projeto

Apesar de algumas metodologias de desenvolvimento de softwares serem acessíveis há algum tempo, ainda não são largamente utilizados nas empresas, que continuam não utilizando uma abordagem estruturada em seus processos. Na indústria de jogos ocorre o mesmo. No entanto, de acordo com [LLOPIS, N. 2004], os desenvolvedores de jogos têm demonstrado maior interesse na aplicação de práticas de engenharia de softwares recentemente.

Essa tendência observada na indústria foi uma das motivações que levou a equipe a optar por utilizar uma metodologia de desenvolvimento no projeto de formatura. Outro fator que influenciou a equipe nessa escolha foi a necessidade de garantir o término do projeto no prazo estipulado.

A equipe decidiu avaliar o Processo Unificado (UP) e o conjunto de práticas de desenvolvimento, intitulado como Extreme Programming (XP), dentre as quais foi escolhida a metodologia UP. Nos itens seguintes é realizada uma breve apresentação das duas metodologias, a descrição do processo de escolha e os resultados dessa avaliação.

2.1.1 UP

O UP é um processo de desenvolvimento de software que usa uma abordagem orientada a objetos em sua concepção e que, ao contrário do XP, prevê uma etapa mais longa de planejamento. Essa etapa engloba duas (iniciação e elaboração) das quatro fases que compõe um projeto de software segundo o UP: iniciação, elaboração, construção e transição, existindo ainda a possibilidade de serem realizadas iterações em cada uma dessas etapas ao longo do projeto. Outra diferença é o fato do UP ser bastante orientado a documentos, enquanto no XP as práticas são mais relevantes.

A fase de iniciação é uma etapa focada nos aspectos de negócio e tem como principais objetivos o levantamento de requisitos e a análise da viabilidade técnica e financeira de um projeto que atenda aos requisitos levantados. O levantamento de requisitos envolve a criação de modelos que ajudem a entender o negócio e as funcionalidades do sistema a ser projetado. O UP recomenda a utilização de diagramas de casos de uso da UML complementados de documentos textuais para a representação desses modelos.

A utilização de diagramas da UML não é exclusiva dessa atividade. Diversas atividades do UP podem ser realizadas com o auxílio da UML e existem inúmeras ferramentas CASE próprias para a criação desses diagramas.

Já a fase seguinte, a de elaboração, tem foco nos aspectos técnicos do desenvolvimento. Uma atividade inicial e de fundamental importância dessa fase é o refinamento dos requisitos levantados na fase anterior para evitar que pequenas dúvidas relativas ao negócio prejudiquem a fase de construção que vem a seguir.

A atividade que caracteriza a fase de elaboração é a definição da arquitetura a ser adotada. A arquitetura é descrita através de modelos que representam os

inúmeros componentes do sistema e a maneira como esses componentes interagem para a realização de uma determinada tarefa. Além disso, faz parte dessa etapa o planejamento das inúmeras iterações que compõe a fase de construção para se saber quais são os resultados esperados em cada uma delas.

A fase seguinte é a fase de construção que, como se pode observar na tabela abaixo, ocupa a maior parte do esforço e tempo de um projeto de desenvolvimento de software. A fase de construção engloba o desenvolvimento dos componentes especificados na arquitetura e seus testes. Nessa fase, são pontos cruciais o gerenciamento dos recursos e o controle e a otimização dos processos.

	Iniciação	**Elaboração**	**Construção**	**Transição**
Esforço:	~5%	20%	65%	10%
Tempo Gasto:	10%	30%	50%	10%

TABELA 1 - Distribuição de esforço e tempo durante um projeto que utiliza UP
(Dados retirados de [WEST, 2006])

A última fase, a de transição, envolve duas preocupações: testes e logística. Testa-se o produto em ambiente de produção, realizam-se refinamentos de acordo com a avaliação dos usuários, os documentos de suporte ao usuário são concluídos e a distribuição do produto aos usuários é planejada e executada.

As fases do desenvolvimento sugeridas pelo UP são executadas a cada lançamento de uma nova versão de um sistema. No entanto, as atividades executadas em cada fase são extremamente particulares a cada projeto. Elas podem variar de acordo com alguns parâmetros como o tipo de software que está em construção e o tipo de versão (inicial ou atualização).

2.1.2 XP

A XP se trata de uma disciplina de desenvolvimento ágil de software. A XP afirma que é possível conduzir um projeto de software de maneira progressiva, partindo de um projeto simples e incrementar esse projeto ao longo de seu desenvolvimento, adicionando complexidades apenas quando necessário e fazendo ajustes pequenos, porém de maneira constante.

Entre os desenvolvedores de software sempre se constatou que a curva de *Custo de Modificação X Tempo* de um projeto de software é exponencial, ou seja, quanto mais tarde for feita uma modificação no sistema seu custo será exponencialmente mais alto. Dessa forma sempre se desejou ter todas as mudanças feitas durante as fases iniciais do projeto (especificação, análise, design).

Para que esse desenvolvimento incremental e de constantes alterações proposto pela XP funcione, a disciplina propõe que, com o uso das diversas tecnologias e práticas voltadas para o desenvolvimento de software existentes atualmente, a curva de *Custo de Modificação X Tempo* do Projeto possa ser "achatada".

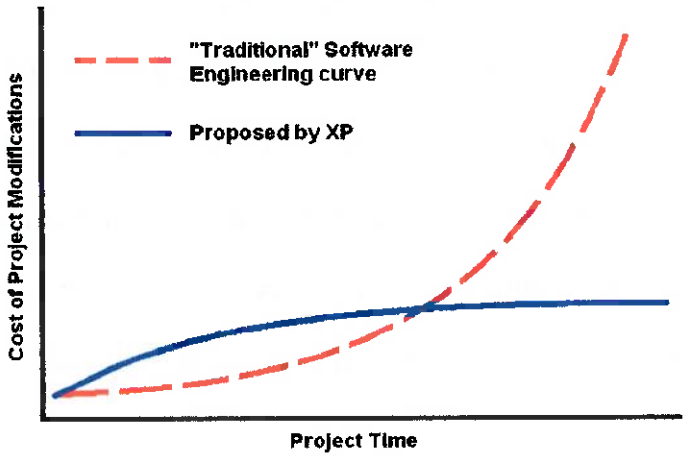

FIGURA 7 - Comparação entre as curvas de custo de modificação [NAKAMURA et al, 2005]

Dessa forma, para se aplicar a XP é necessário que algumas práticas sejam seguidas, dentre elas:

- **Histórias do usuário** – Análogo aos Casos de Uso do UP, são descrições de funcionalidades desejadas pelo cliente. Elas são base para que a equipe de desenvolvimento possa estimar os recursos e tempo necessários para a conclusão das tarefas do projetos. Os clientes podem definir quais histórias serão implementadas primeiro, assim como o intervalo entre o lançamento de versões funcionais.
- **Programação em pares** – a codificação é realizada com dois programadores em cada computador. Assim, o código é constantemente revisado durante sua concepção, e a comunicação entre desenvolvedores é incentivada.
- **Testes de unidade** – são testes automatizados e realizados de maneira contínua durante o desenvolvimento. São criados pelos desenvolvedores para que eles possam testar o código que escrevem. Esses testes se focam em porções pequenas do sistema como uma classe ou um pequeno conjunto de classes.
- **Integração continua** – qualquer alteração deve ser integrada ao código base diariamente. Os testes devem funcionar perfeitamente depois das integrações.
- **Refatoração** – deve ser realizada continuamente para a eliminação de código duplicado e desnecessário, e para que a arquitetura do sistema evolua ao longo do projeto.

As práticas citadas acima são consideradas as mais importantes da metodologia, porém existem outras. Uma delas afirma que é importante que uma pessoa jamais seja considerada detentora do desenvolvimento de um módulo. Qualquer desenvolvedor deve ser capaz de atuar em qualquer parte do código a todo momento. Outra prática condena a excessiva utilização de horas extras, afirmando que se isso se repete em semanas consecutivas, o andamento do projeto está incorreto.

Por fim, existe a necessidade de que a equipe de desenvolvimento tenha acesso constante aos clientes que utilizarão o sistema e de que seja realizada uma padronização de código.

Essas práticas colaboram uma com as outras, sendo que o ponto forte de uma compensa o ponto fraco de outra.

Durante o desenvolvimento de um projeto de software, as mudanças são inevitáveis, por isso a XP propõe que é mais vantajoso controlar o desenvolvimento fazendo diversos pequenos ajustes ao invés de poucos grandes ajustes.

De acordo com seus criadores, a XP se baseia em quatro valores fundamentais:

- Comunicação
- Simplicidade
- Feedback
- Coragem

Como explicado em [BECK, 2004]:

A falha na Comunicação é um dos principais causadores de problemas em projetos de software, por isso a XP propõe práticas que não podem ser realizadas sem comunicação.

A Simplicidade diz que devemos optar pelas alternativas mais simples possíveis, evitando antecipar funcionalidades que no momento não são necessárias.

O Feedback é o conhecimento do estado atual do sistema, através de informações vindas do resultado de testes de unidade, testes de funcionalidades feitas pelos clientes e dos próprios usuários. Feedback também pode acontecer quando o programador avalia as histórias escritas pelo cliente.

Coragem, no âmbito da XP, significa não hesitar em eliminar código ou implementar mudanças caso a solução atual não esteja suficiententemente boa, ou se uma nova idéia de implementação surgir.

Esses quatro valores, que se complementam mutuamente, servirão de guia para o desenvolvimento das diversas práticas dessa metodologia.

O XP é considerado mais adequado para projetos até médio porte de *software* feito "sob encomenda", com um cliente bem definido. Apesar disso ele pode ser adaptado para ser usado no desenvolvimento de "produtos de prateleira" ou de *middlewares*, como no nesse trabalho. [BECK, 2004]

2.1.3 Critérios de escolha

Nesta seção, os critérios de escolha utilizados para a seleção da metodologia de desenvolvimento do projeto serão enumerados juntamente com seus pesos. As notas atribuídas a cada critério variam de 0 a 5, sendo que números maiores representam uma solução mais adequada. Já o peso pode assumir valor 1 ou 2 de acordo com sua relevância na escolha. Notas e pesos foram atribuídos

subjetivamente com base na análise da literatura sobre as metodologias. Os critérios selecionados são os seguintes:

- **Tempo de aprendizado** – como o projeto tem curto período de desenvolvimento é importante que o aprendizado da metodologia a ser utilizada não prejudique o andamento geral do projeto. Esse quesito é considerado importante e terá peso 2 na avaliação.
- **Aplicação no desenvolvimento de jogos** – apesar de ser um projeto de software, o desenvolvimento de jogos possui certas particularidades que impedem o uso de alguns processos em seu formato original. Esse item avalia o quanto uma determinada metodologia é adequada e pode ser adaptada ao desenvolvimento de jogos. Esse item não é prioritário para a escolha já que qualquer opção escolhida poderá ser adaptada de alguma forma ao projeto e, por isso, receberá peso 1.
- **Ferramentas de apoio** – a aplicação de cada metodologia pode ser auxiliada por softwares especializados. Como as diversas opções seguem filosofias diferentes, uma comparação de funcionalidades não faria sentido. Porém, podemos comparar a quantidade de opções disponíveis e se essas opções são pagas ou não, quesito que definirá a viabilidade de se usar uma determinada ferramenta. Se uma determinada metodologia for excelente para esse projeto, ela será aplicada independentemente das ferramentas de apoio já que os conceitos podem ser aplicados sem o auxílio delas. Por isso, esse item terá peso 1.
- **Adaptabilidade ao projeto** – sem dúvida esse é um item decisivo para a escolha da metodologia de desenvolvimento e receberá peso 2. Artigos como

o [DEMACHY, 2003] abordam a utilização das metodologias utilizadas no mercado de softwares na indústria de jogos eletrônicos, porém nem sempre as conclusões que esses artigos obtêm podem ser aplicadas ao projeto em questão uma vez que se trata de um projeto acadêmico com algumas particularidades como, por exemplo, a pequena preocupação com a parte artística que é muito considerada em projetos comerciais de jogos.

2.1.4 Resultados da avaliação

Critério	Peso	UP	XP
Tempo de aprendizado:	2	4	2
Aplicação no desenvolvimento de jogos:	1	4	3
Ferramentas de apoio:	1	3	3
Adaptabilidade ao projeto:	2	4	2
Total:		24	14

TABELA 2 - Tabela de escolha de metodologia

A tabela acima mostra as notas obtidas por cada metodologia na avaliação realizada pela equipe. Pode-se observar que a metodologia UP obteve grande vantagem. Abaixo a avaliação será comentada por critério.

Com relação ao tempo de aprendizado, a vantagem do UP se deve à familiaridade que os integrantes da equipe de desenvolvimento possuem com o processo já que é a metodologia ensinada e praticada nas inúmeras disciplinas do curso. Apesar da familiaridade, como se trata de um processo complexo, algumas

horas serão gastas para recordar e aprofundar o estudo dos conceitos relacionados, o que explica a não obtenção de nota máxima.

A nota 2 obtida pelo XP se deve ao fato de dois integrantes da equipe já terem realizado uma pesquisa e experimentos sobre o assunto em um trabalho anterior [NAKAMURA et al., 2005]. A nota não foi maior porque os outros dois integrantes jamais tiveram qualquer contato com o XP e a metodologia não é ensinada ou utilizada nas disciplinas da universidade e, portanto, existe uma carência de conhecimentos teóricos e experiências práticas. Como conseqüência, a equipe despenderia de muito tempo para aprender uma nova metodologia que possui uma filosofia bem diferente das metodologias mais tradicionais.

Com relação à aplicação dessas metodologias no desenvolvimento de jogos, pôde-se constatar durante a fase de pesquisa de artigos científicos que existem relatos de sucesso [FLOOD, 2003] na utilização de ambas metodologias no desenvolvimento de jogos, desde que sejam aplicados com certas adaptações. A vantagem de se usar o UP deve-se ao fato de essa metodologia já ter sido utilizada com sucesso pelos integrantes do Interlab em alguns de seus projetos de maior porte.

Com relação às ferramentas de apoio às metodologias, foi realizada uma pesquisa para se saber qual delas é melhor suportada por ferramentas. Concluiu-se que existem pacotes de apoio à ambas, no entanto, tais softwares são proprietários e pagos e, portanto, não foram considerados nesta avaliação. Como alternativa, optou-se por avaliar softwares de modelagem e verificou-se que existe uma quantidade grande de ferramentas de modelagem UML, até mesmo de ferramentas gratuitas ou livres, como se pode observar em http://en.wikipedia.org/wiki/List_of_UML_tools. [WIKIPEDIA_UML] Como o UP utiliza UML e o XP proporciona

essa possibilidade, decidiu-se que ambas as metodologias receberiam nota 3, não sendo maior devido a escassez de softwares específicos que apoiam todo o ciclo de desenvolvimento.

As notas atribuídas ao último critério têm como base o segundo critério (Aplicação no desenvolvimento de jogos). O XP perdeu ponto devido às seguintes constatações:

- Apesar de ambas as metodologias considerarem a existência de um cliente, a sua participação dentro de um projeto dirigido de acordo com o XP é muito maior, já que é recomendada a possibilidade de acesso constante ao cliente, diferentemente do UP que procura concentrar o contato com o cliente no final de cada iteração. Além disso, mesmo que a equipe tentasse simular a existência desse cliente constantemente acessível, os resultados não seriam satisfatórios já que os desenvolvedores geralmente pensam de maneira diferente das pessoas para as quais prestam serviço.
- Seria inviável aplicar uma das práticas principais do XP, a programação em pares uma vez que os integrantes estarão em período de estágio durante grande parte do desenvolvimento.

Após a realização dessa escolha, exigiu-se da equipe um estudo mais aprofundado a respeito do UP. O resultado alcançado através desses estudos foi a criação do Apêndice A – Unified Process, que descreve o processo de maneira mais detalhada.

2.2 *Engine para criação de jogos eletrônicos*

Game engines são bibliotecas de *software* utilizadas para a criação de jogos. Eles facilitam o trabalho do desenvolvedor, pois escondem detalhes de implementação, fornecendo uma interface de programação que permite o desenvolvedor se concentrar nos aspectos relacionados principalmente ao jogo em si. Um determinado *engine* pode facilitar a implementação dos seguintes elementos de um jogo, por exemplo:

- renderização: tratamento de imagem, parte gráfica;
- som;
- tratamento de entrada e saída: interação com o usuário;
- rede;
- detecção de colisão: interação entre objetos;
- inteligência artificial: comportamento de objetos;
- portabilidade;

Apesar de ser um componente de *software* específico para a criação de jogos, é importante que ela seja genérica no sentido de possibilitar o desenvolvimento de jogos de segmentos diferentes, ou de diversos jogos diferentes de um mesmo segmento.

2.2.1 EnJine

O enJine é um *engine open-source* voltado para a criação de jogos 3D utilizando as tecnologias Java e Java 3D. É um dos projetos mais antigos do grupo de pesquisa de jogos do Interlab e atualmente se encontra na versão 3.0.

Seu principal objetivo atualmente é ser utilizado como ferramenta didática para o ensino de projeto de jogos eletrônicos, computação gráfica, engenharia de software e demais conceitos de ciência da computação.

[NAKAMURA et al. 2006] explica que, devido a esse objetivo, o foco do projeto é prover um *game engine* de código aberto, bem estruturado, simples o suficiente para permitir o desenvolvimento de jogos relativamente complexos em um curto período de tempo, incluindo o tempo de aprendizado.

Sendo assim, um dos requisitos mais importantes do enJine e, conseqüentemente, de qualquer outro módulo que venha a ser adicionado a ele é a simplicidade. Isso decorre do fato de que os alunos de computação, principais usuários do enJine, em geral possuem pouco tempo disponível para o seu aprendizado.

Um objetivo secundário do enJine é seu uso como plataforma de testes para a aplicação de novas tecnologias em jogos, como por exemplo o desenvolvimento de jogos em Realidade Aumentada. A simplicidade também é importante para essa aplicação, para reduzir a incerteza relativa ao enJine ao testar uma tecnologia nova para o grupo.

O enJine possui um conjunto de classes Java que permite a representação de elementos de alto nível de um jogo como por exemplo personagem, jogador e cenário. Ele fornece também a renderização de gráficos 3D através da API Java3D, suporte a som, colisão e interação com o usuário.

Suporte a Inteligência Artificial, Física e *Scripting* são funcionalidades planejadas para futuras versões do enJine [NAKAMURA et al. 2006].

2.2.1.1 Arquitetura do enJine

Como ilustrado na Figura 7 o enJine pode ser divido em três camadas principais. Em sua camada mais baixa encontram-se as APIs do Java e do Java3D, alicerces de toda sua implementação.

Na camada logo acima, estão todos os módulos funcionais do enJine. Esses módulos fornecem às camadas superiores as funcionalidades básicas necessárias para se construir um jogo, como por exemplo, renderização gráfica, saída de som e colisão. É possível construir jogos utilizando somente essa camada, mas como tratam-se de funcionalidades de natureza mais genérica, essa tarefa é mais complexa e deve ser realizada somente quando a próxima camada não atender os requisitos do jogo.

Por fim a camada superior oferece classes com uma implementação padrão dos serviços fornecidos pelo enJine para estilos de jogos diferentes, com o intuito de fornecer um nível maior de abstração para os usuário de modo a acelerar o tempo de desenvolvimento de um jogo [NAKAMURA et al. 2006].

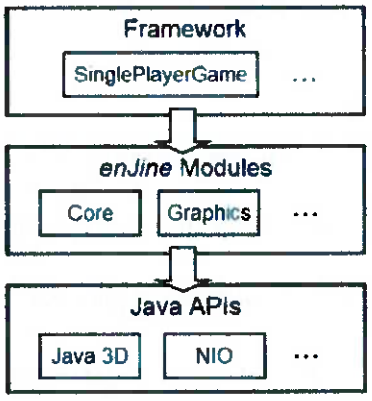

FIGURA 8 - As três camadas do enJine

A Figura 8 mostra a organização dos pacotes que compõem o enJine.

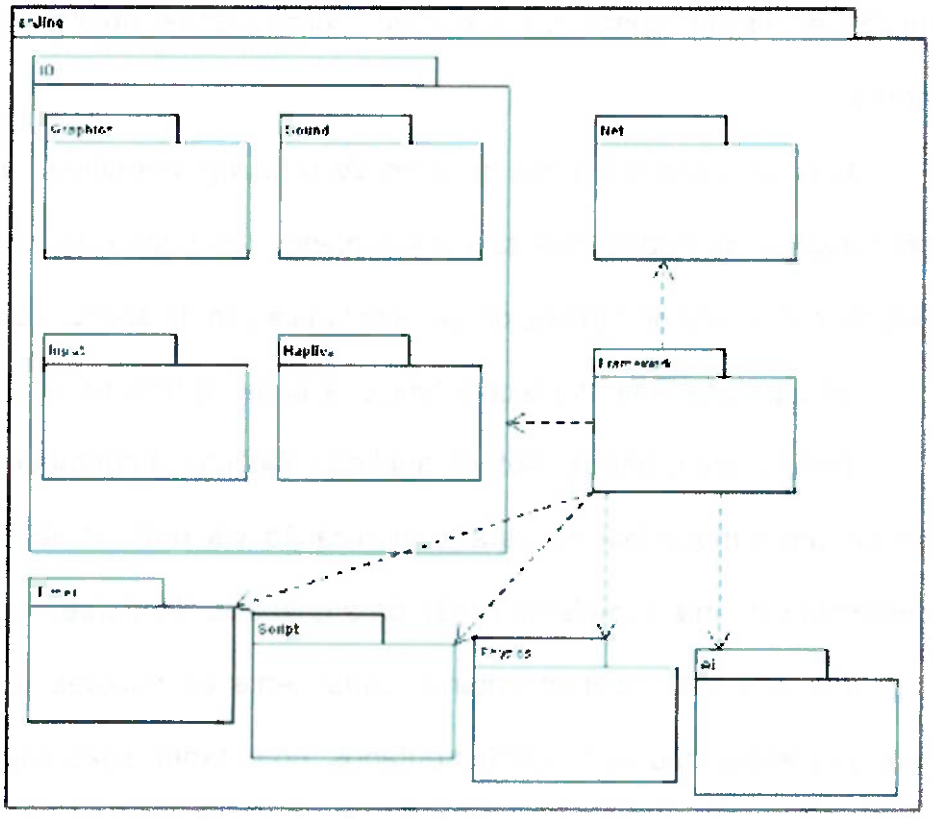

FIGURA 9 - Pacotes do enJine

O pacote *Framework* fornece ao programador uma implementação básica dos serviços fornecidos pela enJine voltada para certos tipos de jogos, facilitando o desenvolvimento. Esse pacote representa a camada superior do enJine citada anteriormente.

O pacote *IO* cuida de toda a parte de entrada e saída do jogo, subdividindo-se em outros quatro pacotes: *Graphics*, que cuida da parte de renderização, *Sound* que é responsável pelo controle dos efeitos sonoros e música, *Input* que trata os dispositivos de entrada como mouse, teclado, joystick, etc. , e o pacote *Haptics* que cuida da saída táctil.

O pacote *AI* fornece algumas funcionalidades de inteligência artificial enquanto o pacote *Physics* fornece funcionalidades básicas para o uso em

simulações físicas como, por exemplo, colisão entre objetos e interação com o terreno.

O pacote *Script* fornece serviços de *scripting*, permitindo que programadores criem trechos de código que são interpretados em tempo de execução de forma a interagir com a enJine através de um nível mais alto de abstração.

Já o pacote *Timer* fornece a funcionalidade de temporizadores.

Finalmente o pacote *Net*, no qual se concentra a contribuição desse trabalho, fornece um suporte básico para comunicação via rede. Atualmente esse módulo contempla somente a implementação da arquitetura cliente/servidor.

Como citado no item anterior, atualmente os pacotes *AI, Physics e Script, além do Haptics* não estão implementados, no entanto, suas implementações estão planejadas para as próximas versões do enJine.

2.3 Jogos eletrônicos maciçamente multiusuário

Jogos denominados *Massively Multiplayer Game* (MMG) em inglês e maciçamente multiusuário em português, são jogos que tem duas características principais, devem suportar um número muito grande de jogadores simultaneamente, onde esta quantidade pode variar entre algumas centenas de jogadores até centenas de milhares de jogadores [KENT, 2004; IGDA, 2004] e devem suportar simulação de estado persistente [CECIN, F. R. et al. 2005], isto é, o ambiente do jogo é constantemente alterado pelas ações dos jogadores e essas alterações são mantidas e vão estar presentes numa sessão futura de jogo. Devido a essa demanda de suporte a muitos jogadores e a necessidade de se guardar alterações do "mundo virtual" a cada nova ação tomada, esses jogos necessitam de uma infra-estrutura de rede que seja acima de tudo escalável.

As seguintes funcionalidades são desejáveis para um jogo MMG:

- Suporte a múltiplos servidores (balanceamento de carga): teoricamente, um jogo online pode utilizar um único servidor físico (uma única máquina) desde que esse forneça processamento e taxa de transmissão suficiente para comportar o número de jogadores desejado. Porém, na prática, pode ser difícil [IGDA, 2004] hospedar em uma única máquina todo o poder de processamento (CPU e memória) necessário para executar a simulação, em tempo real, de um "mundo virtual" habitado por milhares ou milhões de jogadores.
- Mecanismos seguros para a autenticação de jogadores: a segurança na identificação de cada usuário no sistema é fundamental para garantir que os dados de um jogador não sejam acessados por outro usuário.
- Sincronização de estados entre jogadores e o servidor: para jogos multiusuário, é necessário manter a sincronização do estado do jogo (servidor) e o estado dos jogadores para garantir que todos os usuários estejam sincronizados com relação à situação atual da partida.
- Divisão dos jogadores em grupos lógicos para otimizar a utilização dos recursos de rede: a economia de recursos de rede num ambiente que deve suportar milhares de jogadores é fator fundamental para este tipo de aplicação, e uma das técnicas utilizadas é dividir os jogadores em grupos lógicos (definidos de acordo com a implementação e necessidade do jogo), e assim as mensagens de sincronização de estado são passadas apenas para aqueles grupos que se interessam pela mensagem. Por exemplo, se dois grupos estão em regiões A e B distintas e uma ação na área A não afeta o estado da área B, faz sentido então que apenas os jogadores na área A

estejam interessados naquela ação e que os jogadores da área B não recebam nenhuma mensagem relativa a ela.

Além destas características relacionadas ao módulo de redes, neste tipo de jogo existe ainda a necessidade de interação com outros módulos como o acesso a banco de dados e outros aspectos relacionados à segurança (criptografia de dados, por exemplo).

2.4 *Pré-processamento*

Em jogos maciçamente multiusuário costuma-se utilizar o conceito de pré-processamento lógico e visual para gerar resultados de ações no próprio cliente antes de recebê-los do servidor.

Um pré-processamento comumente utilizado em jogos maciçamente multiusuário é a predição de movimento, também chamado de "*Dead Reckoning*". A predição de movimento efetuada pelo cliente tem duas funções principais, uma é de fornecer uma resposta imediata das ações tomadas (os jogadores querem ver o resultado de suas ações imediatamente em suas telas) e a segunda é de esconder o atraso causado pela variação de tempo de envio de mensagem ao servidor, processamento da informação e retorno da ação ao cliente.

"*Dead Reckoning*" é uma forma de replicação computacional no qual todos os clientes que estão participando no jogo acabam simulando todas as entidades envolvidas. Um exemplo de um caso para a utilização de "*Dead Reckoning*" seria no momento que um jogador faz uma requisição de movimento para seu personagem no jogo. Essa requisição vai ser transformada em uma mensagem, enviada ao servidor, processada e enviada de volta ao cliente sob forma de uma nova

mensagem de replicação informando que a requisição de movimento foi aceita ou não e que vai ser responsável por atualizar a posição deste personagem. Dependendo do atraso inserido pela rede e mesmo do tempo de processamento no servidor (que pode variar em função do número de jogadores em uma determinada região, por exemplo), a sensação de resposta para o jogador que fez a requisição de movimento seria prejudicada, pois a resposta visual do sistema viria apenas depois de todo o processamento ter sido realizado. O "*Dead Reckoning*" funciona tentando extrapolar o comportamento do personagem antes mesmo de o cliente que fez a requisição ter recebido a resposta do servidor, isto é, no caso citado anteriormente, no momento em que o jogador fizer a requisição de movimento, o cliente inicia a movimentação do personagem no mesmo instante em que a sua mensagem é enviada ao servidor. Assim, quando a mensagem de resposta do servidor chegar com o pedido de movimento efetivado, o movimento já terá sido realizado, escondendo do jogador esse tempo de atraso entre sua requisição e o processamento visual da ação. O *Dead Reckoning* funciona bem com atrasos relativamente pequenos ou para ações do jogador que apresentem pequena variação. Mudanças bruscas realizadas pelo jogador em um ambiente com atraso alto normalmente fazem com que esse algoritmo apresente resultados insatisfatórios. É um consenso, no entanto, que essa solução é preferível à alternativa de não fornecer nenhum feedback da ação do usuário enquanto se aguarda a resposta do servidor. [ARONSON, 1997]

A implementação de algoritmos de "*Dead Reckoning*" não estava no escopo do projeto, mesmo porque cada implementação está muito vinculada à própria implementação do jogo, mas a arquitetura do módulo de redes desenvolvido suporta a implementação de tais comportamentos, permitindo assim ao desenvolvedor do

jogo adicionar uma lógica de pré-processamento ao montar e dar inicio à troca de mensagens entre o cliente e o servidor. A aplicação desenvolvida pelo grupo e também apresentada neste documento utiliza o pré-processamento durante a movimentação do personagem.

2.5 *Tratamento assíncrono de mensagens*

O tratamento assíncrono de mensagens é um requisito muito comum para servidores que devem suportar diversos clientes simultâneos.

De maneira simples, a diferença entre interações assíncronas e síncronas reside no fato de que quando se trata de uma interação síncrona, deve-se realizar uma atividade de cada vez, em uma ordem fixa, enquanto na assíncrona é possível realizar mais de uma atividade simultaneamente, e a ordem de termino delas não é fixa.

Quando se trata de processos de entrada e saída num sistema por sincronismo, entende-se que apenas um processo vai ser processado por vez, isto é, quando uma requisição é feita a um componente síncrono, esta será tratada e o sistema ficará bloqueado até que o processo em questão termine.

Em um sistema assíncrono o tratamento de um processo não implica em bloqueio de outras rotinas do sistema, o que é muito importante para um jogo multiusuário, pois o tratamento de cada mensagem recebida não pode bloquear ou prejudicar o processamento de outras rotinas do sistema, como por exemplo a renderização de toda parte visual, devido ao atraso comum no tráfego das mensagens pela rede.

O conceito de assincronismo será aprofundado no item 4.2 quando se fala sobre o Java NIO, a tecnologia utilizada para a implementação do módulo de redes do enJine.

2.6 *Migração entre servidores*

Outra característica bastante comum em jogos maciçamente multiusuário é a existência de mais de um servidor para administrar o mundo de jogo. Diversas formas de implementação podem ser utilizadas para esse fim, como por exemplo a utilização de servidores por área de interesse ou até mesmo redundância de servidores. Isso tem como objetivo principal a melhora de performance do jogo e, consequentemente, o aumento na capacidade de usuários simultâneos do mesmo.

Para este projeto a forma de implementação não é tão relevante, pois depende do jogo ao qual ela está sendo aplicada, mas a possibilidade da existência de diferentes servidores e, principalmente, a necessidade de transferência de clientes entre estes servidores torna-se um requisito fundamental do módulo em desenvolvimento.

Assim, optou-se por incluir no módulo um método de migração entre servidores onde, indicando-se o cliente a ser transferido e o endereço IP do servidor de destino, o processo é realizado automaticamente através da utilização de mensagens internas de controle.

Uma forma de utilização desta funcionalidade está descrita mais adiante ao ser explicado o desenvolvimento do jogo "Lego Adventures Online" (item 4.4).

2.7 *MMG e o escopo do projeto*

O desenvolvimento de um módulo de redes para um *engine* de criação de jogos requer que diversos tipos de jogos multiusuário sejam suportados, não somente MMG. No entanto, devido a suas peculiaridades, este estilo de jogos serviu como base para a aquisição de requisitos para o projeto do módulo de redes discutido neste trabalho.

O projeto prevê a implementação de uma lógica de balanceamento de carga e divisão de usuários por áreas pré-definidas, especificamente o módulo deve suportar múltiplos servidores, onde cada um é responsável por uma área de jogo distinta. Essas áreas são conectadas entre si através de "portais", áreas delimitadas dentro do mapa do jogo que ao serem acionadas transferem o jogador para outra área (para outro servidor) previamente definida. Essa é uma solução bastante comum nos MMGs, como citado por [BERNARDES Jr., J. L. et al., 2003].

Outra característica importante no módulo de redes é o suporte a pré-processamento que permite ao desenvolvedor implementar tratamentos especiais ao jogo como camuflar os atrasos da rede através de técnicas como "*Dead Reckoning*" (discutido em 2.4).

Foge ao escopo deste projeto o desenvolvimento de um módulo responsável pelo controle de banco de dados necessário para se manter a persistência de estado do jogo, assim como interagir com o módulo de redes para autenticação segura de jogadores.

Também não cabe ao escopo do projeto a discussão de um módulo de segurança que implemente algoritmos de criptografia para dar suporte à segurança das mensagens trocadas pela rede por exemplo.

3 DECISÕES DE PROJETO

Neste item procura-se apresentar os argumentos baseados na literatura relacionada para justificar algumas escolhas de projeto, como o modelo de arquitetura adotado.

3.1 Arquiteturas de Rede

A escolha da arquitetura de redes sobre a qual estará alicerçado o desenvolvimento de um jogo maciçamente multiusuário é parte fundamental para o projeto de qualquer jogo deste gênero. Essa escolha terá grande influência na definição da quantidade máxima efetiva de usuários conectados simultaneamente, dos níveis de segurança contra usuários trapaceiros ("*cheaters*") e também da escalabilidade para permitir no futuro um crescimento do jogo.

Para a escolha dentre as diversas arquiteturas existentes atualmente, apresentadas a seguir, levou-se em conta os seguintes critérios:

- Custo de implementação e manutenção, performance do sistema: hardware e consumo de banda basicamente.
- Tolerância a jogadores trapaceiros: o grau de vulnerabilidade a trapaças determina a viabilidade de uma arquitetura para jogos on-line [CECIN et al. 2005].
- Escalabilidade
- Possibilidade de se manter um mundo persistente, isto é, se um jogador deixar de jogar num momento, caso ele volte mais tarde, seu jogador estará lá da mesma forma exceto pelas modificações feitas posteriormente por outros jogadores, mantendo a característica da ultima sessão de jogo. Esse é um requisito essencial para jogos MMG.

- Viabilidade. Se a arquitetura está bem madura, com a existência de material suficiente de conceitos, exemplos de implementação e casos de sucesso no uso em desenvolvimento de jogos para suportar o desenvolvimento do protótipo, além de materiais através das quais possamos concluir a viabilidade da integração de uma determinada arquitetura ao enJine.
- Confiabilidade e tolerância a falhas.

Nas próximas seções as principais alternativas de arquitetura serão descritas e analisadas em relação a esses requisitos.

3.1.1 Peer-to-Peer

Uma arquitetura *peer-to-peer* (p2p) oferece grandes vantagens sobre arquiteturas centralizadas para várias aplicações, incluindo MMGs. Descentralização com alta escalabilidade e ausência de um único ponto de falha são as principais vantagens dessa arquitetura [KARUNASEKERA et al., 2004].

Nesse tipo de arquitetura, cada usuário do p2p, também chamados de nó, é responsável pelo processamento de suas ações no jogo e pelo controle de seu estado. Cada jogador, executando uma ação, informa todos os nós aos quais ele está conectado, e assim a consistência do estado do jogo é mantida entre todos os participantes.

A capacidade de combinar o processamento e banda dos vários usuários pode permitir que um número muito elevado de usuários interajam ao mesmo tempo sem, no entanto, sobrecarregar um servidor ou então comprometer a rede e gerar atrasos significativos na troca de informações entre os jogadores.

No entanto, assim como na arquitetura cliente-servidor, existem alguns pontos problemáticos nesse tipo de arquitetura. O fato de os próprios jogadores controlarem

o estado do jogo implica em problemas com relação à segurança e tolerância a trapaça, assim como torna complicada a autenticação segura para que apenas usuários permitidos tenham acesso ao jogo.

Manter a consistência, ordenação e propagação de eventos é outra questão delicada nesta arquitetura, pois uma vez que os usuários controlam o estado do jogo, deve-se providenciar mecanismos para a distribuição segura de informações a estes usuários, isto é, cada usuário deve manter um conjunto de informações sobre o estado do jogo e ao mesmo tempo garantir que outros jogadores possam consultar suas informações para manter o jogo em um estado consistente. Isso pode ser alcançado implementando, por exemplo, subdivisões no mundo virtual, seccionando logicamente os usuários em grupos de interesse comum [BERNARDES Jr., J. L. et al., 2003], todos usuários em uma região do cenário do jogo compartilhariam o mesmo interesse sob informações relacionadas àquela área por exemplo.

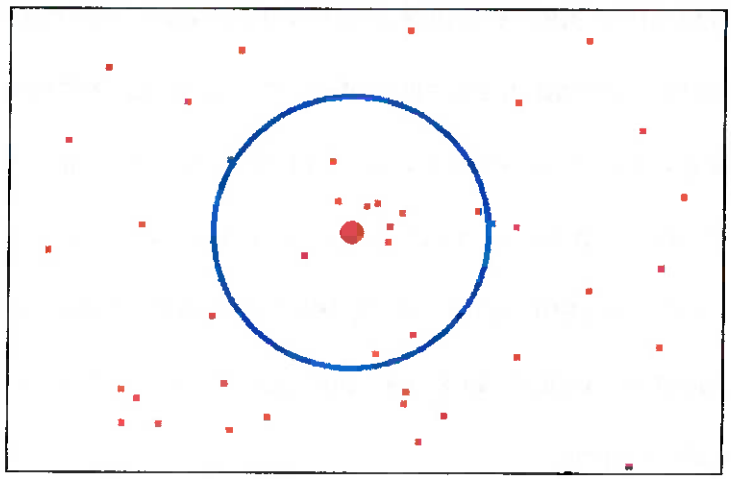

FIGURA 10 - Os peers (pontos vermelhos) pertencem a áreas de interesse (círculo azul) [HU, S. et al., 2004]

Um outro problema desta arquitetura está relacionada à persistência do mundo virtual do jogo. Mostrou-se extremamente difícil garantir a persistência de informações do jogo, principalmente caso haja uma falha catastrófica, isto é, uma

falha que envolvesse múltiplos jogadores desconectando-se repentinamente sem que haja chance de se copiar suas informações para os nós (cada usuário no p2p) redundantes, visto que não há uma fonte central dessas informações.

3.1.2 Cliente – Servidor

Uma arquitetura cliente-servidor é uma arquitetura bem estabelecida com vasto material de pesquisa disponível onde os jogadores (clientes) conectam-se a um servidor que será responsável por sincronizar suas ações e mantê-los atualizados sobre as mudanças ao seu redor. É uma arquitetura que permite a conexão de algumas centenas de jogadores, mas que pode ser estendida para alguns milhares através de soluções com suporte a clusters [HU, S. et al., 2004].

A maioria dos MMGs disponíveis comercialmente utilizam modelos centralizados como suporte à distribuição do jogo [CECIN, F. R. et al. 2005]. Essa centralização permite que se tenha alto controle sobre os estados do jogo garantindo assim que existam formas mais eficientes de tolerar jogadores trapaceiros.

Esta arquitetura caracteriza-se também por permitir a persistência dos dados do mundo virtual de forma simples. Uma vez que o servidor está funcionando como árbitro e por ele passam todas as ações realizadas pelos jogadores, é possível a cada mensagem recebida atualizar um banco de dados com toda a informação atualizada deste mundo.

No entanto, esta arquitetura apresenta alguns pontos negativos. Justamente por ser responsável por sincronizar a ação de todos os jogadores, o servidor pode tornar-se o gargalo dessa comunicação [BERNARDES Jr., J. L. et al., 2003] além de ser um único ponto de falha.

Um outro aspecto negativo relacionado a uma arquitetura cliente-servidor é o problema da escalabilidade. Um MMG tem como característica principal uma grande

quantidade de jogadores conectados simultaneamente e, uma vez que a quantidade de informação transferida entre os jogadores e o servidor é alta, a banda necessária para suportar este vasto número de jogadores é enorme [ASSIOTIS, M. et al.,2005]. Tipicamente, jogos cliente-servidor baseados em "avatares" (ou controle de um único objeto dinâmico por jogador), como Quake e Ultima Online [ORIGIN,2004], consomem, de forma aproximada, entre 2 Kbytes/s e 5 Kbytes/s por jogador [CECIN, F. R. et al. 2005].

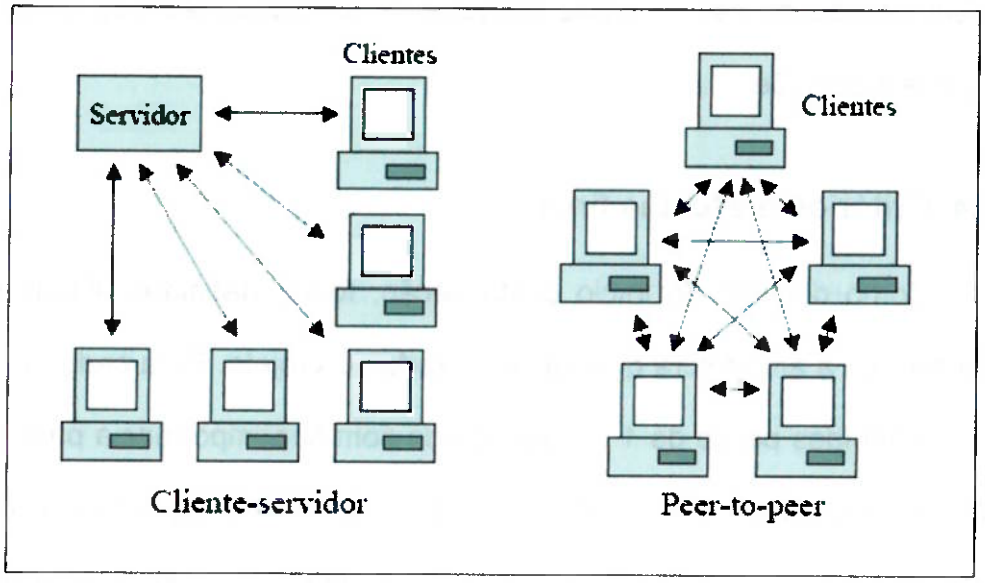

FIGURA 11 - Modelo de arquitetura para cliente/servidor e p2p [CECIN, F. R. et al. 2005]

3.1.3 Modelo híbrido

A terceira e última arquitetura pesquisada é na verdade uma combinação das duas descritas anteriormente, utilizando comunicação p2p entre os clientes e um servidor para realizar o controle de segurança, recuperação de falhas e autenticação segura, por exemplo.

Este modelo apresenta-se como um modelo que combina as características de segurança do modelo cliente-servidor com a alta escalabilidade e baixo custo da rede p2p.

Existem algumas soluções interessantes baseadas nesta arquitetura, como por exemplo, o projeto *FreeMMG* [CECIN, F. R. et al. 2005] que propõe técnicas de implementação visando sempre a segurança e escalabilidade, apresentando inclusive uma solução para recuperação de falhas catastróficas.

O grande problema desta a arquitetura hibrida, é a grande complexidade de implementação de cada módulo (segurança, sincronização p2p e com o servidor, etc.) que a compõe.

3.1.4 Critérios e escolha final

Como descrito no início desta seção, foram definidos alguns critérios para escolhermos a arquitetura que seria utilizada no projeto. Para cada um dos critérios, foram atribuídos pesos de 1 a 5 de acordo com sua importância para o tipo de jogo MMG, de onde foram extraídos os requisitos do módulo de redes, assim como para a integração com o enJine, levando em conta também os seus requisitos. Um peso maior implica em maior importância do critério.

Para cada arquitetura proposta foi atribuído, subjetivamente com base na análise da literatura relacionada, um valor de acordo com o critério em questão. Um valor mais alto, implica que aquela arquitetura atende de melhor forma àquele critério.

Critérios	Pesos	Peer To Peer	Cliente / Servidor	Híbrido
Custo: Implementação, manutenção	5	5	2	3
Performance: Consumo de banda	5	5	2	4
Tolerância a jogadores trapaceiros	4	1	5	3
Escalabilidade:	3	5	3	4
Mundo persistente:	5	1	5	3
Viabilidade:	5	2	5	2
Complexidade:	5	3	4	1
Confiabilidade e tolerância a falhas:	3	5	2	4
	Totais:	114	**125**	101

TABELA 3 - Tabela de escolha de arquitetura

Com as justificativas apresentadas na descrição de cada uma das arquiteturas, montou-se a tabela que indicou que a arquitetura baseada em Cliente e Servidor é a mais indicada para o projeto em questão.

4 IMPLEMENTAÇÃO

Esse item descreve a maneira como o UP foi aplicado ao projeto, as atividades desempenhadas e as descartadas, o que foi aplicado com sucesso ou fracasso e as principais decisões de projeto que foram tomadas.

Além disso, apresenta-se um explicação mais aprofundada da tecnologia NIO utilizada ao longo de toda implementação bem como uma visão mais detalhada do enJine.

4.1 UP aplicado ao projeto

4.1.1 As 6 boas práticas

As 6 boas práticas de desenvolvimento de software recomendadas pelo UP, explicitadas no item 2 do Apêndice A, foram aplicadas ao projeto, mas com diferentes prioridades.

O desenvolvimento iterativo do sistema foi a prática com a maior prioridade devido à complexidade do sistema e à escassez de tempo. Essa prática será detalhada em item posterior.

Com relação à gerência de requisitos, procurou-se a sua realização através de constante comunicação entre os membros da equipe, entre a equipe e o orientador (cliente), e do uso de uma ferramenta. Os requisitos a serem atendidos em cada etapa do projeto eram definidos em reuniões periódicas com o orientador e divulgados e documentados via e-mail.

Adicionalmente, a equipe utilizou o serviço disponível gratuitamente na Internet chamado *Jotspot* (http://www.jot.com) [JOT]. Trata-se de um *wiki*, uma página em que o conteúdo é facilmente editado pelos membros cadastrados.

Através desse serviço, os requisitos definidos eram expostos com a identificação do responsável por cada um, e os integrantes podiam destacar as tarefas finalizadas, podendo-se controlar o andamento de cada etapa.

O projeto não fez uso de uma metodologia formal para a criação de uma arquitetura baseada em componentes. Decidiu-se que isso não seria necessário porque o módulo iria manter a estrutura existente no enJine, cujo desenvolvimento procurou contemplar os requisitos alcançados com a componentização como flexibilidade, reaproveitamento de código e, principalmente, facilidade de compreensão através da simplicidade.

Já a modelagem visual do software foi realizado através de uma ferramenta aberta chamada StarUML(http://staruml.sourceforge.net/en/) [STARUML]. Essa ferramenta se propõe a ser uma alternativa viável a softwares proprietários como o Rational Rose, suporta os diagramas especificados pelo padrão UML 2.0, gerenciado pela OMG e, principalmente, possui todas as funcionalidades de que a equipe necessitou em termos de modelagem de *software*.

Os requisitos não-funcionais foram verificados através de alguns testes desenvolvidos pela equipe que foram realizados através do uso conjunto de:

- softwares desenvolvidos pelo grupo, que simulam o comportamento de aplicativos clientes desenvolvidos com o módulo criado, enviando contínua e repetidamente, mensagens ao servidor.
- software de análise de rede Wireshark (http://www.wireshark.org/) [WIRESHARK], que permite capturar os pacotes enviados e recebidos para analisar seu conteúdo.

As mudanças no software foram gerenciadas através do uso da ferramenta CVS para armazenamento e controle de versões, além da prática de divulgação de mudanças através de e-mails e de registros no *Wiki Jotspot*.

4.1.2 Fases

Esta seção visa detalhar as atividades realizadas em cada fase definido pelo UP, bem como justificar as atividades que não foram realizadas.

4.1.2.1 Fase de iniciação

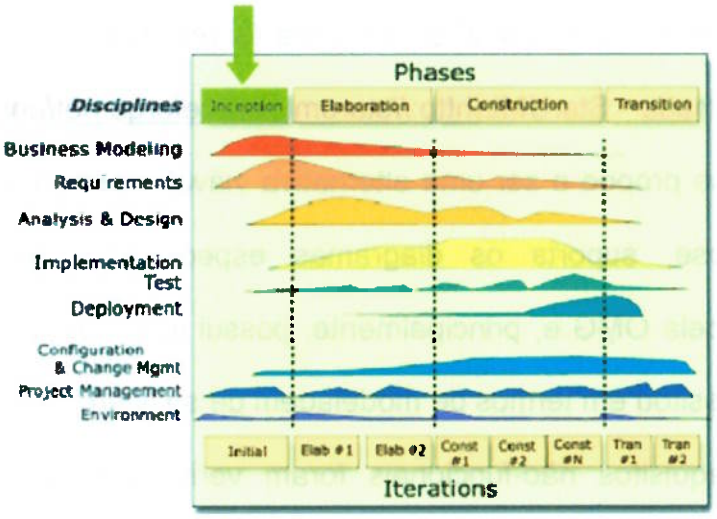

FIGURA 12 - Fase de iniciação (Extraída e adaptada de [IBM, 2005])

O foco dessa fase é a realização de decisões de negócio, deixando de lado detalhes de implementação. Na iniciação, as atividades mais importantes são a definição dos objetivos do projeto e dos riscos envolvidos, as pesquisas de mercado, as estimativas de custo e os estudos de viabilidade econômica.

Por motivos de clareza, dividimos as atividades em dois grupos que denominamos: atividades de negócio, que descreve as análises de risco, objetivos, viabilidade e escopo do projeto. E as atividades técnicas, que descreve a elaboração

de documentos de especificação, documentos de casos de uso e outras atividades de modelagem.

4.1.2.1.1 Atividades de negócio

Os objetivos do projeto foram inicialmente definidos em meados do mês de Janeiro e refinados durante essa fase inicial através do documento de proposta de projeto e do documento de especificação entregues para avaliação no primeiro quadrimestre do ano.

Com relação aos riscos envolvidos no projeto, é importante tê-los documentados para que sejam controlados a todo momento. No entanto, essa documentação não ocorreu e os riscos foram considerados e analisados durante as atividades de maneira informal, o que pode ser considerado uma das falhas de aplicação da metodologia por parte da equipe. Como trata-se, no entanto, de um projeto de porte, equipe de desenvolvimento e investimento relativamente pequenos, considerou-se essa abordagem adequada. [SMITH, 2002] afirma que em projetos com essas características as fases de iniciação e elaboração podem inclusive ser bastante reduzidas.

As pesquisas de mercado são importantes, pois possibilitam a identificação de necessidades do mercado que podem vir a ser exploradas comercialmente. Apesar disso não ser o foco em um projeto de formatura, a realização dessas pesquisas é importante por ser um fator motivador tanto para o estudante, pois esse se interessará em poder adquirir conhecimentos relacionados a assuntos valorizados pelo mercado, quanto para a universidade, que terá mais pessoas envolvidas com o estudo e solução de problemas atuais da sociedade.

Por isso, a atividade inicial realizada e que iria motivar o restante do projeto, foi uma pesquisa sobre o estado atual do mercado de jogos eletrônicos tanto no

Brasil quanto no restante do mundo. Como produto dessa atividade temos o documento intitulado "Proposta para o Trabalho de Conclusão de Curso" entregue em 01/02/2006 e cujo conteúdo foi aproveitado para a confecção desta monografia.

Estimativas de custo não foram realizadas porque o projeto não envolveu questões de patrocínio ou pagamento por parte de um cliente. Além disso, não houve nenhum gasto por parte dos estudantes e da universidade com itens que seriam considerados em projetos comerciais como pagamento de funcionários e aquisição de equipamentos e licenças de software, por exemplo. Os estudos de viabilidade econômica também não foram realizados pelas mesmas razões.

Após a aprovação dos aspectos de negócio e conseqüente permissão de início de projeto, uma atividade crucial a ser realizada é a confecção de um planejamento preliminar com os principais marcos do projeto. No projeto, isso foi feito através da criação de um cronograma chamado "Cronograma A" e que está presente no Apêndice B – Cronogramas deste documento. Como o tempo é o principal recurso investido nesse projeto, bem como seu principal limitante, esse cuidado com os cronogramas é plenamente justificado, ainda que não se trate de um projeto comercial.

4.1.2.1.2 Atividades técnicas

A principal atividade técnica recomendada para essa fase é a confecção de um documento de especificação preliminar com os requisitos mais importantes do sistema. Isso foi realizado tendo como resultado a confecção do documento de especificação entregue em 06/03/2006. Naturalmente, os requisitos especificados foram refinados durante o projeto.

Recomenda-se também a especificação dos casos de uso principais do sistema. Essa atividade foi realizada através da criação de um documento simples e

que não utiliza a notação formal (diagramas UML e especificação detalhada de cada caso de uso), contendo apenas a listagem e a breve descrição dos casos principais.

Optou-se por não fazer algo mais detalhado nessa fase devido à grande possibilidade de alteração dos casos de uso, pois se tratava de uma fase inicial em que faltava amadurecimento das idéias e base de conhecimento técnico.

Outra recomendação é a criação de um protótipo inicial. Porém, foi decidido que essa atividade seria descartada, pois se interpretou que essa recomendação do UP é direcionada para o desenvolvimento de outro tipos de sistemas, como o de um ERP, por exemplo, no qual seria interessante criar um protótipo para validar junto ao cliente o padrão da interface com o usuário do aplicativo. No entanto, um protótipo foi desenvolvido em uma fase futura do projeto;

4.1.2.2 Fase de elaboração

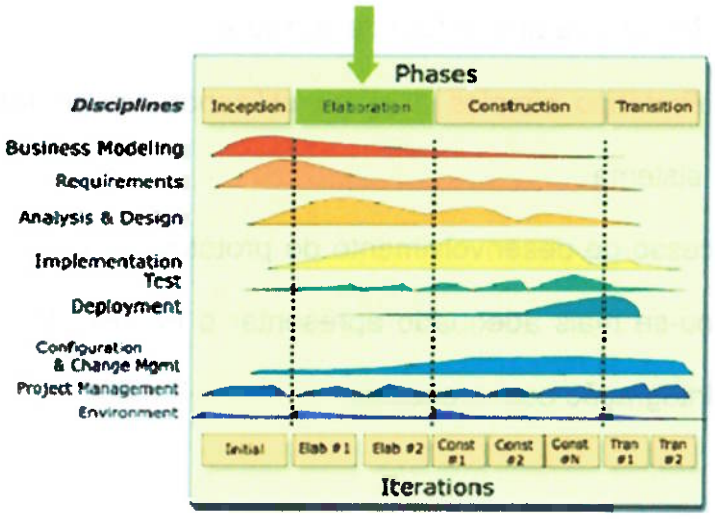

FIGURA 13 - Fase de elaboração (Extraída e adaptada de [IBM, 2005])

As atividades dessa fase do UP visam assegurar que a arquitetura, requisitos e riscos estão estabilizados para que se possa realizar estimativas mais precisas das fases posteriores.

É nessa fase que ocorre a identificação dos requisitos não-funcionais dos sistemas desenvolvidos. Isso foi realizado, em grande parte, baseando-se na especificação dos requisitos funcionais realizado na fase anterior. Estas informações não estão presentes em nenhum dos documentos entregues durante o ano, sendo adicionados diretamente a esta monografia.

Outra atividade é a determinação de todos os casos de uso e atores do sistema. Nesse projeto, em particular, o ator é sempre um jogo que utiliza o módulo para criar as funcionalidades de rede. Os casos de uso foram definidos em sua totalidade, no entanto não foram devidamente documentados com diagramas e detalhamento de cada caso. Isso acabou ocorrendo somente no início da fase de construção, o que pode ser considerado outra falha na aplicação do UP.

Apesar da importância dos itens anteriores, foi definido que os seguintes produtos esperados ao final dessa fase deveriam ser priorizados:

- Modelo da arquitetura de software
- Protótipo simples de jogo utilizando um protótipo da arquitetura do sistema

O processo de desenvolvimento do protótipo será detalhado em seguida. No entanto, julgou-se mais adequado apresentar o Modelo de arquitetura de software no item 4.3 (Integração com o enJine).

4.1.2.2.1 Protótipo

O desenvolvimento de um protótipo nessa fase do projeto é crucial para se testar a viabilidade técnica do sistema, pois através dele pode-se validar a

arquitetura definida e testar a tecnologia de implementação escolhida, dando base para a decisão de se continuar ou não o projeto.

Para esse trabalho, o protótipo desenvolvido foi um aplicativo onde cada usuário conectado a um servidor pode movimentar um cubo colorido através de um ambiente tridimensional, conforme pode ser observado na figura abaixo.

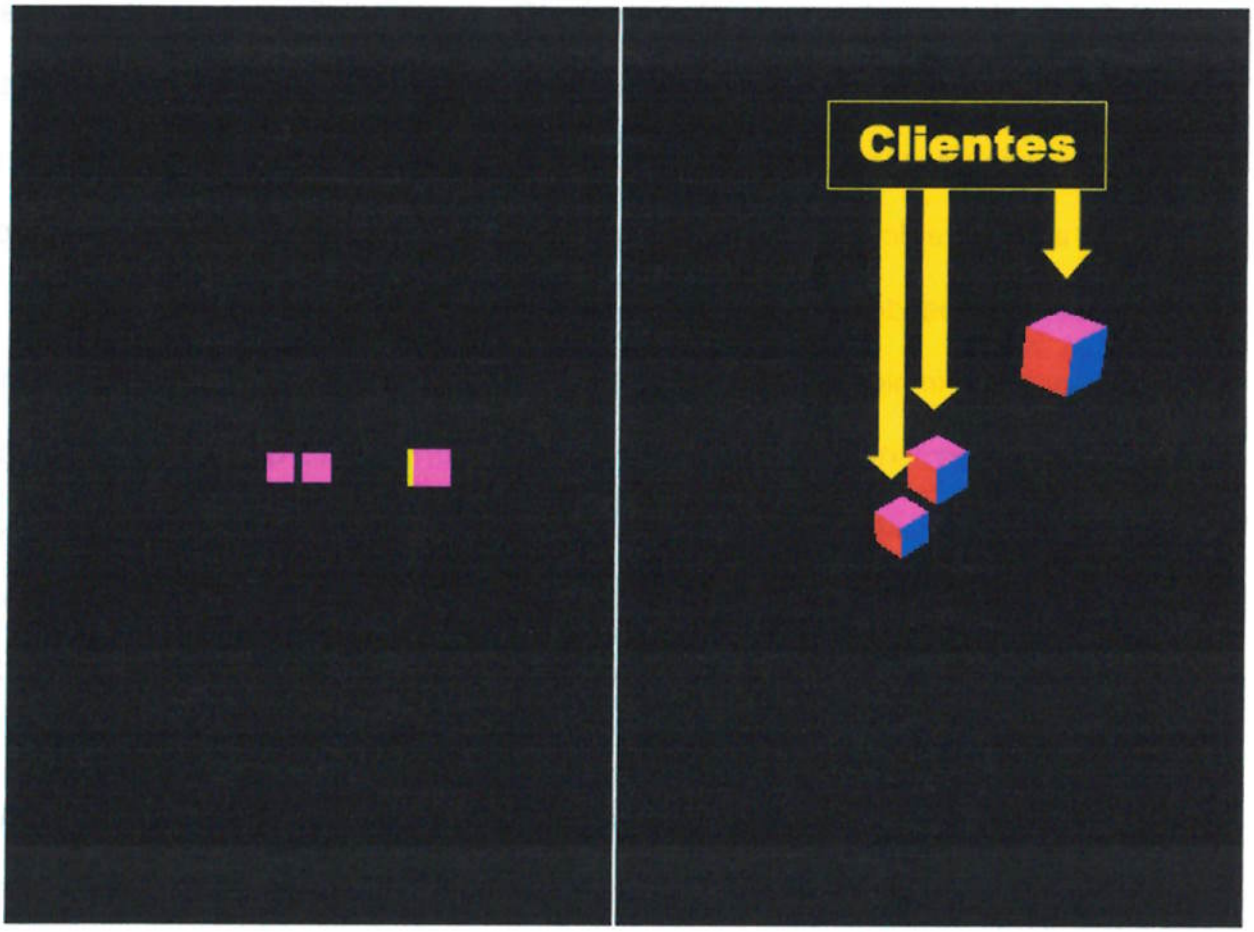

FIGURA 14 - Interface do protótipo

O aplicativo cliente da figura é responsável por enviar os comandos dos usuários ao servidor e por atualizar a posição utilizando as coordenadas recebidas do servidor. O controle de lógica do jogo como a atualização das coordenadas dos cubos é realizado no servidor.

A implementação desse aplicativo foi realizada com a utilização de uma tecnologia relativamente nova chamada Java NIO, explicada detalhadamente no item 4.2, e com uma abordagem de trabalho elaborada pelo grupo.

4.1.2.2.1.1 Abordagem de desenvolvimento

Após a escolha da tecnologia de implementação, o próximo passo foi a definição de uma abordagem de trabalho. Era preciso definir como seria realizada a divisão do trabalho e como os trabalhos individuais seriam integrados mais tarde.

Com relação à divisão do trabalho, foi adotado um método de trabalho onde cada integrante do grupo foi responsável pela criação de uma versão do protótipo de maneira independente. Isso foi interessante porque era difícil reunir a equipe para realizar uma implementação em grupo, já que um dos integrantes do grupo se encontrava no exterior e os outros estavam em período de estágio. Além disso, dessa forma os integrantes passaram a ter maior liberdade para aplicar as idéias que, analisadas sob seus pontos de vista individuais, foram consideradas as melhores. Isso fez com que naturalmente fosse gerada uma grande quantidade de idéias diferentes.

Outro ponto positivo do método foi o fato do conhecimento ter sido adquirido de maneira uniforme pelos integrantes da equipe. Isso porque muitas vezes os diferentes protótipos enfrentavam problemas semelhantes, criando oportunidades para discussão e troca de experiências.

Uma desvantagem clara da abordagem adotada é a alocação de todos os membros da equipe na mesma tarefa. Porém, analisando o ambiente em questão, pode-se afirmar que essa desvantagem foi sobreposta pelos ganhos com a utilização da abordagem. Isso porque, os desenvolvedores tinham uma idéia vaga do que poderia ser implementado e não tinham experiência com a tecnologia

utilizada e com desenvolvimento de aplicações cliente-servidor, o que faria com que a tarefa de confecção de um planejamento detalhado e uma divisão de trabalho eficiente fosse uma tarefa desgastante e que não geraria retornos satisfatórios, já que o planejamento seria, provavelmente, realizado de maneira inadequada e não iria contribuir para uma fase de desenvolvimento mais eficiente.

Para facilitar a integração no fim do desenvolvimento do protótipo adotou-se a ferramenta de controle de versão CVS. Os motivos para a escolha são a sua maturidade e confiabilidade por ser utilizado em projetos bastante conhecidos e de grande porte há algum tempo, a sua gratuidade e excelente suporte da comunidade, o suporte nativo ao CVS da ferramenta de desenvolvimento adotada (Eclipse) e o fato de evitar a necessidade de se providenciar um servidor para sua utilização já que pôde-se encontrar um serviço gratuito na Internet que oferece a funcionalidade necessária ao projeto, o CVSDude (http://cvsdude.com) [CVSDUDE]. Além disso, membros da equipe já tinham familiaridade com o uso dessa ferramenta.

4.1.2.3 Fase de construção

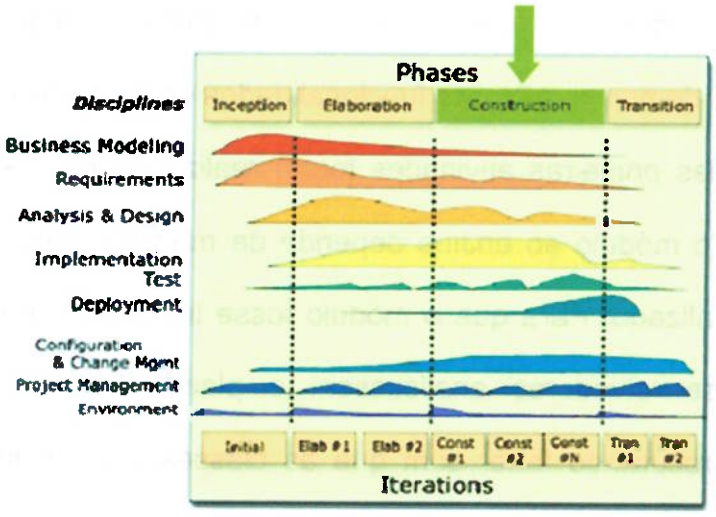

FIGURA 15 - Fase de construção (Extraída e adaptada de [IBM, 2005])

Na fase de construção, a implementação das funcionalidades ocorre de fato e é o foco do projeto, já que as duas fases iniciais do UP priorizam a especificação e modelagem do sistema.

Como explicado no item anterior, devido a falha na aplicação da metodologia, somente no início da fase de construção é que os casos de uso de módulo de redes foram devidamente detalhados. Esses casos de uso encontram-se no Apêndice D.

Durante essa fase de desenvolvimento, recomenda-se que, além da implementação, sejam realizados os testes funcionais do sistema para verificar se os requisitos definidos na especificação estão sendo atendidos.

No projeto de desenvolvimento do módulo de redes, essa fase envolveu quatro atividades:

1. Desenvolvimento do módulo com base no protótipo desenvolvido na fase anterior.
2. Integração do módulo ao enJine.
3. Realização de testes dos requisitos funcionais.
4. Testes dos requisitos não-funcionais.
5. Projeto e desenvolvimento do produto, jogo online para teste e demonstração das funcionalidades do módulo de redes desenvolvido.

As duas primeiras atividades foram realizadas em paralelo, uma vez que a integração do módulo ao enJine depende da maneira como o desenvolvimento do módulo é realizado. Para que o módulo fosse integrado ao enJine, era necessário que os desenvolvedores analisassem e planejassem sua codificação a todo momento, procurando fazer com que as classes criadas fossem estruturadas de maneira semelhantes às já existentes.

Isso foi feito porque seria muito custoso em termos de complexidade e tempo integrar o módulo ao enJine após a realização da codificação sem tal preocupação.

Os testes funcionais foram realizados à medida que as atividades acima eram desenvolvidas. No entanto, um teste completo só foi possível após o encerramento da codificação e a verificação de que o sistema estava relativamente estável após a correção de alguns bugs.

Esses bugs eram problemas que ocorriam durante o processo de interpretação de mensagens devido a uma mal formação da mesma.

Esses testes foram realizados com a utilização do protótipo, executando inúmeros clientes conectados ao servidor através da Internet. Cada integrante do grupo, a partir de sua casa, executou um determinado número de clientes.

As últimas duas atividades foram realizadas após a finalização dos testes dos requisitos funcionais. Assim, foi possível executá-los em paralelo já que se tratam de tarefas independentes, alocando-se cada atividade a uma dupla.

A dupla responsável pelos testes de requisitos não-funcionais deveria gerar resultados através das quais fosse possível avaliar o novo módulo de redes em termos de performance e escalabilidade.

Já os responsáveis pelo desenvolvimento do produto tinham como objetivo criar um jogo que utilizasse todos as funcionalidades do módulo desenvolvido para demonstrar sua viabilidade.

As últimas recomendações importantes do UP relacionadas à fase de construção são as entregas da primeira versão do software já integrada à plataforma de produção, de uma descrição da versão e de um manual do usuário.

Dentre essas atividades recomendadas, a criação do manual do usuário foi priorizada e executada durante e após o desenvolvimento do jogo de teste. A descrição da versão foi sendo realizada espontaneamente.

A integração à plataforma de produção foi desconsiderada, pois se trata de uma recomendação que não se aplica ao projeto em questão. Isso porque, no caso, o software desenvolvido é para distribuição e não implantação. Além disso, não existe uma plataforma que possa ser considerada de produção e foi considerado que o fato do software poder ser executado com facilidade em todos os ambientes em que foi testado é motivo suficiente para afirmar que essa recomendação do UP foi satisfeita.

4.1.2.4 Fase de transição

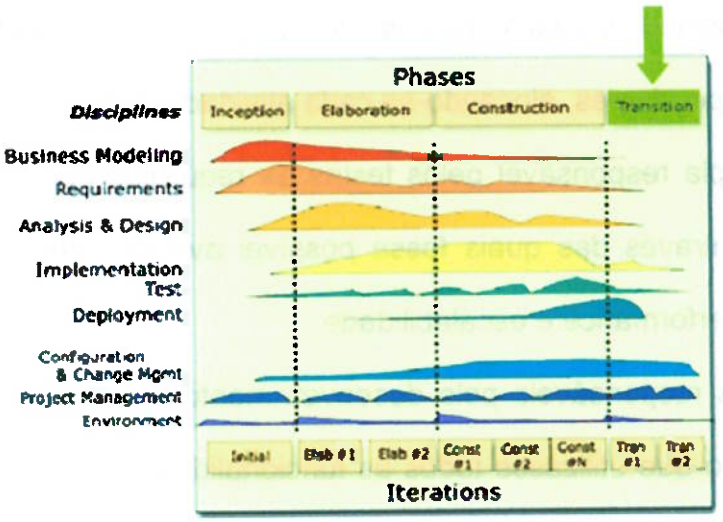

FIGURA 16 - Fase de transição (Extraída e adaptada de [IBM, 2005])

As atividades recomendadas na fase de transição, em geral, estão relacionadas com o processo de entrega do produto ao cliente ou a instalação no ambiente de produção, além da realização de treinamentos dos usuários e administradores do sistema.

Neste projeto, o papel de cliente é exercido pelo Interlab e uma das atividades planejadas foi a entrega do sistema, manual do usuário e monografia, e a instalação e demonstração do módulo no ambiente do laboratório.

Treinamentos de usuários e administradores não foram realizados. Optou-se por realizar somente o treinamento dos integrantes da equipe para a realização da apresentação do produto, tanto para o cliente quanto para a banca de avaliadores.

4.1.3 Iterações

O projeto realizou diversas iterações ao longo de suas fases. Isso foi facilitado por três fatores:

1. planejamento inicial proporcionou uma visão relativamente clara de como as fases poderiam ser divididas.
2. entregas periódicas de resultados cobrados pelo professor das disciplinas do projeto de formatura permitiu que definíssemos o que algumas das iterações deveriam gerar.
3. reuniões periódicas com o orientador do projeto auxiliaram na definição das iterações e na verificação dos produtos gerados.

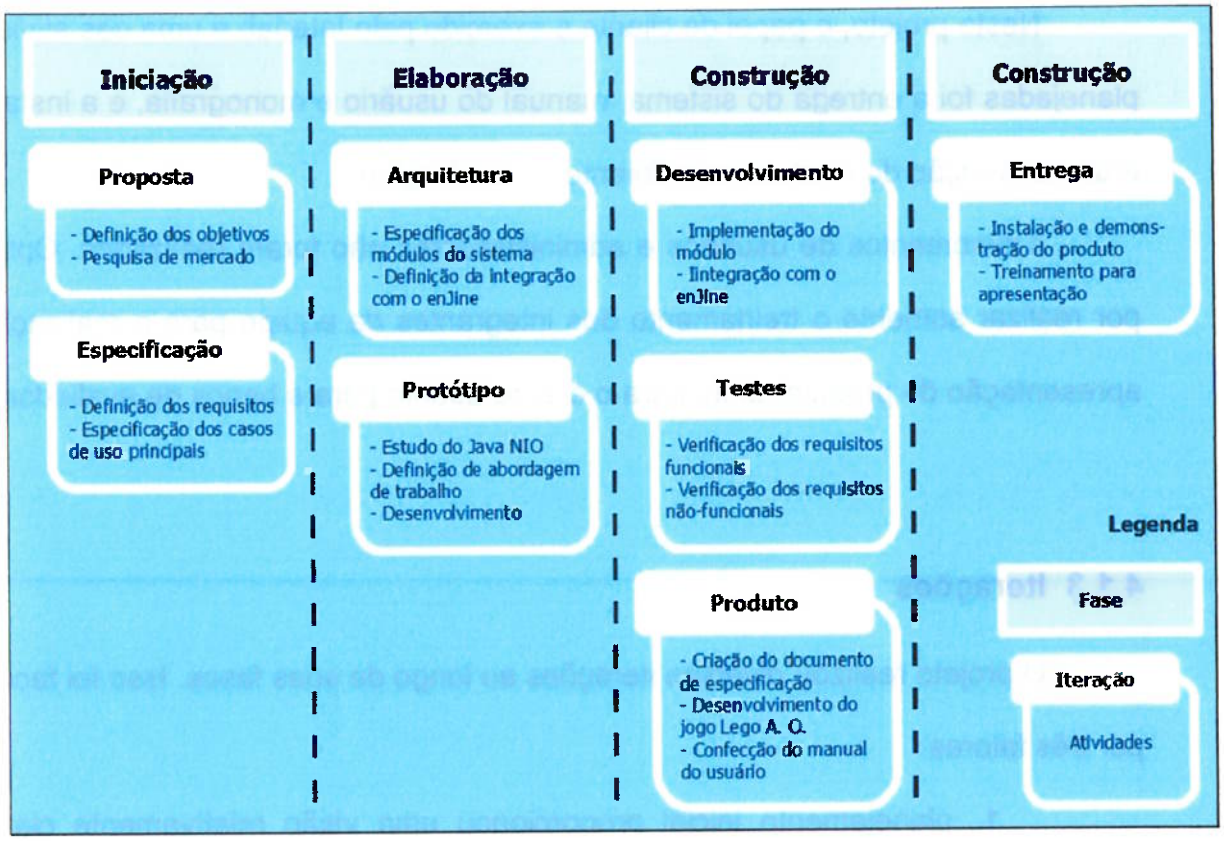

FIGURA 17 - Fases, iterações e atividades do projeto

Na fase de iniciação, houveram duas iterações, com a geração de um documento em cada uma delas. A entrega desses documentos foi uma exigência do professor da disciplina para possibilitar o acompanhamento do projeto, o que facilitou a identificação da necessidade dessas iterações.

A mesma situação ocorre na iteração de Arquitetura da fase de elaboração. Já a iteração de Prototipação, foi uma recomendação do orientador do projeto para familiarização com a tecnologia de desenvolvimento e teste da arquitetura proposta anteriormente. O produto gerado nessa fase foi um aplicativo de teste que permitia que clientes conectados ao servidor movessem cubos coloridos em um ambiente 3D.

Na fase de construção, pode-se observar a existência de três iterações. A primeira, de desenvolvimento, para finalizar a implementação do módulo e sua

integração ao enJine, cujo produto foi o mesmo aplicativo de teste da iteração de prototipação. Já as duas iterações seguintes dependiam do término do desenvolvimento e iniciaram após seu término. Foram duas atividades recomendadas pelo orientador e foram realizadas em paralelo. Os testes geraram um relatório com os resultados dos testes e o desenvolvimento do produto gerou o jogo "Lego Adventure Online" e um manual de utilização do novo módulo.

Por fim, a última iteração foi criada pela equipe para exercitar atividades relacionadas ao contato com clientes e teve como produto um material de apresentação do módulo de redes desenvolvido e do jogo "Lego Adventures Online".

4.2 Tecnologia de Implementação - Java NIO

O primeiro passo do desenvolvimento do protótipo foi a escolha da tecnologia de implementação. A linguagem JAVA oferecia a possibilidade de se criar aplicações de rede através da utilização em conjunto das classes dos pacotes java.io e java.net. Com o lançamento da versão 1.4 do J2SE, foi introduzida a API NIO.

Essa nova API é composta de 6 pacotes: java.nio, java.nio.channels, java.nio.charset, java.nio.channels.spi, java.nio.charset.spi e java.util. regex. Além disso, algumas classes de java.io e java.net foram modificadas para suportar a nova API.

O pacote java.nio não substitui o antigo java.io. Ambos se complementam e, em alguns casos, implementam funcionalidades semelhantes de maneira diferente, cabendo ao usuário decidir qual implementação é mais interessante para determinada aplicação.

A API NIO introduz novos recursos relativos à entrada/saída em geral, não se limitando a área de redes. No entanto, funcionalidades não relativas a redes, apesar de poderem ser utilizadas no decorrer do projeto, não serão explicadas neste documento, pois não fazem parte do escopo deste trabalho. Maiores informações a esse respeito podem ser encontrados em: http://java.sun.com/j2se/1.4.2/docs/guide/nio/ [NIO]

O NIO permite implementar lógica de entrada e saída de alta performance porque ele transfere para o sistema operacional a responsabilidade pela execução de atividades mais custosas, aumentando assim a velocidade da execução [TRAVIS, 2003].

Outra diferença fundamental entre o NIO e o java.io é que o primeiro lida com os dados em forma de blocos enquanto o último lida com eles na forma de fluxo, ou seja, um byte por vez. A vantagem de se tratar os dados na forma de blocos é que esse método é muito mais eficiente, por outro lado é um método mais complexo do que o tratamento de dados na forma de fluxo.

Um requisito muito comum de servidores, especialmente em servidores de jogos, é a capacidade de suportar múltiplas conexões simultâneas. Para cumprir esse requisito, um servidor precisa esperar pela chegada de mensagens de diversos canais de rede simultaneamente. Dessa forma, uma estratégia de multiplexação se faz necessária. As seguintes estratégias de multiplexação podem ser escolhidas:

- *Socket Polling*
- *Socket-per-thread*
- *Readiness Selection*

No caso de *Socket Polling* o servidor precisa "varrer" cada um de seus canais de comunicação verificando a chegada de mensagens. Essa estratégia é a mais

ineficiente e a que oferece menor escalabilidade, pois ao efetuar a varredura o servidor deixa de executar outras tarefas como, por exemplo, atualizar o estado do jogo.

A estratégia de *Socket-per-thread* é criar uma *thread* de execução para cada canal de comunicação existente, de modo que essa *thread* se responsabilize pela verificação de mensagens. É uma abordagem mais eficiente que a anterior, porém de escalabilidade limitada pela eficiência do *thread scheduler* que é o componente da linguagem de programação responsável pelo agendamento das *threads* de execução.

A última abordagem, que foi escolhida nesse projeto, é a *Readiness Selection*. Nessa abordagem, a responsabilidade de varredura de canais de conexão é transferida para o sistema operacional, de modo que o programa não precisa parar sua execução normal para fazer essa varredura. É o método que oferece maior escalabilidade, no entanto requer suporte do sistema operacional e da máquina virtual. [HITCHENS, 2006]

O principal recurso do NIO utilizado no projeto é de entrada/saída assíncrona ou não-bloqueante. Através desse recurso, é possível implementar a estratégia de *readiness selection* explicada anteriormente. Devido a essa característica, as chamadas de funções de verificação de chegada de mensagens são de retorno imediato, ou seja, o programa não fica bloqueado esperando a chegada de uma mensagem, continuando sua execução normal.

Em aplicações de rede cliente/servidor, uma vantagem clara desse comportamento é a possibilidade do servidor tratar diversas requisições de clientes através de uma única *Thread* de execução. Ao contrário, no método de entrada/saída bloqueante, cada nova conexão deveria ser tratada por uma nova

Thread de execução, o que dificultava a criação de aplicações de rede escaláveis, como por exemplo jogos maciçamente multiusuário, já que os recursos do servidor seriam consumidos rapidamente devido a grande quantidade de usuários. Isso acontece pois existe um consumo adicional de memória e processamento necessários para a criação de cada nova *Thread* de execução, bem como para seu gerenciamento.

Antes do desenvolvimento do protótipo foi realizado um estudo preliminar da tecnologia escolhida através do desenvolvimento de um pequeno aplicativo de troca de mensagens. Isso foi feito para verificar a existência de alguma característica da tecnologia que pudesse inviabilizar seu uso no projeto.

4.3 Integração com o enJine

Neste item, explica-se como as funcionalidades de redes implementadas foram incorporadas à arquitetura do enJine.

Vale relembrar aqui o que foi dito no item 2.2.1, que o requisito principal do enJine e de qualquer módulo que seja adicionado a ele é a simplicidade. Esse requisito foi levado em conta durante toda a implementação do módulo de redes, priorizando uma arquitetura mais simples.

4.3.1 Arquitetura

Ao estender as funcionalidades do enJine de forma clara e direta, desejou-se manter o nível de abstração mais alto possível.

O enJine já faz a abstração de um jogo através de sua classe Game, que contém todos os atributos e métodos para a implementação de um jogo sem funcionalidades de rede.

Sendo assim, o primeiro passo da implementação foi criar uma classe MultiplayerGame no pacote Core, herdada da classe Game, adicionando métodos e atributos usados para o recebimento de mensagens.

O próximo passo, já pensando na arquitetura cliente/servidor, foi criar as classes MultiplayerServer e MultiplayerClient no pacote Framework, que implementam os métodos abstratos definidos em MultiplayerGame, bem como as funcionalidades de um servidor e de um cliente respectivamente.

Tanto o MultiplayerServer como o MultiplayerClient utilizam a classe Connection, que fornece serviços de envio e recebimento de mensagens, e também a classe Listener, que é responsável pela verificação de chegada de mensagens pela rede.

A classe Listener é instanciada e executada em uma *thread* de execução diferente da *thread* do MultiplayerServer e MultiplayerClient, de modo que a verificação de mensagens não prejudica a execução das outras tarefas de um jogo, como por exemplo a renderização e o tratamento de colisões.

A interface MultiplayerMessage foi criada para oferecer uma abstração do conceito de mensagem. Ela possui apenas dois métodos, sendo um para montagem de mensagem e um para a desmontagem. Desse modo o programador fica livre para implementar o formato de mensagem que mais se enquadra em seu jogo, e pode otimizá-la da maneira que desejar.

Uma implementação padrão de MultiplayerMessage chamada DefaultMultiplayerMessage foi criada para facilitar o trabalho do programador. No entanto essa implementação é bastante simplificada e sem nenhuma otimização.

E finalmente foi criada a classe MultiplayerState herdada da classe GameState.

No enJine, a classe GameState representa uma fase do jogo, uma tela de abertura ou qualquer outro estado da aplicação. Isso é válido também para a classe MultiplayerState, com a diferença que essa classe também contém métodos para a interpretação de mensagens e o controle de elementos do jogo que precisam ser replicados em todos os clientes.

Optou-se por acrescentar essas funcionalidades ao GameState pois é possível que cada fase, ou estado do jogo, interprete mensagens de diferente formas.

A figura 17 abaixo mostra o diagrama de classes do módulo de redes. Alguns detalhes foram omitidos por motivos de clareza.

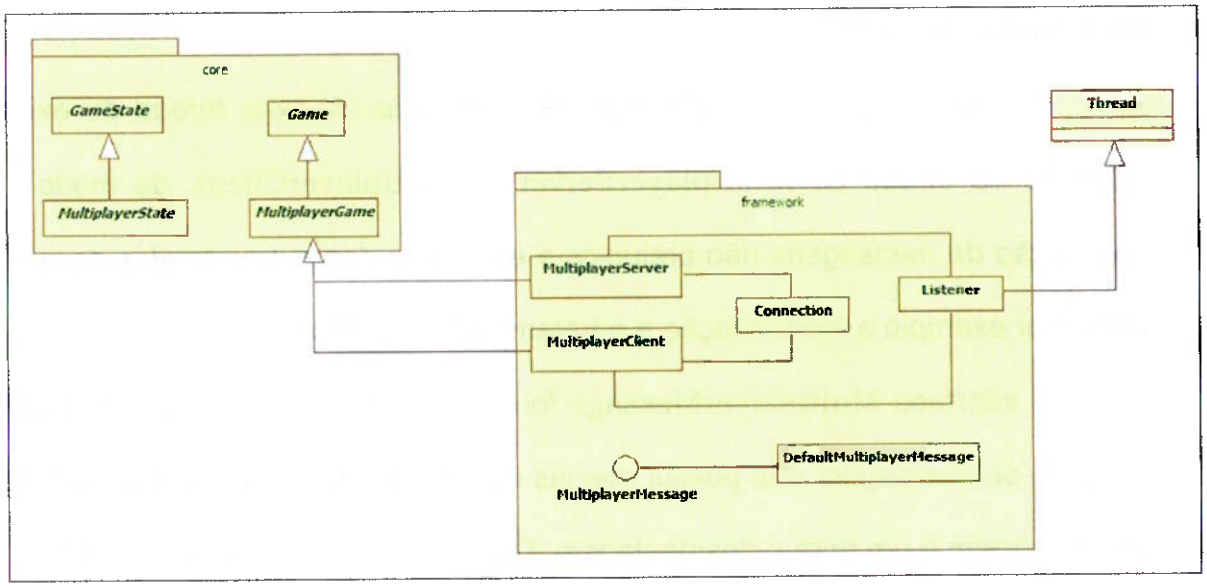

FIGURA 18 - Diagrama de classes do módulo de redes

4.4 *Lego Adventures Online*

Para realização dos testes de funcionalidade do módulo de redes do enJine, desenvolvido durante este projeto, foi também construída uma aplicação simples chamada "Lego Adventures Online". Essa aplicação é um jogo desenvolvido

utilizando o enJine com foco em suas funcionalidades de rede. Abaixo encontra-se descrito o processo de desenvolvimento do jogo e de que forma as funcionalidades de rede criadas durante o projeto estão inseridas no mesmo. Mais informações sobre o jogo foram colocadas em um documento de *game design* que se encontra no Apêndice C.

4.4.1 Estrutura básica

O jogo é foi desenvolvido com objetivo de demonstrar as funcionalidades do módulo de redes do enJine e portanto sua estrutura é bem simplificada. Ele é composto somente por bonecos de lego representando os jogadores e cubos representando os itens a serem coletados. O cenário foi removido para diminuir o tempo de renderização e evitar que este influenciasse nas análises do módulo.

4.4.2 Desenvolvimento

Pode-se dividir o processo de desenvolvimento do jogo em três fases distintas.

Inicialmente foi necessário transformar o jogo "Lego Adventures", antes para apenas um jogador, em um jogo que possuísse suporte para diversos jogadores. Baseando-se no protótipo desenvolvido anteriormente criou-se o servidor e o cliente para o jogo. Enquanto o servidor seria responsável pela sincronização dos jogadores e também pelo processamento das ações e das colisões, o cliente controlaria apenas a interação com o usuário (inputs e visualização). Esse método de divisão foi escolhido para demonstrar que o módulo desenvolvido permite a distribuição de tarefas entre cliente e servidor, prática bastante comum nos atuais MMG.

Nessa primeira fase foi definida também uma estrutura para as mensagens de comunicação entre cliente e servidor, bem como a função que realiza o tratamento

das mesmas. O módulo permite a cada desenvolvedor definir como será a mensagem de seu jogo, possibilitando diversos tipos de implementação. Neste caso foi utilizada a própria classe de mensagem padrão inclusa no módulo, onde as mensagens são strings de caracteres. A estrutura definida para cada mensagem foi "*tipo;id;var1;var2;var3;var4\n*" onde *tipo* indica o tipo da mensagem, *id* indica o número identificador do objeto que a gerou ou ao qual ela se refere e os campos *var1* até *var4* possuem valores diferentes para cada tipo de mensagem de acordo com sua necessidade. Os campos são separados pelo caractere ';' e ao término da mensagem é colocado o caractere '\n' para facilitarem a manipulação das mesmas. Abaixo encontram-se alguns exemplos de mensagens:

- rotação do jogador (tipo;id;direção\n): "2;10;L\n"
- atualização do estado inicial (tipo;id;tipo_obj;pos_x;pos_y;pos_z): "4;7;P;7.3;0;-10.5"

Numa segunda etapa foram implementadas basicamente as regras do jogo. Assim, foram criadas as funções responsáveis por contagem dos pontos, geração de itens, e tratamento de colisões.

Por fim, foi melhorada a parte visual do jogo acrescentando-se *overlays* que apresentam a pontuação atual de cada jogador. Foram também desenvolvidas algumas funcionalidades presentes em jogos atuais, como o pré-processamento e a migração entre servidores, para mostrar que o módulo de redes desenvolvido dá suporte também a esses métodos de otimização. Abaixo encontra-se uma descrição mais detalhada de como ambos foram implementados no jogo.

4.4.2.1 Pré-processamento

Devido ao fato do tratamento das ações estar localizado no servidor todo movimento solicitado pelo cliente seria realizado com um atraso correspondente ao

tempo de envio/retorno das mensagens para o servidor mais um tempo de processamento da mensagem no mesmo. O que poderia provocar demoras visíveis entre o apertar da tecla e a respectiva movimentação do personagem. Para evitar isso foi implementado o pré-processamento que consiste simplesmente na execução do movimento independente do recebimento de resposta do servidor. Assim, no momento que o jogador aperta a tecla uma mensagem é enviada para o servidor indicando a ação a ser realizada e, antes mesmo do recebimento de uma resposta, a ação é executada no cliente. Caso a mensagem de resposta não seja recebida ou indique a impossibilidade de execução da ação esta é desfeita.

4.4.2.2 Migração entre servidores

Foi criado também um segundo servidor com o mesmo ambiente que o primeiro para que fosse possível demonstrar a migração entre servidores. O processo de migração é bem simples. Um evento gera a solicitação de mudança de servidor e envia esta para o cliente. No servidor o tratamento dessa solicitação consiste simplesmente na exclusão desse cliente da lista de clientes conectados e consequente envio de mensagens para os demais clientes informando-os desse exclusão. Já no cliente, a solicitação provoca uma desconexão do atual servidor e uma nova conexão no servidor indicado pela mensagem. No "Lego Adventures Online", quando o personagem passa de uma determinada posição do cenário esse evento e disparado solicitando a migração.

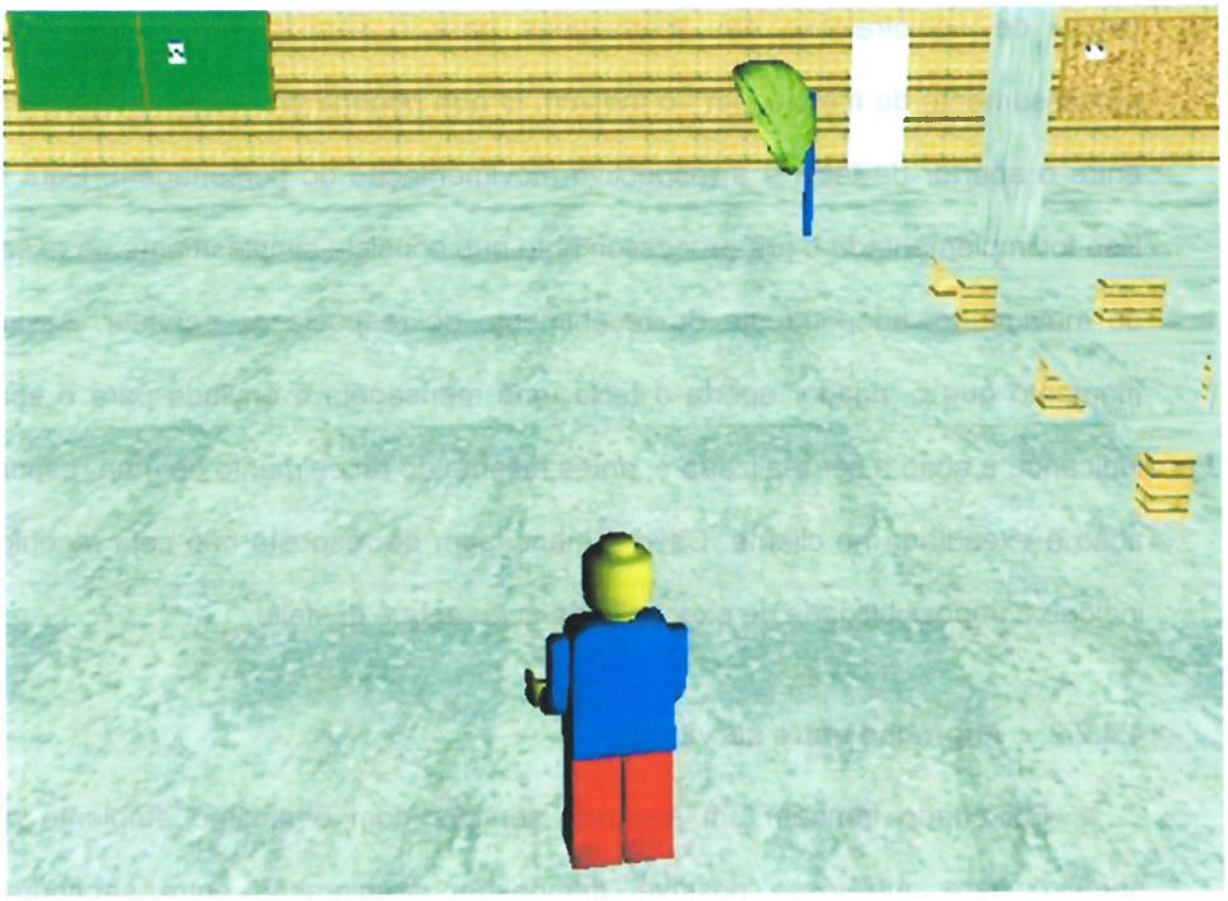

FIGURA 19 - Tela de jogo do Lego Adventures

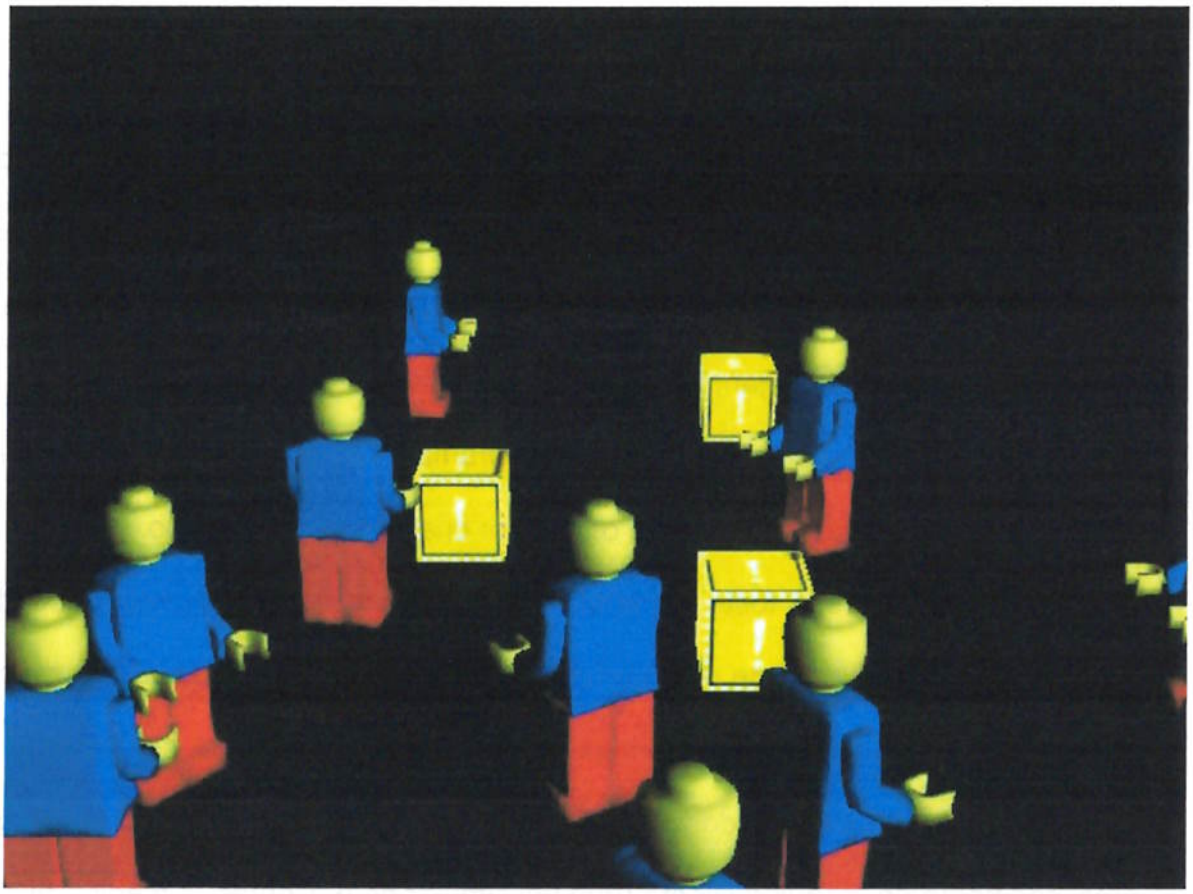

FIGURA 20 - Tela do jogo Lego Adventures: jogadores e itens (cenário omitido)

5 RESULTADOS

Para a realização de testes de carga relacionados ao módulo desenvolvido foi criada uma classe auxiliar no protótipo implementado durante a etapa de elaboração do projeto. Esta classe *ClientRobot* permite que um número **n** configurável de clientes sejam instanciados em *threads* separadas.

Cada um desses **n** clientes está programado para enviar aproximadamente uma mensagem a cada 100ms para o servidor criado para o protótipo.

O objetivo desse teste foi analisar o tráfego no servidor, coletando dados sobre consumo de banda de *upstream* e *downstream*, ou seja, o consumo de banda no sentido servidor/clientes e no sentido clientes/servidor, respectivamente.

Uma desvantagem desta abordagem é o fato de se sobrecarregar o processamento da máquina cliente (que vai simular os **n** clientes) à medida que esse número **n** aumenta.

O servidor foi executado em uma máquina AMD Athlon64 3000 enquanto os clientes foram simulados em um processador Intel Pentium Duo 1.66Ghz, ambas máquinas possuíam 1Gb de memória RAM e uma conexão à Internet de 256Kbps.

Para a captura dos pacotes foi utilizado o aplicativo Wireshark (http://www.wireshark.org) [WIRESHARK]. Todos os pacotes foram capturados no servidor e em seguida analisados.

Segue abaixo os dados obtidos:

Teste 1: Servidor - Pacotes Capturados: 5000								
Número de Clientes Simulados		2	4	6	8	10	12	14
UPSTREAM	Tamanho médio de Pacote (bytes)	89,837	121,073	143,993	168,882	206,887	246,632	289,985
	Média de Bytes/s	1682,819	4343,006	7455,94	11232,04	15007,68	15149,09	15220,94
DOWNSTREAM	Tamanho médio de Pacote (bytes)	66,17	65,933	65,379	65,242	66,013	70,15	74,853
	Média de Bytes/s	1236,781	2358,514	3394,348	4388,04	4773,33	4280,635	3892,952
Tamanho do Pacote (bytes)		Contagem (%)						
UPSTREAM	0-19	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
	20-39	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
	40-79	8,05%	6,96%	8,74%	8,25%	4,59%	5,35%	4,78%
	80-159	91,83%	87,68%	49,82%	31,31%	13,58%	6,10%	9,04%
	160-319	0,12%	5,12%	40,84%	59,07%	75,75%	78,06%	48,25%
	320-639	0,00%	0,24%	0,60%	1,01%	5,51%	9,65%	37,53%
	640-1279	0,00%	0,00%	0,00%	0,28%	0,56%	0,76%	0,36%
	1280-2559	0,00%	0,00%	0,00%	0,08%	0,00%	0,08%	0,04%
Tamanho do Pacote (bytes)		Contagem (%)						
DOWNSTREAM	0-19	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
	20-39	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
	40-79	99,52%	99,36%	99,32%	99,12%	99,00%	95,87%	77,84%
	80-159	0,48%	0,64%	0,68%	0,88%	1,00%	4,01%	21,63%
	160-319	0,00%	0,00%	0,00%	0,00%	0,00%	0,12%	0,52%
	320-639	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
	640-1279	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
	1280-2559	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%

TABELA 4 - Dados relativos à captura de pacotes enviados e recebidos pelo servidor.

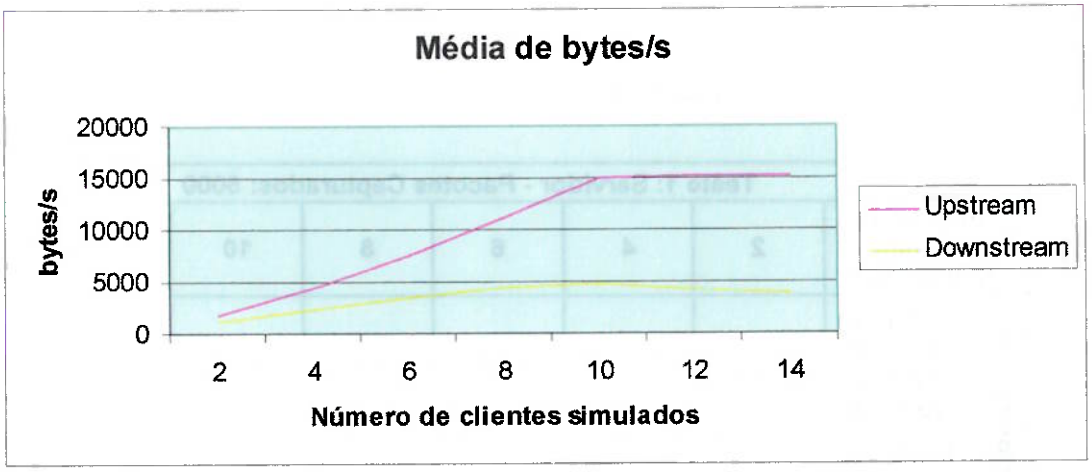

FIGURA 21 - Média de bytes/s em função do número de clientes simulados.

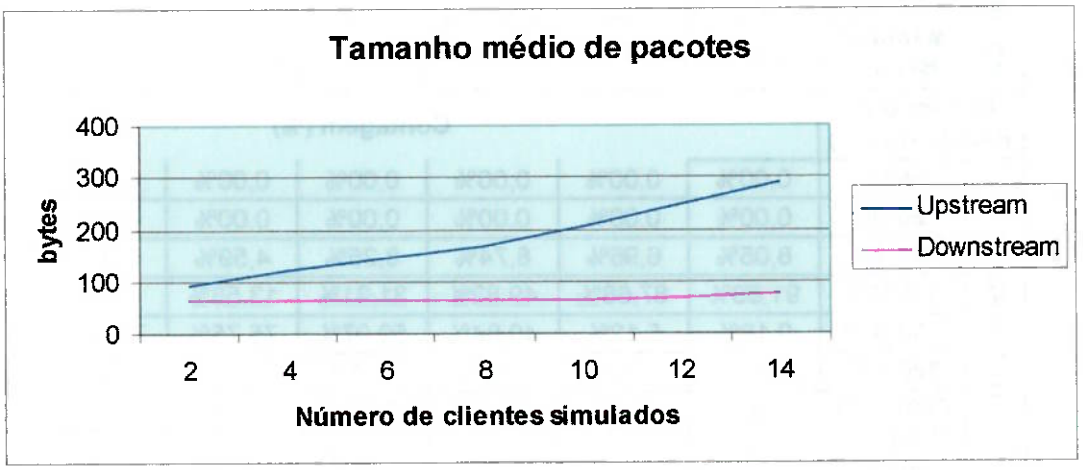

FIGURA 22 - Tamanho médio de pacotes

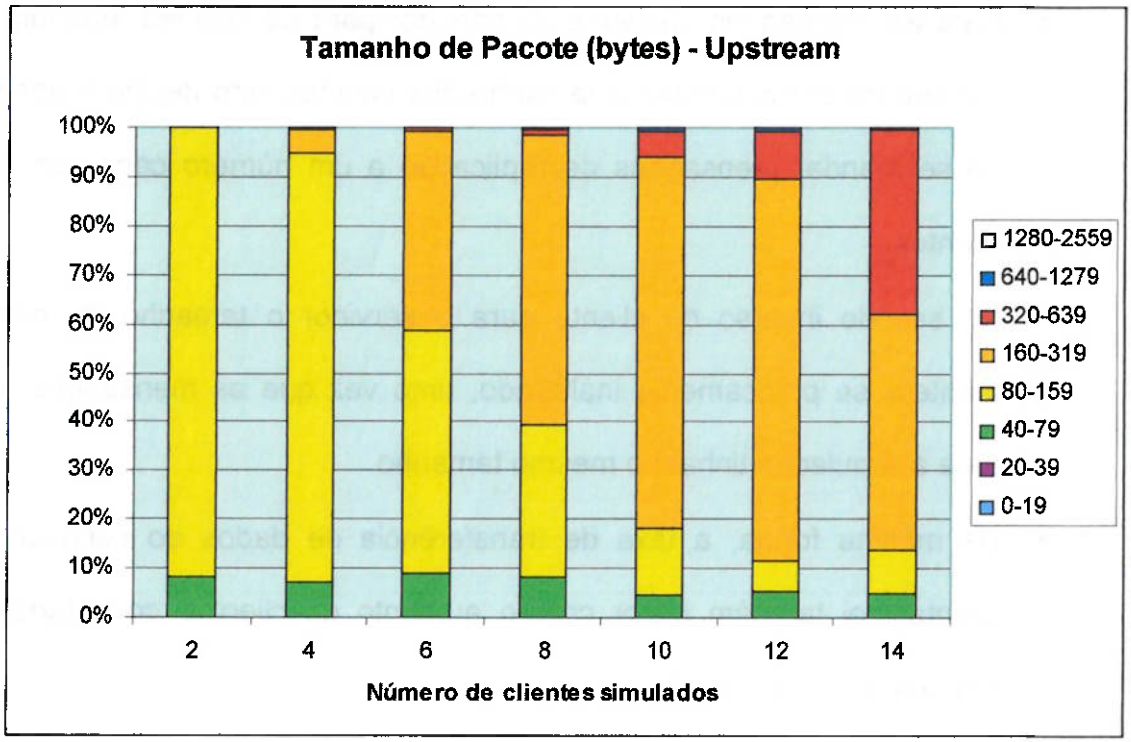

FIGURA 23 - Porcentagem de pacotes de determinado tamanho por faixa (upstream).

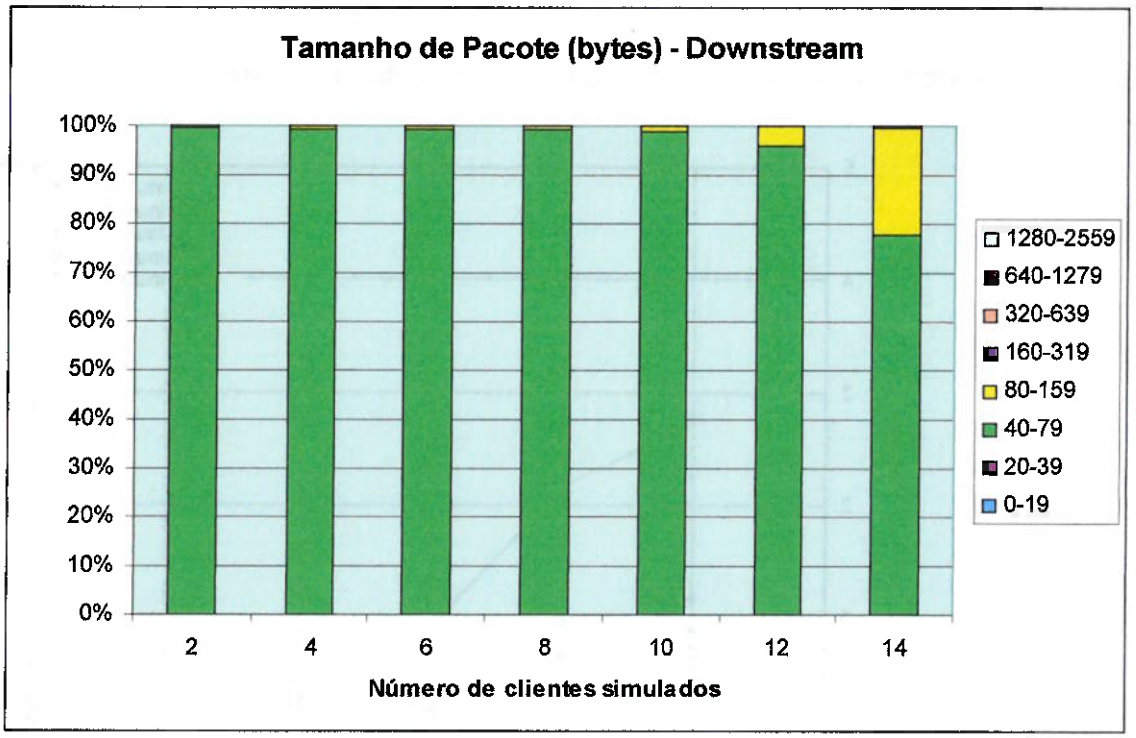

FIGURA 24 - Porcentagem de pacotes de determinado tamanho por faixa (downstream).

De acordo com os dados apresentados chegou-se às seguintes conclusões:

- Para as mensagens enviadas do servidor para os clientes, quando maior o número de clientes maior o tamanho dos pacotes, isto devido à necessidade de se mandar mensagens de replicação a um número cada vez maior de clientes.
- No sentido inverso do cliente para o servidor o tamanho de mensagens manteve-se praticamente inalterado, uma vez que as mensagens utilizadas para a simulação tinham o mesmo tamanho.
- Da mesma forma, a taxa de transferência de dados do servidor para os clientes foi também maior com o aumento de clientes conectados, o que também já era esperado.
- No gráfico 18 foi observado que a partir de 10 clientes simultâneos, a taxa de *upstream* do servidor tende a permanecer no mesmo patamar. Concluímos que esse fato ocorre por que, na plataforma de testes utilizada, a conexão do servidor não suporta taxas mais elevadas de *upstream*.

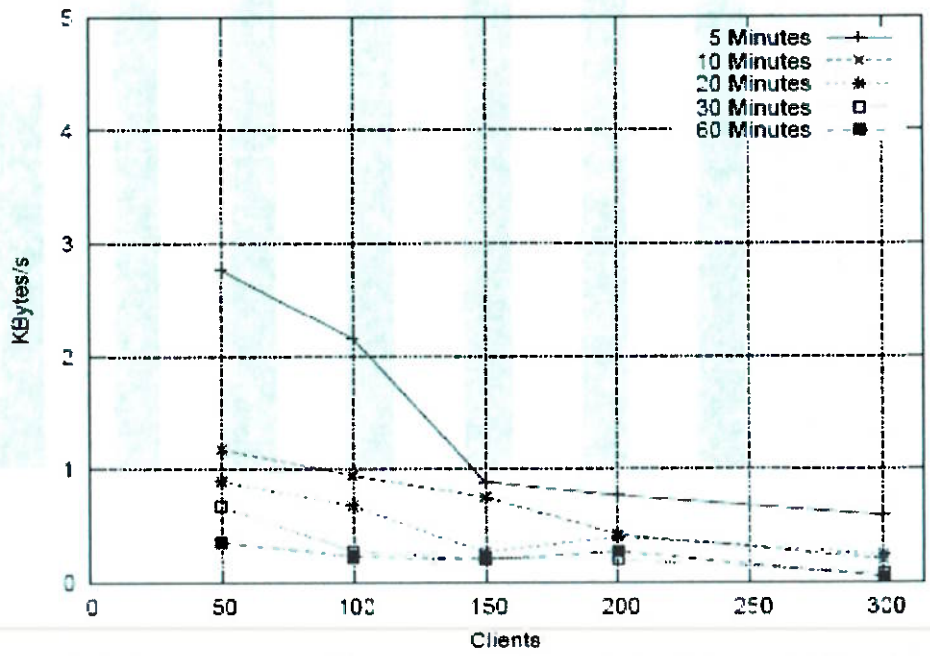

FIGURA 25 - Transferência de dados pelo número de clientes conectados (arquitetura híbrida). [CECIN, F. R. et al. 2005]

Em uma arquitetura híbrida os gráficos de consumo de banda no servidor pelo número de clientes apresentam uma curva decrescente, contrapondo-se ao gráfico apresentado para o modelo cliente-servidor escolhido como podemos ver na figura 18. Isso ocorre basicamente devido às replicações de estado serem realizadas não só pelo servidor, mas também por cada um dos clientes sendo que quanto mais clientes conectados menor a carga pela qual o servidor é responsável. Já na arquitetura cliente-servidor a característica é o aumento do consumo de acordo com o aumento do número de clientes conectados, como apresentado na figura 23 abaixo retirado de [KIM, Jaecheol et al, 2005] e na figura 19 apresentada anteriormente.

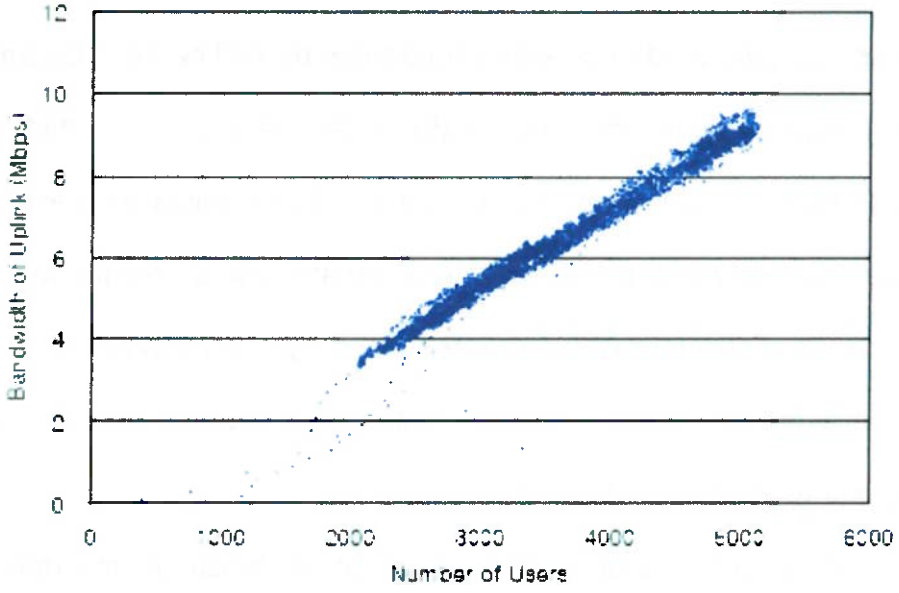

FIGURA 26 - Transferência de dados pelo número de clientes conectados (arquitetura Client - Server). [KIM, Jaecheol et al, 2005]

6 CONCLUSÕES

Ao longo deste trabalho foi apresentado o projeto, implementação, integração e testes do módulo de redes do enJine.

Além do seu requisito de simplicidade de uso, herdado do próprio enJine, os demais requisitos foram definidos com base em pesquisas realizadas sobre jogos do gênero maciçamente multiusuário (MMG), por ser um gênero de crescente popularidade atualmente.

No entanto, percebeu-se ao final desse trabalho que o módulo de redes desenvolvido ainda está longe de suportar o desenvolvimento de um jogo desse gênero. Isso se deve principalmente ao fato de que funcionalidades essenciais para esse tipo de jogo ainda não estão presentes no enJine nem faziam parte do escopo desse projeto, como um módulo de acesso a banco de dados para realizar o controle de persistência do mundo virtual e funcionalidades para implementação de políticas de segurança e criptografia na transmissão de mensagens.

Além disso, através de testes de carga, verificamos que a capacidade do sistema ainda não atingiu um nível satisfatório criando, portanto, a necessidade de se pensar em soluções de otimização.

Apesar disso, o objetivo principal de se construir uma ferramenta que possa ser usada para fins didáticos foi atingido. Com o desenvolvimento do jogo *Lego Adventures Online* o grupo comprovou que é possível criar, em um curto período de tempo, um jogo *online* completo.

O Processo Unificado teve um papel fundamental no direcionamento do projeto, dando metas ao grupo em cada uma de suas etapas e tratando essas etapas em uma seqüência bem definida. No entanto o grupo teve dificuldades em

utilizar os casos de uso para satisfazer os requisitos do módulo, pois as funções dos atores ficavam restritas a funcionalidades providas pelo módulo e não representavam uma seqüência de ações em si.

6.1 Trabalhos futuros

Muita coisa ainda pode ser desenvolvida e aprimorada com relação ao módulo de redes e o enJine. Módulos como o de banco de dados e camadas de segurança podem ser utilizados em conjunto com o módulo de redes para aprimorar as funcionalidades providas ao desenvolvimento de jogos *multiplayer* (em especial os MMG), por exemplo.

O desenvolvimento de um módulo de redes para suportar jogos com muitos jogadores simultâneos é um tema muito complexo e amplo e por isso oferece inúmeras abordagens de implementação, arquiteturas, protocolos e funcionalidades, assim, muitas otimizações podem ser realizadas sobre o módulo desenvolvido e sua integração com o engine.

REFERÊNCIAS BIBLIOGRÁFICAS

Assiotis, M.; Tzanov, V. **A Distributed Architecture for Massive Multiplayer Online Role-Playing Games,** 2005.

ARONSON, J. **Dead Reckoning: Latency Hiding for Networked Games,** 1997. Disponível em <http://www.gamasutra.com/features/19970919/aronson_01.htm>. Acesso em 1 dez. 2006.

BECK, K. **Programação extrema aplicada: acolha as mudanças.** 2005. Bookman.

BERNARDES Jr., J. L. et al. **A Survey on Networking for Massively Multiplayer Online Games,** 2003

BLIZZARD. **World of Warcraft,** 2004.
Disponível em <http://www.worldofwarcraft.com/index.xml>.
Acesso em: 22 nov. 2006.

CAELUM. Disponível em <http://www.caelum.com.br/caelum/apostila/caelum-java-objetos-fj11.pdf>. Acesso em 02 dez.2006.

CALTAGIRONE, S.; Keys, M.; Schlief, B.; Willshire. M. J. **Architecture for a Massively Multiplayer Online Role Playing Game Engine,** December 2002.

CECIN, F. R.; Martins, M. G.; Barbosa, J. L. **FreeMMG: A Hybrid Peer-to-Peer and Client-Server Model for Massively Multiplayer Games,** 2005.

CVSDUDE. Disponível em <http://cvsdude.com>. Acesso em 02 dez. 2006.

DEMACHY, Thomas. **Extreme Game Development: Right on Time, Every Time,** 2003. Disponível em <http://www.gamasutra.com/resource_guide/20030714/demachy_01.shtml>. Acesso em 22 nov. 2006.

ENJINE. Disponível em <http://enjine.incubadora.fapesp.br/portal/Downloads/Documentation/Tutorials/>. Acesso em 02 dez. 2006.

ESA. **ESSENTIAL FACTS ABOUT THE COMPUTER AND VIDEO GAME INDUSTRY,** 2005.
Disponível em <http://www.theesa.com/files/2005EssentialFacts.pdf>.
Acesso em 02 dez. 2006.

FLOOD, Kevin. **Game Unified Process (GUP),** 2003.
Disponível em <http://www.gamedev.net/reference/articles/article1940.asp>.
Acesso em 22 nov. 2006.

GAMELOCALIZATION. **Publisher revenue - how much and where do they make it?,** 2005 abr. Disponível em <http://www.gamelocalization.net/facts_revenue.html>.
Acesso em: 02 dez. 2006.

GAUTHIERDICKEY, C.; Lo, V.; Zappala, D. **A Fully Distributed Architecture for Massively Multiplayer Online Games**.

GRAVITY. **Ragnarok Online,** 2002.
Disponível em <http://www.ragnarokonline.com/>.
Acesso em 22 nov. 2006.

HARDCODEDGAMES. **Can Blizzard survive World of Warcraft?,** 18 mar. 2005.
Disponível em: <http://hardcodedgames.com/mmpgamedev/2005/03/can-blizzard-survive-world-of-warcraft.html>.
Acesso em 02 dez. 2006.

HITCHENS, R. **How to Build a Scalable Multiplexed Server with NIO,** 2006.
Disponível em <http://developers.sun.com/learning/javaoneonline/2006/coreplatform/TS-1315.pdf>.
Acesso em 17 nov. 2006.

HU, S.; Liao G. **Scalable Peer-to-Peer Networked Virtual Environment,** 2004.

IBM. **Rational Unified Process**
Best Practices for Software Development Teams, 2005.
Disponível em <http://www-128.ibm.com/developerworks/rational/library/253.html>.
Acesso em 14 abr. 2006.

INTERLAB3D.
Diponível em <http://interlab3d.incubadora.fapesp.br/portal/documents/tutorials/>.
Acesso em 02 dez. 2006.

IGDA. **Persistent World Games White Paper**. 2004.
Disponível em <http://www.igda.org/online/IGDA_PSW_Whitepaper_2004.pdf>.
Acesso em 17 nov. 2006.
KARUNASEKERA, S. et al. P2P Middleware for Massively Multi-player Online Games, 2004.

JOGOSBR. Disponível em <http://www.jogosbr.org.br>. Acesso em 02 dez. 2006.
KENT, S. L. **Making an MMOG for the Masses,** 2004.
Disponível em <http://www.gamespy.com/amdmmog/week3/>.
Acesso em 17 nov. 2006.

JOT. Disponível em <http://www.jot.com>. Acesso em 02 dez. 2006.

KIM, Jaecheol et al. **Traffic Characteristics of a Massively Multi-player Online Role Playing Game,** 2005.

LLIOPS, N. **Software Engineering Roundtable Summary,** 2004.
Disponível em < http://www.gamesfromwithin.com/articles/0403/000015.html>.
Acesso em 23 nov. 2006.

MMOGCHART. **Total Active Subscribers.**

Disponível em <http://mmogchart.com/>.
Acesso em 02 dez. 2006.

NAKAMURA, R. et al. **Case Study: Applying Extreme Programming Techniques in the Development of a Computer Game,** 2005.

NAKAMURA, R. et al. **enJine: Architecture and Application of an Open-Source Didactic Game Engine,** 2006.

NIO. Disponível em <http://java.sun.com/j2se/1.4.2/docs/guide/ nio/>.
Acesso em 02 dez. 2006.

NORTON, Darrel. **Extreme programming overview,** 2005.
Disponível em <http://codebetter.com/blogs/darrell.norton/articles/50340.aspx>.
Acesso em 14 abr. 2006.

REZENDE V. L., Ronaldo. **Obtendo qualidade de software com o RUP.**, 2004.
Disponível em <http://www.javafree.org/content/view.jf?idContent=7>.
Acesso em 14 abr. 2006.

SEAL, Sarit. **Rational Unified Process (RUP)**.
Disponível em <http://www.nasscom.org/download/Rational_Unified_Process.pdf>.
Acesso em 14 abr. 2006.

SMITH, J. **A Comparision of RUP and XP**. Rational Software, 2002. (White Paper).

STARUML. Disponível em <http://staruml.sourceforge.net/en/>.
Acesso em 02 dez. 2006.

TRAVIS, Greg. **IBM developerWorks - Getting started with new I/O (NIO),** 2003.
Disponível em <http://www.cs.brown.edu/courses/cs161/papers/j-nio-ltr.pdf>.
Acesso em 17 nov. 2006.

VEJA - Videogames competem com os filmes. Revista Veja. São Paulo: Abril, Edição 1830. 26/11/2003 – 03/12/2003

WEST, David. **Planning a Project with the Rational Unified Process**, 2002.
Disponível em <http://www.imsi-pm.com/home/library/Planning_an_IT_Project_with_RUP.pdf>.
Acesso em: 22 nov. 2006.

WIKIPEDIA_UML. **List of UML tools.**
Disponível em <http://en.wikipedia.org/wiki/List_of_UML_tools>.
Acesso em 02 dez. 2006.

WIRESHARK. Disponível em <http://www.wireshark.org/>. Acesso em 02 dez. 2006.

Apêndice A – Unified Process

1 *Introdução*

O Unified Process (UP) é um processo de engenharia de software cujo objetivo é possibilitar o desenvolvimento de sistemas de alta qualidade com custos e prazos controlados.

O UP descreve boas práticas de desenvolvimento e métodos disciplinados para a definição de tarefas e responsabilidades que são configuráveis, podendo ser aplicados em quaisquer projetos de software, independentemente do porte da empresa ou do projeto.

Além disso, possui três características-chave. Primeiro, o UP é guiado por casos de uso, ou seja, pelos requisitos funcionais definidos pelos clientes. Outra característica é o fato de ser centrado em arquitetura, existindo um foco na criação e manutenção de modelos ao invés de documentos. Isso é suportado pela Unified Modeling Language (UML). Por último, ele é um modelo iterativo e incremental.

Todo o processo é suportado por ferramentas, que automatizam muitas das atividades propostas, compostas principalmente do gerenciamento de artefatos, ou modelos.

2 *As 6 boas práticas*

Como citado anteriormente, o UP define algumas boas práticas do processo de desenvolvimento. Elas são consideradas boas não pelos ganhos obtidos com a sua incorporação, mas pelo fato de serem usadas pelas organizações bem sucedidas. Tais práticas serão enumeradas e descritas a seguir:

1. **Desenvolver softwares iterativamente** – Devido à complexidade dos sistemas desenvolvidos atualmente, não é possível definir inicialmente todo o problema e a sua solução. Ao contrário, é interessante definir diversos ciclos de desenvolvimento com determinados resultados esperados, que proporcionam um melhor entendimento e controle do projeto a cada passo, permitem definir pontos de interação com o cliente para saber sua opinião e possibilitam o foco nos itens de maior risco a cada etapa.
2. **Gerenciar requisitos** – O processo descreve meios de levantar, organizar e documentar requisitos. Esse controle é realizado com foco nos casos de uso do sistema em desenvolvimento, que descrevem de maneira precisa os requisitos funcionais, assegurando que o software final irá satisfazer as necessidades do usuário.
3. **Usar arquiteturas baseadas em componentes** – O UP descreve como desenvolver arquiteturas flexíveis, compreensíveis intuitivamente e que privilegiam o reaproveitamento de código. Para atingir tais resultados, o processo suporta o desenvolvimento de softwares baseados em componentes através da definição de um método sistemático para definir arquiteturas que utilizem tanto componentes já existentes quanto novos. Componentes são módulos responsáveis pela execução de uma determinada função.
4. **Modelar software visualmente** – Outro foco do processo é a modelagem visual utilizando a UML. Através dessa prática é possível descrever a estrutura do sistema sem ambigüidades, focando a sua funcionalidade e escondendo detalhes de implementação.
5. **Verificar a qualidade do software** – Atualmente, o atendimento de requisitos

não-funcionais como performance e confiabilidade são cruciais para a aceitação de um projeto de software. A determinação da qualidade do produto depende dos testes desse tipo de requisitos e o UP oferece meios de planejar, definir, implementar, executar e avaliar tais testes. Além disso, é importante reforçar que a qualidade depende principalmente do processo, de suas atividades e de seus participantes, e não de uma equipe separada focada na garantia de qualidade.

6. **Controlar mudanças no software** – Em projetos de software, as mudanças são inevitáveis. O processo descreve meios de controlar, rastrear e monitorar alterações, o que possibilita o desenvolvimento iterativo citado anteriormente.

3 Estrutura do processo

O UP é estruturado através da relação entre dois itens: as fases e os grupos de atividades do projeto. Essa estrutura pode ser ilustrada pelo gráfico abaixo que mostra a distribuição de esforço dedicado às atividades de acordo com a fase do projeto. O eixo horizontal mostra a dinâmica do processo que pode ser expressa através de fases e iterações. Já o eixo vertical representa o aspecto estático do processo.

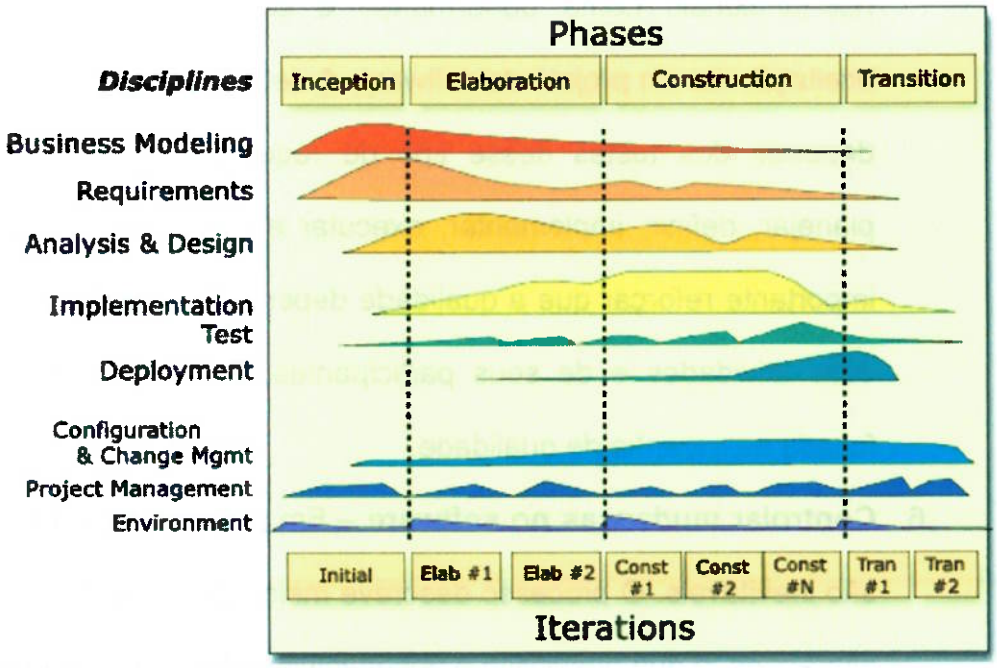

FIGURA 27 - Estrutura do processo

4 *Iterações*

As iterações são fundamentais para se ter sucesso na utilização do UP. Elas são as unidades nas quais uma fase pode ser dividida. Cada iteração é ciclo de desenvolvimento completo cujo resultado é a entrega de um sub-produto que fará parte do produto final.

5 *As fases de um projeto*

Sob o UP, cada novo projeto de desenvolvimento de um software é considerado um ciclo, que por sua vez é dividido em 4 fases: iniciação, elaboração, construção e transição. Ao fim de cada fase, existe um marco para a apresentação de resultados das atividades-chave da fase e para a tomada de decisões críticas.

5.1 Iniciação

O foco dessa fase é a definição do sistema sob o ponto de vista do negócio, deixando detalhes de implementação em segundo plano. Dentre outras coisas, é importante definir os objetivos do projeto, os riscos envolvidos, uma estimativa de custos e um planejamento preliminar com os principais marcos do projeto.

Além disso, é crucial delimitar o escopo do projeto, o que pode ser realizado através da identificação das entidades que irão interagir com o sistema, os atores, e das funcionalidades que o sistema deverá oferecer ao usuário. Essa identificação pode ser realizada através do diagrama de casos de uso definido pela UML, porém o objetivo dessa fase não é a de tentar identificar todos os casos existentes, podendo-se concentrar os esforços nos casos principais.

Ao final da fase de iniciação, os principais resultados esperados são:

- Documento de especificação preliminar descrevendo os requisitos mais importantes do sistema, suas funcionalidades e limitações.
- Modelo de casos de uso contendo os casos principais.
- Glossário inicial do projeto.
- Análise de negócio com pesquisas de mercado e estudo de viabilidade econômica.
- Análise de riscos de projeto.
- Planejamento inicial definindo fases e iterações.
- Um ou mais protótipos.

5.2 Elaboração

As principais atividades da fase de elaboração são a análise do domínio do problema, a determinação da arquitetura fundamental do sistema, o desenvolvimento do plano de projeto e a eliminação dos maiores riscos do projeto. A realização dessas atividades tem como objetivo assegurar que a arquitetura, requisitos e riscos estão estabilizados, para que se possa estimar com maior precisão o esforço, os custos e o tempo necessários para a finalização do desenvolvimento.

Uma das recomendações mais importantes para essa fase é a construção de um protótipo para se testar a arquitetura estabelecida. Esse protótipo deve englobar os casos de uso mais importantes que foram listados na fase de iniciação, pois provavelmente são os de maior risco.

Os produtos dessa fase são os seguintes:

- Determinação de todos os casos de uso e atores, faltando apenas a descrição de alguns casos.
- Identificação dos requisitos não-funcionais.
- Modelo da arquitetura de software.
- Protótipo executável da arquitetura do sistema.
- Revisão das análises de custo e risco.

5.3 Construção

Nessa fase, os componentes e funcionalidades são desenvolvidos e integrados ao produto. O desenvolvimento pode ser realizado de forma paralela para acelerar o processo, porém esse paralelismo pode aumentar a complexidade de gerenciamento do projeto.

Além disso, outro foco dessa etapa é o teste das funcionalidades desenvolvidas, ou seja, a verificação da correspondência do produto com a especificação criada nas fases anteriores.

Ao final da fase de construção, deve-se ter uma primeira versão do software já integrada a plataforma de produção. Caso necessário, pode-se criar uma descrição da versão e um manual do usuário.

5.4 Transição

A fase de transição envolve a instalação do produto no ambiente do usuário final. Esse é um momento delicado, com testes junto ao usuário final, quando é comum surgirem problemas que fazem com que seja necessária a criação de novas versões. É nesse momento também que se realiza o treinamento de usuários e administradores do sistema.

6 *Disciplinas de um projeto*

Como citado no item "Estrutura do processo", o projeto de desenvolvimento sob o UP é estruturado em fases e grupos de atividades. No item anterior, as fases do processo foram descritos, assim como as atividades que as compõe. No entanto, as atividades descritas podem ser agrupadas também por grupos de atividades, mais conhecidos como disciplinas.

Existem nove disciplinas, sendo que três são consideradas disciplinas de suporte e as seis seguintes, disciplinas principais:

- **Modelagem de negócio** – grupo de atividades que procuram assegurar o entendimento e documentação do processo de negócio que o sistema suportará.
- **Captura de requisitos** - definição das funcionalidades que o sistema irá oferecer através da identificação dos atores e casos de uso.
- **Análise e projeto** – disciplina para a determinação da arquitetura do software, cujo objetivo é especificar de maneira precisa os os componentes do sistema e a maneira como eles interagem entre si. A arquitetura determinada é uma abstração do código fonte que será gerado na fase de implementação.
- **Implementação** - conversão dos modelos em código. Nas atividades dessa disciplina, é importante assegurar que o código seja estruturado de maneira que o produto final seja fácil de manter e ser reutilizado. Uma das maneiras de se conseguir isso é através do processo de componentização descrito no UP.
- **Teste** - verificação do correto funcionamento dos componentes desenvolvidos

e da correta integração deles para o atendimento aos requisitos determinados, além da detecção de defeitos.

- **Distribuição** – grupo de atividades que visam realizar a instalação do software desenvolvido no ambiente do cliente de maneira rápida e confiável. Além disso, deve-se assegurar que os clientes tenham condições de usar o sistema através de seu treinamento.

Apêndice B – Cronogramas

O acompanhamento do projeto foi realizado através de diversas formas. O principal recurso utilizado foram os cronogramas. Foram gerados três deles (cronogramas A, B e C) durante o projeto com a utilização da ferramenta livre de gerenciamento de projetos OpenWorkbench (http://www.openworkbench.org/). O primeiro logo no início do projeto em meados de Janeiro. O segundo no fim de Março quando houve a definição da metodologia a ser utilizada no projeto. E por fim, a última no início de Setembro, no período de volta às aulas.

Além dos cronogramas, o acompanhamento se deu através de e-mails para divulgação de atas de reunião. Para planejar atividades e divulgar suas conclusões foram utilizados tanto o e-mail quanto o wiki Jotspot.

Por essa razão, os cronogramas apresentados nesse item não são exatamente os confeccionados durante o projeto, pois foram redefinidos com a inclusão de informações de planejamento contidos em e-mails e no wiki. Além disso, estes cronogramas não foram feitos com o OpenWorkbench, mas refeitos com a utilização de outra ferramenta livre de mesmo gênero chamada Planner (http://winplanner.sourceforge.net/) devido a um ponto fraco do OpenWorkbench que é a escassez de alternativas de visualização, o que dificulta a inserção dos cronogramas em documentos.

Além dos cronogramas, foi gerado um quarto arquivo contendo as atividades efetivamente realizadas pela equipe ao longo do projeto.

Cronograma A

WBS	Name	Start	Finish	Duration
1	Monografia - estrutura definitiva	Mar 29	Apr 18	15d
2	Estudo da enJine	Apr 19	May 2	10d
3	**Arquitetura**	**May 3**	**Jul 13**	**52d**
3.1	Estudo de design patterns	May 3	May 9	5d
3.2	**Protótipo baseado na arquitetura proposta**	**May 10**	**Jun 29**	**37d**
3.2.1	Estudo da tecnologia seleconada (NIO)	May 10	May 16	5d
3.2.2	Desenvolvimento de aplicativo cliente-servidor para teste do NIO	May 17	May 30	10d
3.2.3	Desenvolvimento do protótipo	May 31	Jun 29	22d
3.3	**Documentação do protótipo desenvolvido**	**Jun 30**	**Jul 13**	**10d**
3.3.1	Modelos UML	Jun 30	Jul 6	5d
3.3.2	Processo/metodologia de desenvolvimento	Jun 30	Jul 13	10d
4	**Especificação**	**Jul 14**	**Aug 17**	**25d**
4.1	Domínio	Jul 14	Jul 20	5d
4.2	Requisitos funcionais	Jul 21	Aug 3	10d
4.3	Requisitos não funcionais	Jul 21	Jul 27	5d
4.4	Casos de uso	Aug 4	Aug 17	10d
5	**Análise**	**Aug 18**	**Aug 31**	**10d**
5.1	Diagrama de classes	Aug 18	Aug 24	5d
5.2	Diagrama de sequência dos casos de uso principais	Aug 18	Aug 31	10d
6	**Implementação**	**Sep 1**	**Oct 19**	**35d**
6.1	Desenvolvimento do módulo	Sep 1	Oct 2	22d
6.2	Integração ao enJine	Sep 20	Oct 19	22d
7	**Produto (jogo)**	**Oct 20**	**Nov 20**	**22d**
7.1	Desenvolvimento	Oct 20	Nov 20	22d
7.2	Confecção do manual do usuário	Oct 20	Nov 20	22d
7.3	Documentação	Nov 6	Nov 20	11d
8	**Testes**	**Oct 20**	**Nov 20**	**22d**
8.1	Requisitos funcionais	Oct 20	Nov 2	10d
8.2	**Requisitos não funcionais**	**Oct 20**	**Nov 20**	**22d**
8.2.1	Pesquisa de ferramentas	Oct 20	Oct 30	7d
8.2.2	Definição de domínio e metodologia	Oct 31	Nov 6	5d
8.2.3	Execução dos testes	Nov 7	Nov 20	10d
9	Monografia	Mar 29	Nov 30	176d 7h

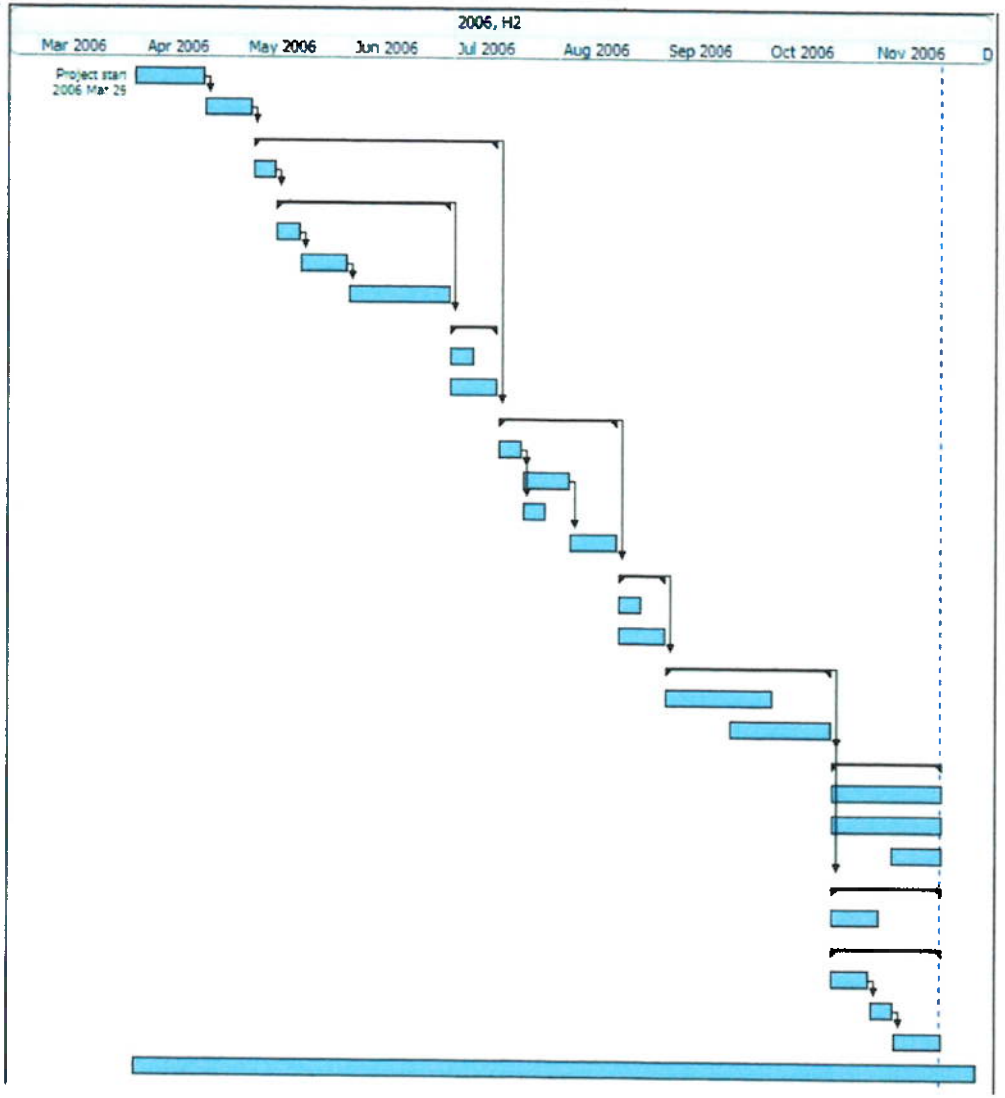

FIGURA 28 - Cronograma criado em meados de Janeiro

Cronograma B

WBS	Name	Start	Finish	Duration
1	Monografia - estrutura definitiva	Mar 29	Apr 18	15d
2	Estudo da enjine	Apr 19	May 2	10d
3	**Arquitetura**	**May 3**	**Jul 13**	**52d**
3.1	Estudo de design patterns	May 3	May 9	5d
3.2	**Protótipo baseado na arquitetura proposta**	**May 10**	**Jun 29**	**37d**
3.2.1	Estudo da tecnologia seleconada (NIO)	May 10	May 16	5d
3.2.2	Desenvolvimento de aplicativo cliente-servidor para teste do NIO	May 17	May 30	10d
3.2.3	Desenvolvimento do protótipo	May 31	Jun 29	22d
3.3	**Documentação do protótipo desenvolvido**	**Jun 30**	**Jul 13**	**10d**
3.3.1	Modelos UML	Jun 30	Jul 6	5d
3.3.2	Processo/metodologia de desenvolvimento	Jun 30	Jul 13	10d
4	**Especificação**	**Jul 14**	**Aug 17**	**25d**
4.1	Domínio	Jul 14	Jul 20	5d
4.2	Requisitos funcionais	Jul 21	Aug 3	10d
4.3	Requisitos não funcionais	Jul 21	Jul 27	5d
4.4	Casos de uso	Aug 4	Aug 17	10d
5	**Análise**	**Aug 18**	**Aug 31**	**10d**
5.1	Diagrama de classes	Aug 18	Aug 24	5d
5.2	Diagrama de sequência dos casos de uso principais	Aug 18	Aug 31	10d
6	**Implementação**	**Sep 1**	**Oct 19**	**35d**
6.1	Desenvolvimento do módulo	Sep 1	Oct 2	22d
6.2	Integração ao enjine	Sep 20	Oct 19	22d
7	**Produto (jogo)**	**Oct 20**	**Nov 20**	**22d**
7.1	Desenvolvimento	Oct 20	Nov 20	22d
7.2	Confecção do manual do usuário	Oct 20	Nov 20	22d
7.3	Documentação	Nov 6	Nov 20	11d
8	**Testes**	**Oct 20**	**Nov 20**	**22d**
8.1	Requisitos funcionais	Oct 20	Nov 2	10d
8.2	**Requisitos não funcionais**	**Oct 20**	**Nov 20**	**22d**
8.2.1	Pesquisa de ferramentas	Oct 20	Oct 30	7d
8.2.2	Definição de domínio e metodologia	Oct 31	Nov 6	5d
8.2.3	Execução dos testes	Nov 7	Nov 20	10d
9	Monografia	Mar 29	Nov 30	176d 7h

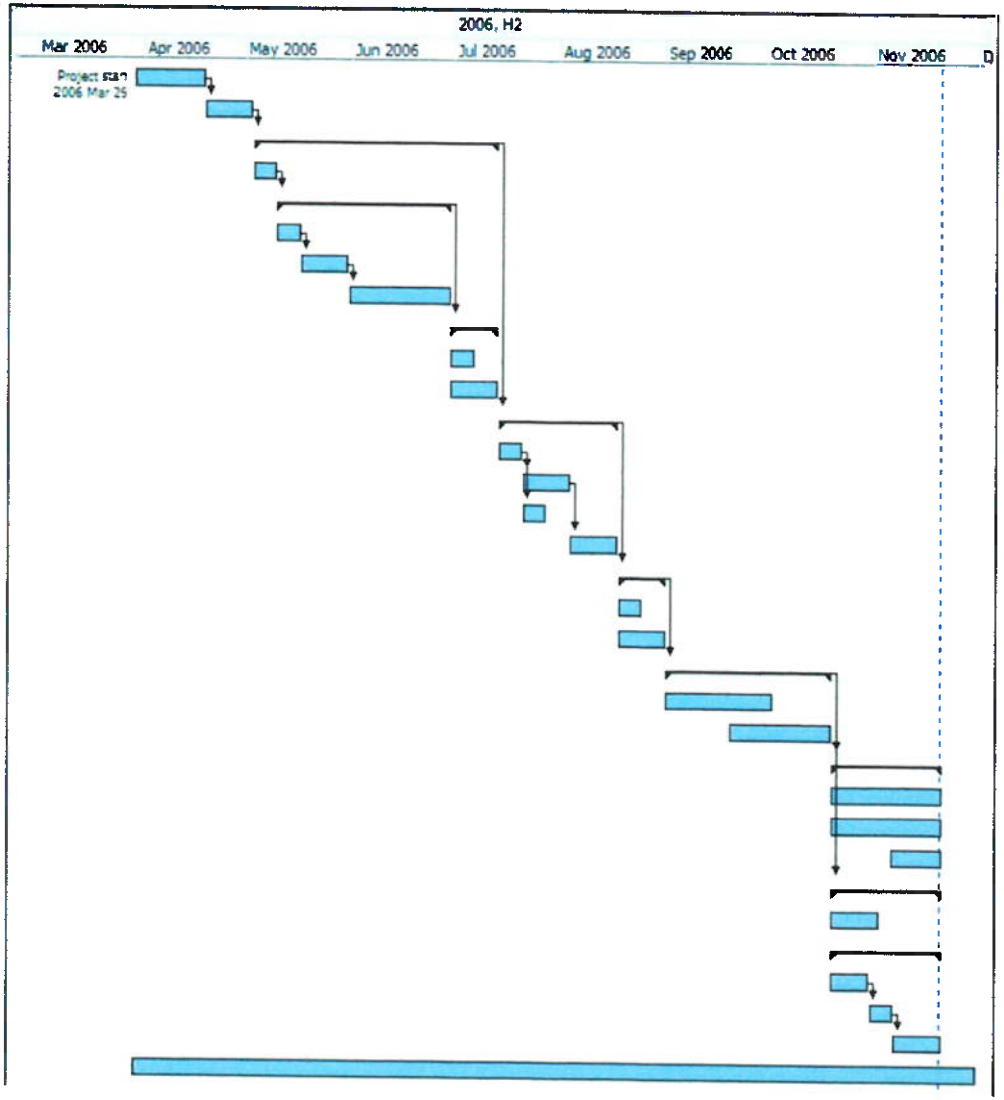

FIGURA 29 - Cronograma gerado após a escolha da metodologia de desenvolvimento

Cronograma C

WBS	Name	Start	Finish	Duration
1	**Especificação**	**Aug 23**	**Aug 29**	**5d**
1.1	Casos de uso	Aug 23	Aug 29	5d
2	**Arquitetura**	**Aug 30**	**Sep 12**	**10d**
2.1	Diagrama de classes	Aug 30	Sep 5	5d
2.2	Diagrama de sequência dos casos de uso principais	Aug 30	Sep 12	10d
3	**Documentação**	**Sep 13**	**Sep 26**	**10d**
3.1	Resultados do desenvolvimento do protótipo	Sep 13	Sep 14	2d
3.2	Razão de não haver conexão entre listener e connection	Sep 15	Sep 15	1d
3.3	Funcionamento das mensagens utilizadas pelo módulo	Sep 18	Sep 20	3d
3.4	Pré-processamento lógico e visual	Sep 21	Sep 22	2d
3.5	Conexão assincrona	Sep 25	Sep 25	1d
3.6	Mensagens entre objetos no servidor	Sep 26	Sep 26	1d
4	**Implementação**	**Sep 27**	**Oct 19**	**17d**
4.1	Desenvolvimento do módulo	Sep 27	Oct 19	17d
4.2	Integração ao enJine	Sep 27	Oct 19	17d
5	**Produto - Lego Adventures Online**	**Oct 20**	**Nov 27**	**27d**
5.1	Desenvolvimento	Oct 20	Nov 20	22d
5.2	Melhorias na parte gráfica	Nov 21	Nov 27	5d
5.3	Confecção do manual do usuário do módulo	Oct 20	Nov 20	22d
5.4	Documentação	Nov 6	Nov 20	11d
6	**Testes**	**Oct 20**	**Nov 20**	**22d**
6.1	Requisitos funcionais	Oct 20	Nov 2	10d
6.2	**Requisitos não funcionais**	**Oct 20**	**Nov 20**	**22d**
6.2.1	Pesquisa de ferramentas	Oct 20	Oct 30	7d
6.2.2	Definição de domínio e metodologia	Oct 31	Nov 6	5d
6.2.3	Execução dos testes	Nov 7	Nov 20	10d
7	Limpeza e organização de código	Nov 21	Dec 4	10d
8	Monografia	Aug 23	Nov 30	72d

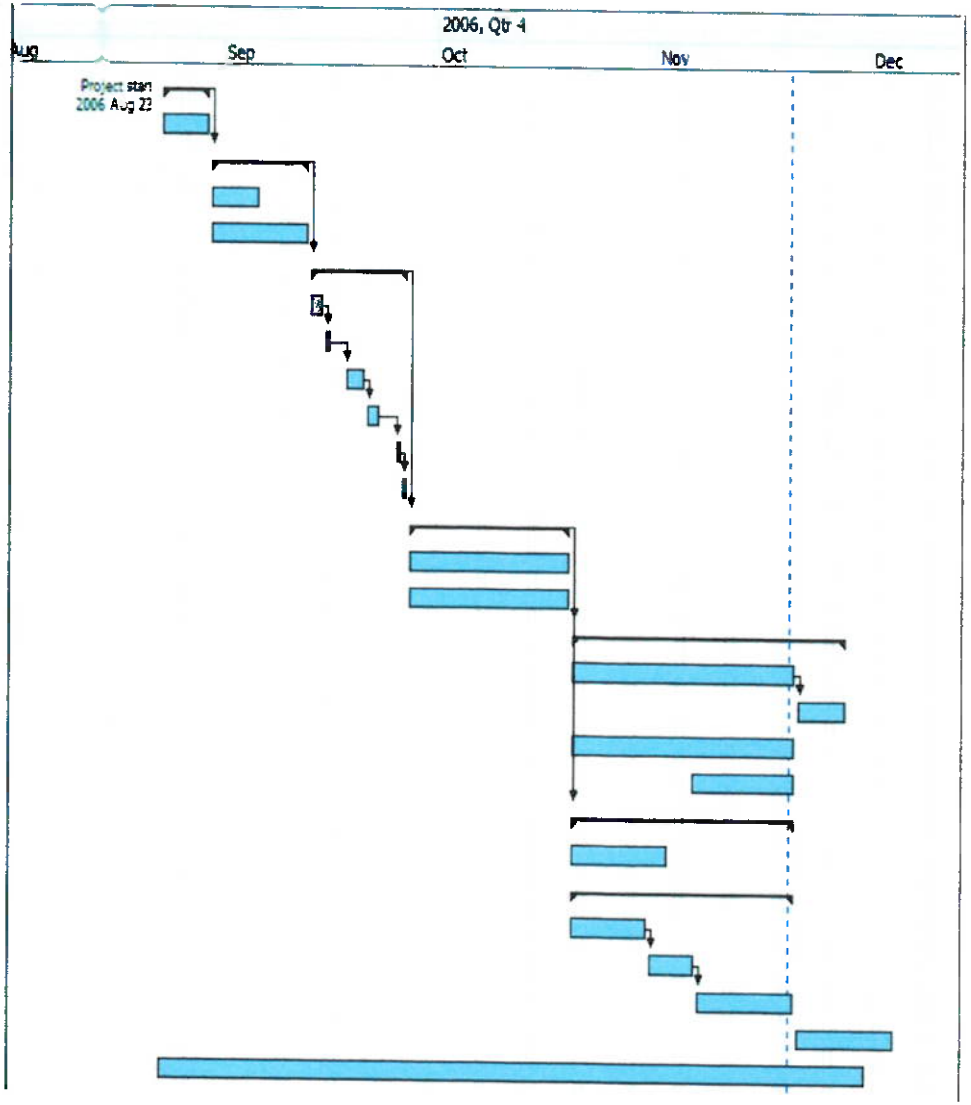

FIGURA 30 - Cronograma confeccionado no período de volta às aulas

Atividades efetivamente realizadas 1 de 2

WBS	Name	Start	Finish	Duration
1	**Proposta**	**Jan 19**	**Feb 2**	**11d**
1.1	Pesquisa - mercado de jogos	Jan 19	Jan 23	3d
1.2	Pesquisa - redes em jogos	Jan 19	Jan 25	5d
1.3	Pesquisa - engines de jogos	Jan 19	Jan 23	3d
1.4	Pesquisa - UP/XP	Jan 19	Jan 27	7d
1.5	Confecção de documento	Jan 30	Feb 2	4d
2	**Especificação preliminar**	**Jan 19**	**Mar 6**	**33d**
2.1	**Avaliação de arquiteturas de rede**	**Jan 19**	**Mar 6**	**33d**
2.1.1	Aquisição de artgos	Jan 19	Jan 25	5d
2.1.2	Leitura dos artgos	Jan 19	Feb 1	10d
2.1.3	Redação de resumos dos artgos	Jan 19	Feb 8	15d
2.1.4	Análise das alternativas	Feb 9	Feb 22	10d
2.1.5	Documentação	Feb 23	Mar 6	8d
2.2	Requisitos funcionais	Feb 9	Mar 6	18d
3	**Arquitetura**	**Mar 7**	**Mar 24**	**14d**
3.1	Refinamento dos requisitos funcionais	Mar 7	Mar 9	3d
3.2	Especificação dos módulos do sistema	Mar 10	Mar 21	8d
3.3	Especificação da integração com o enJine	Mar 10	Mar 21	8d
3.4	Documentação	Mar 22	Mar 24	3d
4	**Pesquisa aprofundada de metodologias UP/XP**	**Mar 7**	**Mar 30**	**18d**
4.1	Aquisição de materiais	Mar 7	Mar 13	5d
4.2	Leitura	Mar 7	Mar 20	10d
4.3	Análise das alternativas	Mar 21	Mar 27	5d
4.4	Documentação	Mar 28	Mar 30	3d
5	**Pesquisa de ferramentas gratuitas**	**Mar 6**	**Mar 10**	**5d**
5.1	Modelagem UML	Mar 6	Mar 10	5d
5.2	Gerenciamento de projetos	Mar 6	Mar 10	5d
6	Versão inicial da monografia	Mar 31	Apr 14	11d
7	Estudo da enJine	Mar 31	May 1	22d
8	Estudo de design patterns	May 2	May 15	10d
9	**Protótipo**	**May 2**	**Aug 22**	**81d**

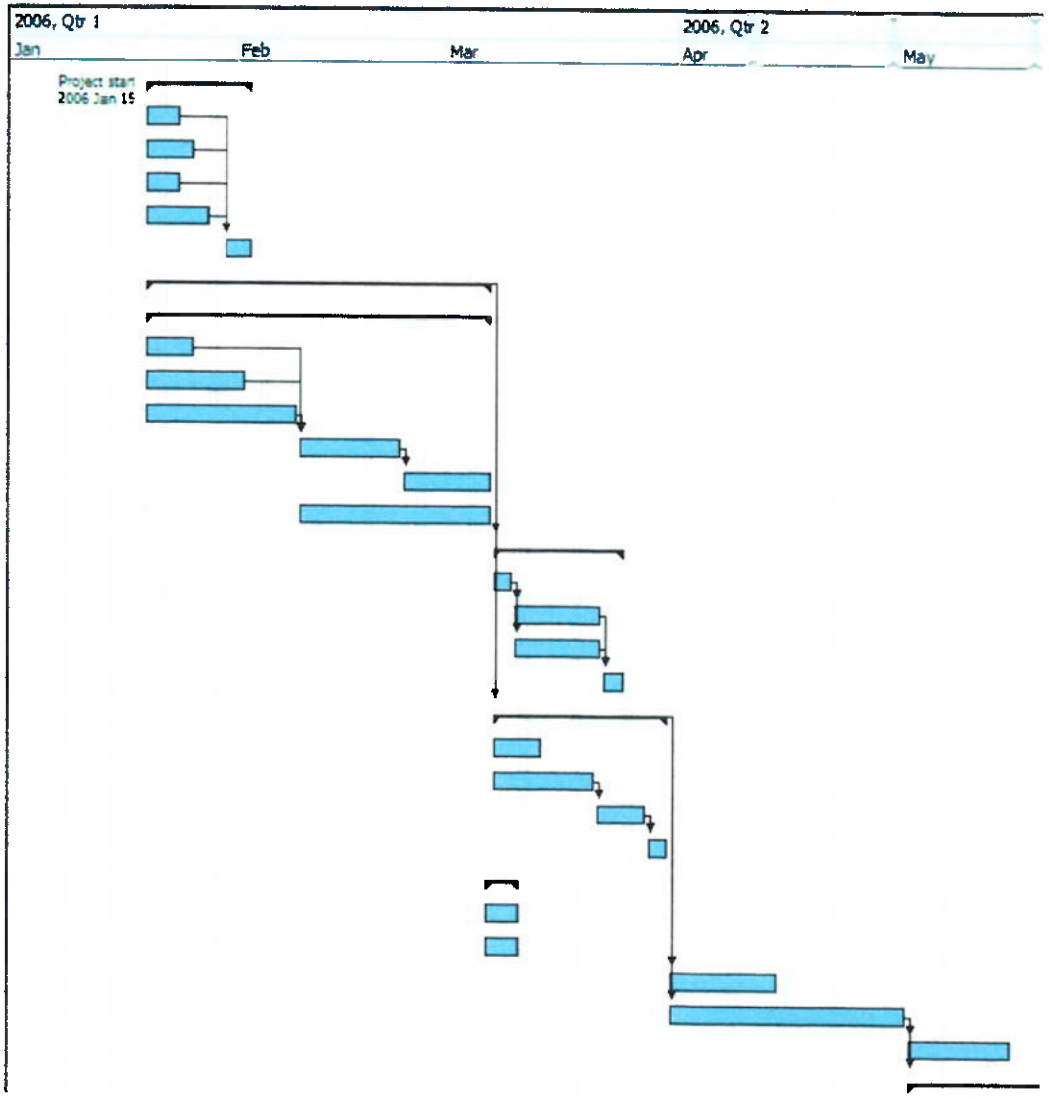

FIGURA 31 - Primeira parte das atividades efetivamente realizadas

Atividades efetivamente realizadas 2 de 2

WBS	Name	Start	Finish	Duration
9	**Protótipo**	**May 2**	**Aug 22**	**81d**
9.1	Estudo do NIO	May 2	May 8	5d
9.2	Aplicativo cliente-servidor de teste	May 9	May 22	10d
9.3	Preparação de ambiente de desenvolvimento	May 23	May 25	3d
9.4	Desenvolvimento do protótipo	May 26	Jul 26	44d
9.5	**Documentação**	**Jul 27**	**Aug 16**	**15d**
9.5.1	Diagrama de classes	Jul 27	Jul 31	3d
9.5.2	Funcionamento do protótipo	Aug 1	Aug 16	12d
9.5.3	Processo/metodologia de desenvolvimento	Jul 27	Aug 9	10d
9.6	Integração da prototipação à monografia	Aug 17	Aug 22	4d
10	**Revisão da monografia**	**Aug 23**	**Aug 29**	**5d**
10.1	Erros	Aug 23	Aug 29	5d
10.2	Referências	Aug 23	Aug 29	5d
10.3	Estilo	Aug 23	Aug 29	5d
11	**Detalhamento de alguns itens da monografia**	**Aug 23**	**Sep 20**	**21d**
11.1	Resultados do desenvolvimento do protótipo	Aug 23	Aug 29	5d
11.2	Razão de não haver conexão entre listener e conn	Aug 30	Aug 31	2d
11.3	Funcionamento do mecanismo de mensagens	Sep 1	Sep 7	5d
11.4	Pré-processamento lógico e visual	Sep 8	Sep 14	5d
11.5	Diferenças de mensagens síncronas/assíncronas	Sep 15	Sep 15	1d
11.6	Mensagens entre objetos no servidor	Sep 18	Sep 20	3d
12	**Implementação**	**Sep 21**	**Oct 20**	**22d**
12.1	Desenvolvimento do módulo	Sep 21	Oct 11	15d
12.2	Integração ao enJine	Sep 21	Oct 20	22d
13	Testes de requisitos funcionais	Oct 23	Oct 27	5d
14	Limpeza e organização do código	Oct 23	Oct 27	5d
15	**Produto - Lego Adventures Online**	**Oct 23**	**Nov 22**	**23d**
15.1	Especificação	Oct 23	Oct 25	3d
15.2	Desenvolvimento	Oct 26	Nov 15	15d
15.3	Confecção do manual do usuário do módulo	Oct 26	Nov 15	15d
15.4	Melhorias na parte gráfica	Nov 16	Nov 22	5d
16	**Testes de requisitos não-funcionais**	**Nov 13**	**Nov 24**	**10d**
16.1	Definição de domínio e metodologia	Nov 13	Nov 17	5d
16.2	Execução	Nov 20	Nov 24	5d
17	**Monografia**	**Nov 6**	**Nov 17**	**10d**
17.1	Relatório - UP aplicado ao projeto	Nov 6	Nov 10	5d
17.2	Acertos em Mercado de Jogos e MMGs	Nov 13	Nov 17	5d
17.3	Acertos NIO e pré-processamento	Nov 13	Nov 17	5d

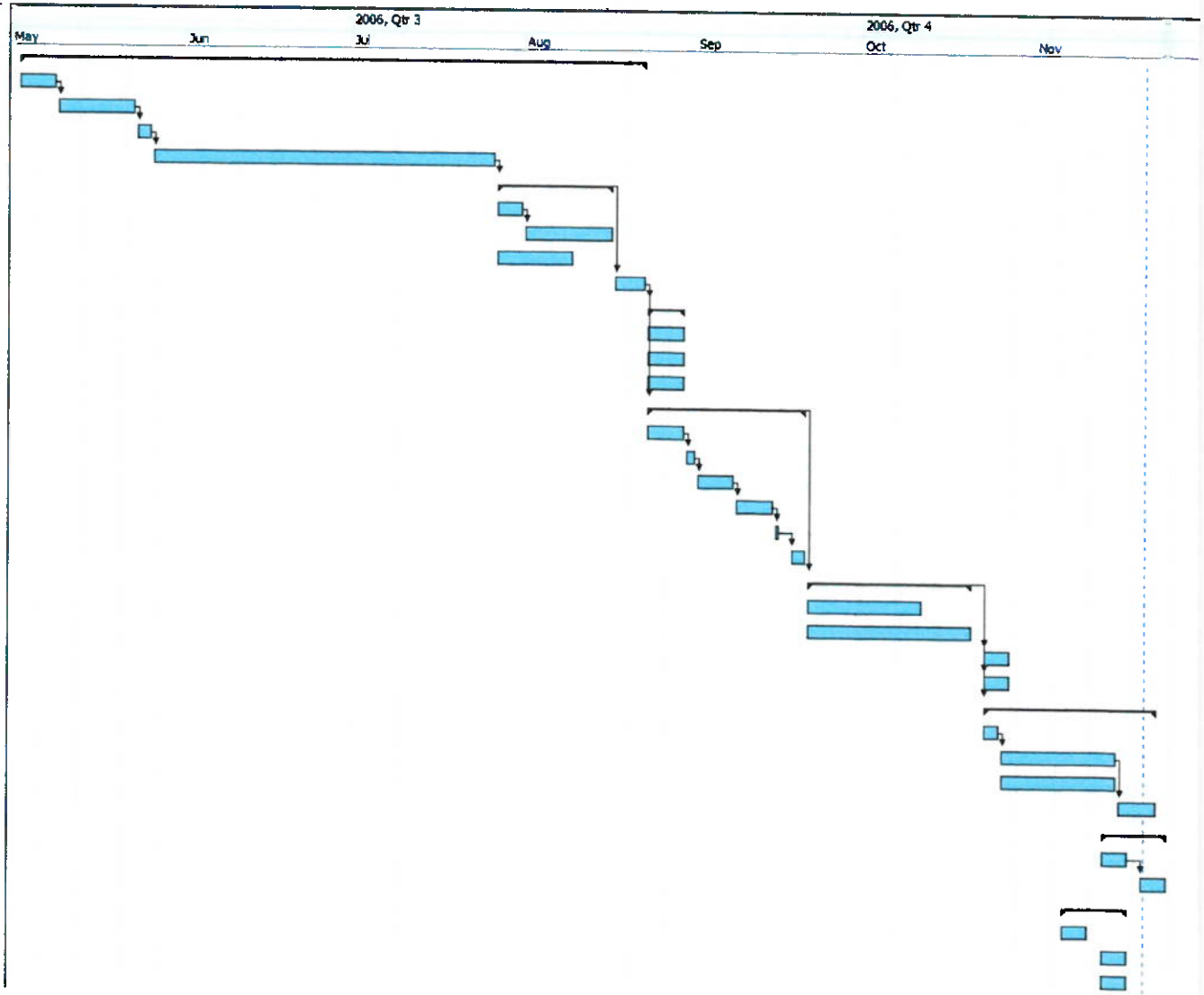

FIGURA 32 - Complemento das atividades efetivamente realizadas

Apêndice C – Lego Adventures Online

1 Introdução

O aplicativo "Lego Adventures Online" foi desenvolvido a partir do jogo "Lego Adventures", anteriormente criado pela equipe formada por Rodrigo Carvalhedo Petriche e Tatiana Fiorillo Hwa na disciplina de Computação Gráfica para aplicação de diversos conceitos da disciplina. Este último trata-se de um jogo eletrônico desenvolvido em Java/Java3D, utilizando-se também de ferramentas desenvolvidas no Laboratório de Tecnologias Interativas (Interlab) como o enJine e o Interlab3D, além do programa Art of Illusion (AoI), disponivel gratuitamente na Internet. Ele foi adaptado a este projeto para utilizar o novo módulo de redes desenvolvido, passando a se chamar "Lego Adventures Online".

Este documento visa apresentar seu game design, descrevendo as características do jogo, sua especificação e as alterações sofridas durante o processo de desenvolvimento e também na adaptação para o projeto online.

2 Game Design

Nos itens seguintes, serão apresentadas as propostas iniciais, as alterações durante o projeto e adaptações para versão online de inúmeros componentes do jogo "Lego Adventures".

2.1 Escopo

2.1.1 Proposta inicial

O jogo será do tipo "Caça ao Tesouro". Cada fase corresponde à procura do personagem por um tesouro. O tesouro é encontrado através das pistas.

As pistas estarão espalhadas pelo cenário e cabe ao personagem encontrá-las no tempo determinado, ir até as mesmas e ler seu conteúdo, visando a achar a próxima pista.

2.1.2 Alterações e versão final

Deciciu-se mudar um pouco o escopo do jogo tirando o conceito de fase e criando um jogo estilo adventure como geralmente são conhecidos jogos desse tipo como *Indiana Jones and the Fate of Atlantis* ou *Full Throtle* da *Lucas Arts*. Nesse tipo de jogo não existem fases, mas uma história única que é desenvolvida ao longo do jogo. Para isso o personagem interage como o cenário através de ações pré-determinadas como olhar, pegar, etc.

No jogo o personagem possui uma única ação ativada ao apertar-se o botão de ação em contato com um objeto com o qual é possível interagir. Não foi desenvolvida a história do jogo, mas sim uma interface simples que pode ser usada para o desenvolvimento de um jogo desse tipo, com algumas ações em um cenário.

2.1.3 Adaptação – Lego Adventures Online

A introdução de diversos jogadores ao "Lego Adventures" causou uma pequena alteração de escopo do jogo. Nessa nova versão jogo passou a ser competitivo e o objetivo é vencer seus adversários coletando mais itens que eles. Esses itens aperecem aleatoriamente no jogo cada vez que o item anterior é coletado por algum jogador.

2.2 Personagem

2.2.1 Proposta inicial

O personagem será baseado nos bonecos do brinquedo Lego, mas estará representando um aluno da Escola Politécnica.

As habilidades do personagem consistem em andar (setas do teclado), correr (Shift + setas do teclado) e pegar as pistas encontradas apenas passando por cima delas.

O personagem estará divido em seis diferentes objetos para que seja realizada sua animação: cabeça; braço esquerdo; braço direito; tronco; perna direita; perna esquerda; As mãos estarão integradas aos braços.

2.2.2 Alterações e versão final

O personagem não sofreu muitas alterações. Ele continua sendo um boneco de Lego dividido em seis partes. Foi feita apenas uma modificação, excluindo-se a habilidade do personagem pular e transformando-se a habilidade de pegar pista em executar ação (realizada com a barra de espaço quando o personagem está em contato com um objeto).

2.2.3 Adaptação – Lego Adventures Online

Na versão online o personagem não sofreu alterações visuais, porém suas funcionalidades foram reduzidas. Ele pode apenas andar na direção para qual está virado e girar para esquerda ou para direita.

2.3 Cenário

2.3.1 Proposta inicial

O cenário será baseado no prédio de Engenharia Elétrica da Escola Politécnica. Esse cenário será feito em módulos permitindo a expansão futura. O módulo inicial é constituido do pátio da entrada do prédio e será o cenário base do jogo.

2.3.2 Alterações e versão final

O cenário manteve-se inalterado e ficou realmente muito semelhante com o pátio de entrada do prédio da elétrica, conforme previsto.

2.3.3 Adaptação – Lego Adventures Online

Nessa versão o cenário foi removido para reduzir a quantidade de objetos a serem renderizados na tela e garantir que esse aspecto não influenciaria na análise das funcionalidades do módulo de redes.

2.4 Visão/Câmera

2.4.1 Proposta inicial

O jogo será em terceira pessoa e o jogador poderá rotacionar as câmeras 360° ao redor do personagem, além de possibilitar que ele olhe para cima e para baixo.

Esses movimentos podem ser automáticos (acompanhando o movimento do personagem) ou manuais (Ctrl + setas do teclado).

2.4.2 Alterações e versão final

A câmera foi limitada a uma única que acompanha o personagem durante seu movimento mas não é rotacionada, mantedo a visão en terceira pessoa.

2.4.3 Adaptação – Lego Adventures Online

Para esta versão não ocorreram novas modificações na câmera. Vale ressaltar apenas que a câmera acompanha somente o personagem do jogador e não os adversários, uma vez que agora existem diversos personagens na tela ao mesmo tempo.

3 Desenvolvimento

3.1 Ferramentas

O desenvolvimento do jogo foi auxiliado por diversas ferramentas. Esse item visa apresentar duas dessas principais ferramentas. O primeiro é o Interlab3D utilizado

principalmente para o aprendizado da API Java3D. O segundo é o Art of Illusion (AoI) utilizado para a modelatem de personagens e cenários.

3.1.1 Interlab3D

O Interlab3D é uma ferramenta aberta (licenciada sob a GPL) de apoio ao aprendizado de computação gráfica desenvolvido em Java pelo Interlab. Através dele é possível explorar aspectos do desenvolvimento de aplicações de alto nível de Computação Gráfica através de uma interface visual.

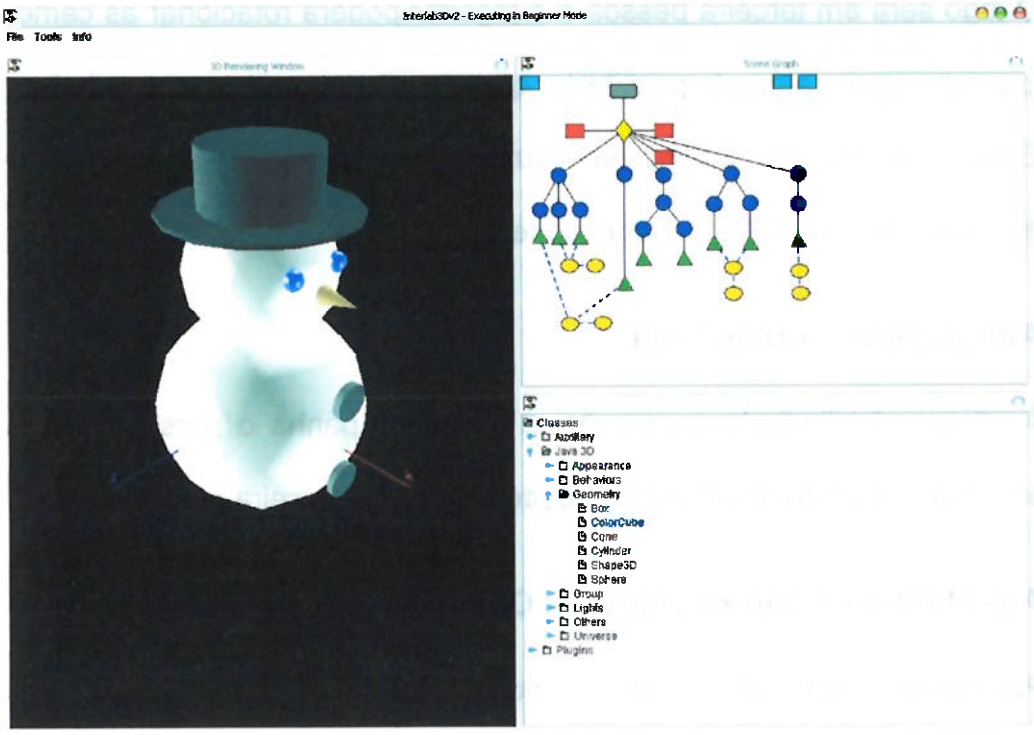

FIGURA 33 - Interface do Interlab3D

Um dos focos do programa é facilitar o aprendizado do conceito de "Grafo de Cena", utilizado pela API Java3D para a construção de cenas tridimensionais. Além disso, através do Interlab3D, é possível se familiarizar com o processo de desenvolvimento utilizando Java3D sem a necessidade de codificação. Com isso, consegue-se obter uma boa base conceitual e operacional para o posterior uso da enJine. Outra funcionalidade do programa explorada foi a geração de código Java a partir da

cena construída visualmente. O personagem foi incluído no código do jogo através desse procedimento. Ele foi montado visualmente através da importação e posicionamento dos modelos 3D das pertes que o compõe.

Para acelerar o processo de aprendizado de suas funcionalidades, diversos tutoriais são disponibilizados na página oficial do projeto na Internet.

3.1.2 Art of Illusion (AoI)

O Art of Illusion é um programa aberto (GPL) escrito em Java que permite a criação de modelos tridimensionais. O cenário do "Lego Adventures" foi construído através desse programa. Manual de usuário e tutoriais podem ser encontrados na página oficial do aplicativo.

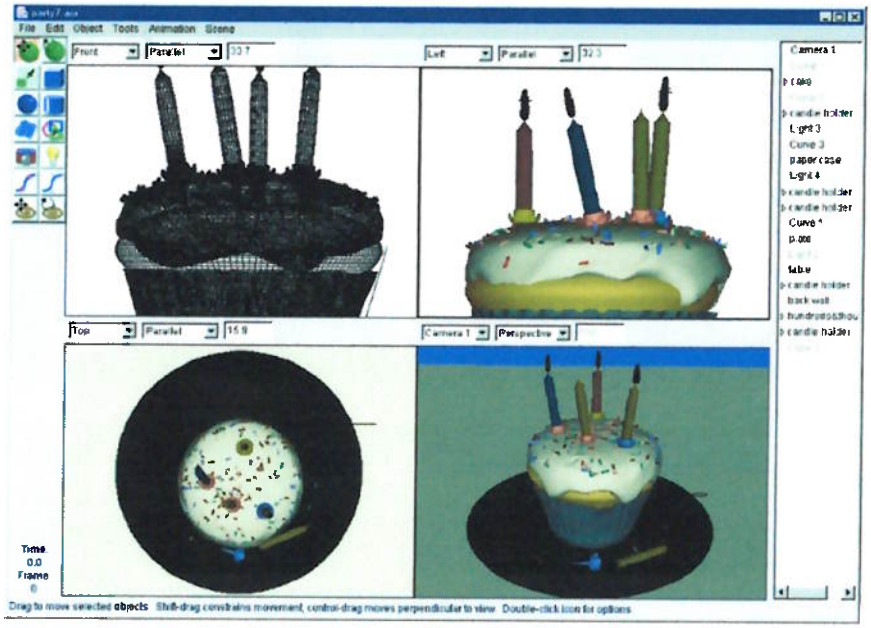

FIGURA 34 - Interface do Art of Illusion

Apêndice D – Casos de uso do módulo de redes

1. Objetivo

Este anexo apresenta os padrões utilizados na descrição dos casos de uso, o diagrama de casos de uso e a descrição dos casos de uso relacionados ao planejamento do módulo de redes do enJine.

2. Padrão utilizado para a especificação dos Casos de Uso

Segundo as diretrizes do processo unificado, utilizou-se a seguinte padronização para a descrição dos casos de uso:

- Identificador: identificador numérico do caso de uso.
- Nome: identificador único para o caso de uso que descreve a atividade realizada.
- Descrição: uma breve descrição do Caso de Uso, identificando seu objetivo, o ator que inicia o caso e quando é o seu término.
- Evento iniciador: nome do evento que inicia a execução do caso de uso.
- Atores: identificação dos atores participantes do caso de uso.
- Pré-condição: condições que devem ser verdadeiras para iniciar o caso de uso.
- Seqüência de eventos: descrição dos passos principais da seqüência do cenário principal do Caso de Uso.
- Pós-Condição: condições que devem ser verdadeiras após a execução do Caso de Uso.
- Extensões: caminhos alternativos do caso de uso, incluindo as exceções de um curso normal de eventos;
- Inclusões: lista de Casos de Uso usados pelo Caso de Uso corrente.

3. Casos de uso do módulo de redes do engine

3.1. Diagrama de casos de uso

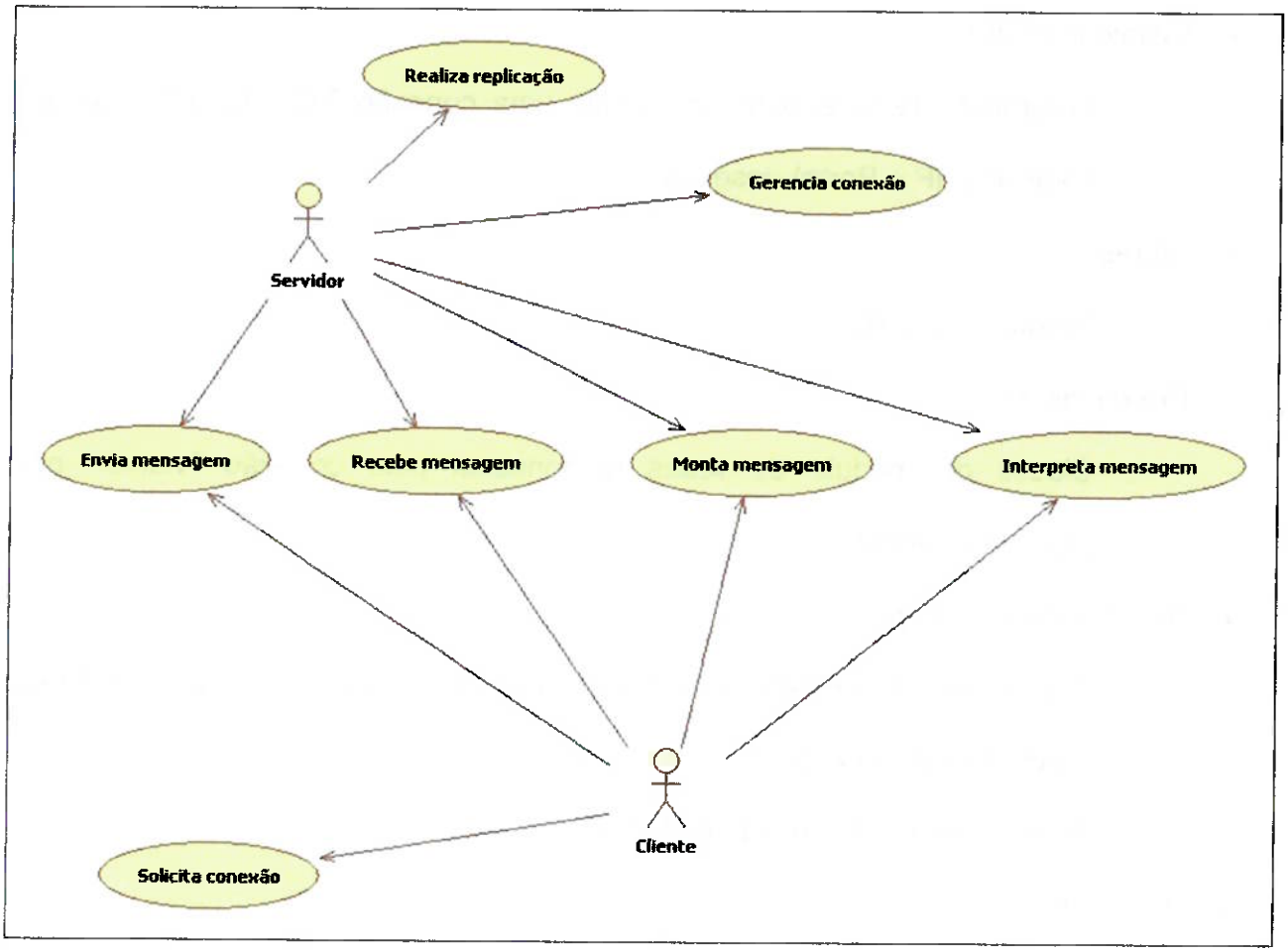

FIGURA 35 - Diagrama de casos de uso do módulo de redes.

3.2. Descrição dos casos de uso

Segue abaixo a descrição dos casos de uso do módulo de acordo com os padrões previamente apresentados. Por se tratar de um middleware, os casos de uso apresentam os serviços fornecidos por cada componente (ator) do módulo de redes em questão.

3.2.1 Solicita conexão

- Descrição
  - Este caso de uso descreve o processo de solicitação de conexão ao servidor.
- Evento iniciador
  - Programa cliente solicita ao módulo uma conexão TCP ou UDP com um endereço (IP + Porta) desejado.
- Atores
  - Programa cliente
- Pré-condição
  - Classe do módulo de redes responsável pela conexão "visível" pelo programa cliente.
- Sequência de eventos
  - Cliente solicita conexão ao servidor através de uma determinada porta em modo não bloqueante.
  - Servidor aceita a conexão do cliente.
- Pós-Condição
  - Cliente conectado a um servidor (estabelecimento de conexão entre um cliente e um servidor).
- Extensões
  - Servidor offline ou destino inexistente (passo 1): Geração de uma exceção.
- Inclusões
  - Aceita conexão (Passo 2)

3.2.2 Envia Mensagem (Cliente)

- Descrição

- Este caso de uso descreve o processo de envio de mensagem de um cliente para um servidor ou vice-versa.

- Evento iniciador
  - Mensagem encontrada na lista de mensagens a enviar.
- Atores
  - Cliente
- Pré-condição
  - Conexão estabelecida entre um cliente e um servidor.
  - Mensagem montada.
- Sequência de eventos
  - Cliente envia mensagem.
- Pós-Condição
  - Mensagem enviada.
  - Mensagem retirada da lista de mensagens a enviar.
- Extensões
  - Não há
- Inclusões
  - Não há

3.2.3 Envia Mensagem (Servidor)

- Descrição
  - Este caso de uso descreve o processo de envio de mensagem de um cliente para um servidor ou vice-versa.
- Evento iniciador
  - Mensagem encontrada na lista de mensagens a enviar.
- Atores

- Servidor

- Pré-condição
  - Conexão estabelecida entre um cliente e um servidor.
  - Mensagem montada.
- Sequência de eventos
  - Servidor verifica o destinatário e envia a mensagem.
- Pós-Condição
  - Mensagem enviada.
  - Mensagem retirada da lista de mensagens a enviar.
- Extensões
  - mensagem de broadcast (passo 1): envia a mensagem para todos os integrantes da área de interesse.
- Inclusões
  - Não há

3.2.4 Recebe Mensagem

- Descrição
  - Este caso de uso descreve o processo de recebimento de uma mensagem pelo cliente e servidor.
- Evento iniciador
  - Módulo detecta presença de mensagem recebida no canal de comunicação.
- Atores
  - Cliente, Servidor
- Pré-condição
  - Conexão estabelecida entre um cliente e um servidor.
- Sequência de eventos

- Módulo insere mensagem na lista de mensagens recebidas.

- Pós-Condição
  - Mensagem inserida na lista de mensagens recebidas.
- Extensões
  - Não há
- Inclusões
  - Não há

3.2.5 Monta Mensagem (Cliente)

- Descrição
  - Este caso de uso descreve o processo de montagem de mensagem no cliente e no servidor.
- Evento iniciador
  - Usuário executa uma ação no jogo.
- Atores
  - Cliente
- Pré-condição
  - Não há
- Sequência de eventos
  - Os campos da mensagem são preenchidos de acordo com a ação executada pelo usuário. (de acordo com as regras de montagem de mensagem)
- Pós-Condição
  - Mensagem montada.
  - Mensagem adicionada à lista de mensagens a serem enviadas.
- Extensões

- Não há

- Inclusões
  - Não há

3.2.6 Monta Mensagem (Servidor)

- Descrição
  - Este caso de uso descreve o processo de montagem de mensagem no cliente e no servidor.
- Evento iniciador
  - Processamento resultante da interpretação de uma mensagem do usuário.
  - Acontecimento no jogo que deve ser notificado aos usuários.
- Atores
  - Servidor
- Pré-condição
  - Não há
- Seqüência de eventos
  - Tipo da mensagem (broadcast ou individual) determinado de acordo com a ação do usuário. Caso for um acontecimento no jogo, a mensagem é de broadcast.
- Pós-Condição
  - Mensagem montada.
  - Mensagem adicionada à lista de mensagens a serem enviadas.
- Extensões
  - Não há
- Inclusões
  - Não há

3.2.7 Interpreta Mensagem

- Descrição
    - Este caso de uso descreve o processo de interpretação de mensagem pelo cliente e pelo servidor.
- Evento iniciador
    - Módulo detecta presença de mensagem na lista de mensagem recebida.
- Atores
    - Cliente, Servidor
- Pré-condição
    - Mensagem presente na lista de mensagens recebidas.
- Seqüência de eventos
    - Lê e interpreta a informação contida nos campos da mensagem (de acordo com as regras de montagem de mensagem).
- Pós-Condição
    - Mensagem retirada da lista de mensagens recebidas.
- Extensões
    - Não há
- Inclusões
    - Não há

3.2.8 Aceita Conexão

- Descrição
    - Este caso de uso descreve o processo de aceitação de conexão pelo servidor.
- Evento iniciador
    - Servidor recebe pedido de conexão de um cliente.

- Atores
    - Servidor
- Pré-condição
    - Servidor "escutando" determinada porta à espera de conexões.
- Seqüência de eventos
    - Cliente solicita conexão ao servidor através de uma determinada porta em modo não bloqueante.
    - Servidor aceita a conexão e adiciona o cliente à lista de clientes.
- Pós-Condição
    - Conexão estabelecida entre um cliente e o servidor. Cliente presente na lista de clientes do servidor.
- Extensões
    - Não há
- Inclusões
    - Não há

3.2.9 Termina Conexão

- Descrição
    - Este caso de uso descreve o processo de encerramento de uma conexão pelo servidor.
- Evento iniciador
    - Requisição de mensagem de término de conexão pelo usuário (cliente)
- Atores
    - Servidor
- Pré-condição
    - Conexão estabelecida entre um cliente e o servidor.

- Seqüência de eventos
    - Servidor envia mensagem de notificação a todos os clientes conectados e elimina o cliente que solicitou a desconexão da lista de clientes conectados.
- Pós-Condição
    - Cliente desconectado do servidor e removido da lista de clientes.
- Extensões
    - Não há
- Inclusões
    - Monta Mensagem, Envia mensagem

Apêndice E - Manual do usuário do módulo de redes do enJine

Este documento visa demonstrar de maneira prática e objetiva, dividida em passos, a maneira de se utilizar o enJine através do seu novo módulo de redes para a criação de jogos multiusuário. Para isso, será desenvolvida uma aplicação simples, porém completa, baseado no projeto de demonstração do enJine denominado "Simple", onde os usuários, executando aplicações cliente e conectados a um servidor, podem movimentar cubos coloridos através de um ambiente tridimensional.

Para acompanhar este material é necessário estar familiarizado com a linguagem Java, com a API Java3D e com o enJine. Não é escopo deste material ser fonte de aprendizado dessas tecnologias, cabendo ao usuário a iniciativa de aprendê-las. Materiais para aprendizado de Java podem ser encontrados em abundância, podendo-se citar como ponto de partida o tutorial da empresa Caelum disponível em: http://www.caelum.com.br/caelum/apostila/caelum-java-objetos-fj11.pdf [CAELUM]. O aprendizado da API Java3D pode ser acelerada através da ferramenta Interlab3D, desenvolvida pelo Interlab para esse fim e disponível juntamente com tutoriais em: http://interlab3d.incubadora.fapesp.br/portal/documents/tutorials/ [INTERLAB3D]. Com relação ao enJine a melhor maneira de aprender a utilizá-lo é através dos tutoriais disponíveis no site oficial no endereço: http://enjine.incubadora.fapesp.br/portal/Downloads/Documentation/Tutorials/. [ENJINE]

1 Cliente

GameClient

O aplicativo cliente é responsável apenas pela recepção de ações do jogador através do e atualização da visualização de acordo com a posição atualizada pelo servidor.

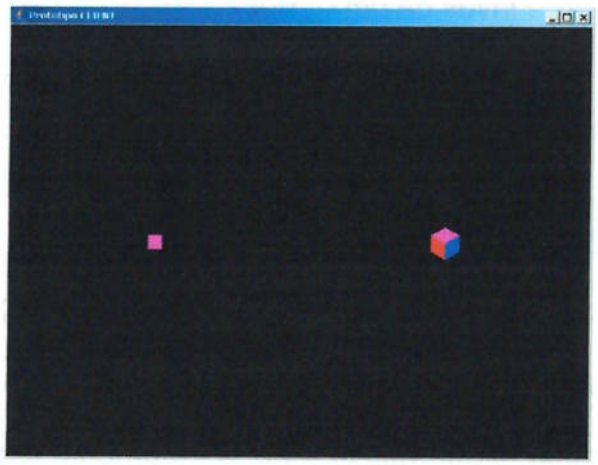

FIGURA 36 - Visualização do aplicativo cliente

A criação de um aplicativo cliente envolve a implementação de uma classe que deve possuir obrigatoriamente um atributo de classe `MultiplayerClient` e que possui um método `main` para a sua execução.

```
import interlab.engine.framework.MultiplayerClient;
...

public class GameClient {

      private MultiplayerClient game;
      ...

      public void run() {
            game.mainLoop(25, new ClientState());
      }
```

```
    public static void main(String[] args) throws Exception{
        GameClient simple = new GameClient();
        simple.initialize();
        simple.run();
        System.exit(0);
    }
}
```

Os métodos `initialize` e `run` são auxiliares e possuem rotinas que configuram os parâmetros do jogo e iniciam o seu loop principal, respectivamente. Suas implementações são idênticas às de um jogo local, exceto pelo fato do game ser agora um objeto do tipo `MultiplayerClient`, e o método `mainLoop` de game receber como parâmetro um `ClientState` invés de um `GameState`.

ClientState

Os estágios do jogo são implementados através da criação de classes-filhas de `MultiplayerState`, tanto no servidor quanto no cliente. Para o cliente, é necessário configurar para cada estágio, a sua atualização, o tratamento das mensagens recebidas, a sua tela e câmera, as teclas que devem ser monitoradas para a recepção de ações dos clientes e a criação dos jogadores.

Para a criação de um estágio cliente, é necessário ter uma referência para o `game` do cliente, ou seja, para um `MultiplayerClient`, conforme o trecho de código abaixo:

```
import interlab.engine.core.MultiplayerState;
...

public class ClientState extends MultiplayerState {

    private MultiplayerClient parent;
    public String messageBuffer = "";
    private InputAction walkRight;
    private InputAction walkLeft;
    private InputAction walkForward;
    private InputAction walkBack;
    public SimpleEntity player;
```

```
        public void initialize(Game game) {
                this.parent = (MultiplayerClient)game;

                //configuração de teclas

                try{
                        parent.connectToServer("192.168.0.1", 8642);
                }catch(Exception e){}

                //configuração de tela e câmera

                //criação do jogador
        }
        ...
}
```

No trecho de código apresentado, pode-se observar também o momento da conexão ao servidor e as configurações de teclas, tela, câmera e jogador no método `initialize` que precisa ser implementado.

A atualização do estado do jogo é realizado no método `update`, onde podemos detectar as entradas do usuário e enviar as mensagens referentes a cada entrada ao servidor.

```
public void update(Game game, float delta) {
        DefaultMultiplayerMessage message = new DefaultMultiplayerMessage();

        if(walkForward.getIntensity() > 0 &&
                walkBack.getIntensity() == 0){
                message.setMessage("1;0;U;"+player.position.x+";"+
                        player.position.y+";"+player.position.z + "@");
                parent.doWrite(message);
                walkForward.setIntensity(0);
        }
...
}
```

Observa-se que a mensagem definida está em formato de texto com suas partes separadas por `';'`. Assim como o cliente envia mensagens nesse formato, ele também recebe mensagens desse modo, por isso, é necessário implementar o método `treatMessage` para tratar mensagens do servidor. Esse método recebe como um de seus parâmetros o atributo do tipo `MultiplayerMessage`. Trata-se de uma interface através da qual se criou a implementação `DefaultMultiplayerMessage`.

```
public void treatMessage(Game game, MultiplayerMessage multiplayerMessage,
      Connection sender) {

      String messages =
            ((DefaultMultiplayerMessage) multiplayerMessage).getMessage();

      //Mensagem = tipo;id;direcao;x;y;z; onde tipo: 0=novo 1=action
      //2=atualiza_posicao 3=remove jogador

      messageBuffer = messageBuffer + messages;
      messageBuffer =  messageBuffer.replaceAll("@", "#@");
      String msgsArray[] = messageBuffer.split("@");
      messageBuffer = "";

      //Tratamento das mensagens
}
```

No trecho acima, primeiramente é realizado o casting de `multiplayerMessage` e é chamado o método `getMessage` implementado em `DefaultMultiplayerMessage`. Como em `messageBuffer` existem diversas mensagens, elas são separadas e armazenadas no vetor `msgsArray`. Em cada posição deste vetor, existe uma mensagem em formato texto com partes separados por ';' como mostrado anteriormente. A partir daí, deve-se separar as partes e tratá-las da maneira que desejar.

GameObject e Viewable

Com relação a criação dos objetos do jogo e das classes `Viewable` correspondentes, o mecanismo é exatamente o mesmo do utilizado na criação de um jogo sem a funcionalidade de redes como está descrito nos tutoriais do enJine citados no início.

2 Servidor

GameServer

O aplicativo servidor do jogo é responsável por receber a ação do cliente, tratar essa ação, atualizar o jogo em seu aspecto lógico e enviar ao cliente os resultados. Visto que o servidor trata somente a lógica do jogo, a criação de um `Viewable` é desnecessária. É preciso existir um `GameObject` e, opcionalmente, um `Colidable` para realizar o cálculo de colisões.

No servidor, uma das classes que deve ser criada deve possuir um atributo do tipo `MultiplayerServer`. A classe é bastante semelhante ao `GameClient`.

```
import interlab.engine.framework.MultiplayerServer;


public class GameServer {
	private MultiplayerServer game;

	public void initialize() throws Exception{
		game = new MultiplayerServer(4444);
		game.initialize();
		System.out.println("Server Initialized");
	}

	public void run() {
		game.mainLoop(25, new ServerState());
	}

	public static void main(String[] args) throws Exception{
		GameServer simple = new GameServer();
		simple.initialize();
		simple.run();
		System.out.println("Server Exited");
		System.exit(0);
	}
}
```

Nesse caso, é necessário passar ao construtor de `MultiplayerServer`, o número da porta na qual o servidor estará a espera de conexões.

ServerState

Assim como no cliente, e estágio do jogo precisa ser implementado no servidor através da implementação da classe `MultiplayerState`.

```
public class ServerState extends MultiplayerState {
	private MultiplayerServer parent;
	public String messageBuffer = "";

	public void initialize(Game game) {
		this.parent = (MultiplayerServer) game;
	}
	...
}
```

Além disso, é necessário criar os `GameObject` e adicioná-los à lista de abjetos replicáveis através da implementação dos métodos abaixo.

```
public void addReplicableObject(int id) {
	SimpleEntity player = new SimpleEntity(id,
		new Vector3d(0,0,0));
	parent.addObject(player);
	replicableObjects.add(player);
}
public void removeReplicableObject(int id) {
	SimpleEntity saiu=null;
	Iterator i=replicableObjects.iterator();
	while (i.hasNext()){
		SimpleEntity aux=(SimpleEntity)i.next();
		if(id==aux.id){
			saiu=aux;
			break;
		}
	}
	parent.removeObject((GameObject)saiu);
	replicableObjects.remove(saiu);
}
```

onde `replicableObjects` é uma lista definida em MultiplayerState.

É interessante implementar os métodos de montagem de mensagens de descrição do estado atual do jogo e de aviso de conexão de um novo cliente.

```
public MultiplayerMessage currentStateMessage() {
	String estado ="";
	Iterator i=replicableObjects.iterator();
	while (i.hasNext()){
		SimpleEntity aux=(SimpleEntity)i.next();
		double x = aux.position.x;
		double y = aux.position.y;
		double z = aux.position.z;
		estado+="4;"+aux.id+";"+x+";"+y+";"+z+"@";
	}
	DefaultMultiplayerMessage message = new DefaultMultiplayerMessage();
	message.setMessage(estado.substring(0,estado.length()));
	return message;
}

public MultiplayerMessage newPlayerMessage(int id) {
	DefaultMultiplayerMessage message = new DefaultMultiplayerMessage();
	message.setMessage("0;" + id +"@");
	return message;
}
```

Por fim, uma das tarefas mais importantes que é o tratamento das mensagens recebidas dos clientes.

```
public void treatMessage(Game game, MultiplayerMessage multiplayerMessage,
	Connection sender) {

	String message =
		((DefaultMultiplayerMessage) multiplayerMessage).getMessage();

	// Mensagem = tipo;id;direcao;x;y;z; onde tipo: 0=novo 1=action
	// 2=atualiza_posicao 3=remove jogador

	messageBuffer = messageBuffer + message;

	messageBuffer =  messageBuffer.replaceAll("@", "#@");
	String msg0[] = messageBuffer.split("@");
	System.out.println("(" +sender.connID + ")" +message);
	messageBuffer = "";

	String msg[] = msg0[0].split(";");
	double x = Double.parseDouble(msg[3]);
	double y = Double.parseDouble(msg[4]);
	msg[5] = msg[5].replaceAll("#", "");
	double z = Double.parseDouble(msg[5]);

	for (int i = 0; i < msg0.length; i++) {
		if(msg0[i].endsWith("#")){
			msg0[i] = msg0[i].replaceAll("#", "");
			msg = msg0[i].split(";");
			if (msg[0].equals("1")) {
				if (msg[2].equals("U")) {
					y += 0.2;
				} else if (msg[2].equals("D")) {
					y -= 0.2;
				} else if (msg[2].equals("L")) {
					x -= 0.2;
				} else if (msg[2].equals("R")) {
```

```
                        x += 0.2;
                    }
                }
            }
            else
            {
                messageBuffer = msg0[i].toString();
            }
        }

        DefaultMultiplayerMessage outMessage = new DefaultMultiplayerMessage();

        outMessage.setMessage("2;0;0;" + x + ";" + y + ";" + z + "@");
        sender.write(outMessage);
        outMessage.setMessage("2;"+sender.connID+";0;"+x+";"+y+";"+ z +"@");
        parent.sendBroadcast(sender, outMessage);
        Iterator i=replicableObjects.iterator();
        while (i.hasNext()){
            SimpleEntity aux=(SimpleEntity)i.next();
            if(sender.connID==aux.id){
                aux.position.set(x, y, z);
                break;
            }
        }
    }
```

Apêndice F – Conteúdo do CD em anexo

Um CD com o código fonte do enJine + módulo de redes, protótipo e o jogo demonstrativo Lego Adventures Online acompanha esse documento.

Além disso um conjunto de documentos entregues ao longo do ano para a matéria Projeto de Formatura também está contido nesse CD.

Por fim, a versão digital dessa monografia também está contida no CD.

A estrutura de diretórios do CD é a seguinte:

- /Monografia/ - contém a versão digital dessa monografia
- /Lego/ - contém o código fonte do jogo demonstrativo Lego Adventures Online
- /enJine/ - contém o código fonte da enJine + módulo de redes
- /protótipo/ - contém o código fonte do protótipo desenvolvido
- /documentos/ - contém um conjunto de documentos entregues ao longo do ano