UNIVERSIDADE DO SUL DE SANTA CATARINA

EDUARDO MAIA DE ALMEIDA

JOGOS SOCIAIS NA REDE:
CONECTANDO O FACEBOOK E ENGINES 3D EM UM JOGO MULTIPLAYER

Palhoça – SC

2013

EDUARDO MAIA DE ALMEIDA

JOGOS SOCIAIS NA REDE:
CONECTANDO O FACEBOOK E ENGINES 3D EM UM JOGO MULTIPLAYER

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade do Sul de Santa Catarina, como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Profª. Vera Schuhmacher

Palhoça – SC

2013

EDUARDO MAIA DE ALMEIDA

JOGOS SOCIAIS NA REDE:
CONECTANDO O FACEBOOK E ENGINES 3D EM UM JOGO MULTIPLAYER

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de Bacharel em Sistemas de Informação e aprovado em sua forma final pelo Curso de Graduação em Sistemas de Informação da Universidade do Sul de Santa Catarina.

Palhoça, novembro de 2013.

Profª. e orientadora Vera R. N. Schuhmacher, MEng.
Universidade do Sul de Santa Catarina

Prof. Rafael Lessa, Esp.
Universidade do Sul de Santa Catarina

Prof. Vinicius Ramos, Me.
Universidade do Sul de Santa Catarina

RESUMO

O objetivo deste trabalho foi pesquisar e criar métodos que facilitem o desenvolvimento de um jogo 3D *multiplayer* em tempo real para a rede social Facebook. No decorrer do trabalho procurou-se, inicialmente, trazer um esboço histórico sobre as redes sociais e os jogos, demostrando o grande potencial desse mercado. É desenvolvido e apresentado um conjunto de soluções propostas para a facilidade de se criar um jogo para o Facebook em *multiplayer*. Esse conjunto de soluções compreende as funcionalidades básicas de um jogo, conduzindo as ações dos personagens e gerenciando suas estatísticas de evolução, obtenção de itens, confrontos com seus inimigos e comunicação entre os vários participantes de um jogo em uma partida *multiplayer*, além de integrar essas ações com o Facebook. Foi desenvolvido também, um protótipo para apresentar as funcionalidades desses componentes.
Palavras-chave: Redes sociais. Jogos digitais. Jogo *Multiplayer*. *Unity 3D*.

ABSTRACT

The objective of this work was to make a research and to develop methods that facilitate the creation of a 3D real-time multiplayer game for the social network Facebook. Throughout the work it was sought to initially bring a historical sketch about social networking and games, demonstrating the great potential of this market. It is developed and presented a set of proposed solutions for an easy creation of a game for Facebook in multiplayer mode. This solution includes the basic features of a game, leading the character actions and managing his statistics evolution, obtaining items, clashes with enemies and communication between the participants of the game in a multiplayer match, and integrate these actions with Facebook. It was also developed a prototype to show the features of these components.
Keywords: Social networks. Digital games. Multiplayer game. Unity 3D.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE SIGLAS

AI - *Artificial Intelligence*

API - *Application Programming Interface*

FPS - *Frames Per Second*

GDD - *Game Design Document*

GUI - *Graphical User Interface*

HTML - *HyperText Markup Language*

HTTP - *Hypertext Transfer Protocol*

IP - *Internet Protocol*

JSON - *JavaScript Object Notation*

LAN - *Local Area Network*

MMO - *Massively Multiplayer Online*

PHP - *Personal Home Page*

RPC - *Remote Procedure Calls*

RPG - *Role-playing game*

SDK - *Software Development Kit*

SQL - *Structured Query Language*

TCP - *Transmission Control Protocol*

UDP - *User Datagram Protocol*

UML - *Unified Modeling Language*

URL - *Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

As redes sociais, como o Facebook, atraem cada vez mais usuários, servindo não só como uma rede de relacionamento entre as pessoas, mas também para propaganda, diversão e informação.

Segundo Recuero (2009), as redes sociais podem ser vistas como um conjunto de dois elementos: atores (pessoas, instituições ou grupos) e suas conexões (interações ou laços sociais), sendo uma metáfora para observar os padrões de conexões estabelecidas entre os atores.

As redes sociais possuem vários atrativos que fazem com que as pessoas venham a aderir a elas. Pode-se enumerar algumas características, de acordo com Figueiredo (2009), que fazem com que os usuários gastem muito tempo conectados, como:

a) livre acesso: os usuários podem acessar de qualquer dispositivo que possua conexão com a internet, gratuitamente e usufruir de todas as ferramentas que a rede social pode oferecer;

b) auto-expressão: se expressar é uma necessidade de todas as pessoas, e a rede social permite que cada um possa escrever sobre o que quiser, do jeito que quiser, sem o receio de estar sendo julgado e podendo emitir opinião sobre diversos assuntos;

c) colaboração: as redes sociais permitem que as pessoas possam fazer campanhas, divulgar necessidades, compartilhar informações e aprender umas com as outras. Ela instiga o senso de colaboração mútuo entre as pessoas;

d) entretenimento: assim como as redes sociais possuem um volume grande de informação importante, ela também, através de seus usuários, proporciona momentos de diversão com os compartilhamentos de vídeos, charges, tirinhas e fotos.

O Facebook é uma rede social que, de acordo com a STATISTICBRAIN, possui mais de 1 bilhão de usuários, é a mais visitada da atualidade. Esse sucesso é devido a grande quantidade de funcionalidades que atraem as pessoas, como permitir que as pessoas encontrem velhos amigos da época do colégio, informações diversas e uma grande comunidade de jogos.

Outro grande recurso que faz com que o Facebook cresça cada vez mais, é a sua ferramenta de desenvolvimento para interligar outros sites a ele, assim como jogos e *fan pages*[1]. Ele possui uma documentação completa para os desenvolvedores poderem unir seus códigos com as funcionalidades do Facebook. Além de ter uma documentação e uma comunidade de desenvolvimento bem grande, ainda possui uma ferramenta de medição do aplicativo. Dados como quantidade de usuários ativos por dia e semana, quanto tempo ficam dentro do jogo, a relação de *bugs*[2] que possam ter ocorrido, dentre outras.

O mercado de jogos encontra-se aquecido, e as redes sociais ajudaram muito nesse crescimento da área. Todas as pessoas que fazem parte da rede podem instalar jogos sem a necessidade de comprar o jogo, facilitando assim, sua divulgação, pois as funcionalidades de compartilhamento, curtir e comentar sobre o jogo, fazem com que este seja rapidamente propagado entre as pessoas.

Jogos sociais são jogos que possuem a característica de fazer com que o jogador possa não apenas vencer o jogo, mas também interagir, competir e fazer amigos de uma maneira diferente. São jogos diferentes dos tradicionais, como, por exemplo, um MMO[3], que possui uma interação bem definida entre os usuários dentro de um universo em comum, através de um servidor. Já nos jogos sociais, não existe um universo global, cada jogador tem o seu espaço, podendo interagir através da rede de amigos, resultando em uma interação assíncrona, ou seja, não se joga e interage ao mesmo tempo com outros jogadores. Porém, não é simples fazer um jogo. É preciso tempo e uma equipe especializada em várias áreas para conseguir desenvolver um jogo que atraia jogadores.

As empresas que fazem jogos sociais conseguiram uma grande ferramenta de exposição dos seus produtos com as redes sociais. Elas criaram uma oportunidade única para as pequenas e médias empresas do setor. Essas empresas fazem jogos por encomenda para empresas que pretendem fins publicitários, treinamento de funcionários e educacionais.

Com os avanços tecnológicos, surgiram grandes empresas que desenvolvem *game engines*, ou seja, um programa de computador com um conjunto de bibliotecas, para simplificar e abstrair o desenvolvimento de jogos eletrônicos ou outras aplicações com gráficos em tempo real. Como, por exemplo, a *Unity 3D*, que se tornou uma das *engines* mais populares do mercado, tanto pela sua qualidade como pela facilidade em aprender a utilizá-la.

---

1 São páginas criadas dentro do Facebook para que as organizações, empresas, celebridades e bandas transmitam muitas informações ao seus seguidores ou ao público que escolher se conectar a elas.

2 Erros que ocorrem no funcionamento da parte lógica de um software.

3 Jogo online que suporta conexões e partidas com milhares de pessoas ao mesmo tempo.

1.1 O PROBLEMA

Os jogos do Facebook hoje em dia estão ficando cada vez mais complexos, com várias funcionalidades e dinâmicas *multiplayer* (multijogadores) para permitir o jogador estar em tempo real interagindo com outros amigos. Isso determina uma maior complexidade da arquitetura do jogo dentro da rede social. Estes aspectos provocam uma resistência de desenvolvedores, pois o tempo necessário para seu desenvolvimento é maior do que os jogos fora das redes sociais.

A grande vantagem dos jogos *multiplayer* é a possibilidade de não ter só como objetivo vencer o jogo, mas sim estar se divertindo junto com seus amigos e obtendo o máximo que o jogo pode oferecer. Os jogos *single player* (para um jogador) perdem na questão de, ao jogador terminar o último objetivo do jogo, não continuar a jogar, procurando outro jogo.

A maior dificuldade encontrada nesse tipo de arquitetura é fazer com que os elementos dentro do jogo aconteçam de forma igual para os dois, ou mais jogadores, dentro de uma dinâmica do jogo, pois dentro desse universo, existem os elementos que são em comum aos jogadores e os elementos que são privados de cada jogador. Como exemplo, um inimigo no cenário deve ser visto e sofrer ações de todos os jogadores ao mesmo tempo, e já um menu de itens que o jogador tenha, é algo particular que só ele aciona e visualiza seus itens.

1.2 OBJETIVOS

Em seguida, serão apresentados o objetivo geral e os específicos deste trabalho.

1.2.1 Objetivo geral

O objetivo desta pesquisa é o desenvolvimento de componentes para a criação de um jogo 3D *multiplayer* básico e em tempo real para a rede social Facebook.

1.2.2 Objetivos específicos

São objetivos específicos deste projeto:

a) criar pacotes de códigos facilitadores para desenvolvedores de jogos *multiplayer* e para redes sociais;

b) a utilização da *Unity 3D* como *engine* para jogos;

c) desenvolver um protótipo de jogo que permita a aplicação dos componentes desenvolvidos;

d) servir de referência para iniciantes em desenvolvimento de jogos e redes sociais, para que tenham mais facilidade na utilização dessas tecnologias;

e) proporcionar ao jogador uma comunidade de amigos que por meio do jogo desenvolvam o senso de apoio mútuo e sociedade;

f) proporcionar ao jogador a possibilidade de convidar amigos para jogarem juntos em tempo real;

g) trazer para o ambiente do jogo um cenário lúdico e instigante que provoque a inquietação do jogador de forma que este se fidelize no uso do jogo motivado pela solução de novas missões.

1.3 JUSTIFICATIVA

Com o crescimento dos jogos sociais, que vieram para revolucionar o mundo do entretenimento *online*, cada vez mais surgem empresas desenvolvedoras desses tipos de

jogos, pois o mercado possui um crescimento exponencial. Esses jogos são uma fonte de lucro dentro das redes sociais, pois eles fazem com que os usuários estejam sempre jogando devido a sua interatividade e competitividade dentro da rede.

Este projeto tem sua importância para a comunidade desenvolvedora de jogos para redes sociais que deseja utilizar a *engine Unity 3D*, juntamente com as ferramentas que o Facebook disponibiliza para desenvolvimento dentro da sua rede, com a capacidade de tornar os jogos *multiplayer* em tempo real.

A ideia surgiu basicamente para que os desenvolvedores tenham menos tempo para gastar com bibliotecas de conexão com o Facebook e *multiplayer* e se dediquem mais nas dinâmicas do jogo em si e na implementação das regras para criar toda a parte de *gameplay*, que inclui todas as experiências do jogador durante a sua interação com os sistemas do jogo.

O projeto vem a atender a necessidade de pessoas que procuram entender como funcionam as conexões entre *server* (servidor) e *client* (cliente) dentro da rede social e sugerir uma arquitetura para os seus jogos.

Por justificativa final, o projeto é fonte de estudo e de pesquisa para desenvolvedores de empresas de jogos, autônomos e para estudantes que queiram se aprofundar mais nas dinâmicas de jogos *multiplayer* em redes sociais ou mesmo poder usar esses pacotes de códigos para facilitar e agilizar o processo de desenvolvimento do jogo.

1.4 ESTRUTURA DA MONOGRAFIA

Capítulo 1: Introdução

A Introdução orienta o leitor quanto ao assunto abordado, o problema que se propõe resolver, os objetivos do trabalho e a justificativa para o desenvolvimento do mesmo.

Capítulo 2: Revisão bibliográfica

Possui toda a referencia teórica que servirá de suporte para o desenvolvimento dos outros capítulos. Os principais assuntos abordados serão: redes sociais, Facebook e desenvolvimento de jogos.

Capítulo 3: Método

Apresenta a metodologia adotada para o desenvolvimento do trabalho, assim como a proposta e as definições do tipo de pesquisa.

Capítulo 4: Modelagem do sistema

O capítulo 4 apresenta uma análise, estudo e testes das ferramentas do Facebook e da *engine* de jogos *Unity 3D*, além dos diagramas UML construídos para a modelagem da aplicação.

Capítulo 5: Desenvolvimento

Neste capítulo, serão descritos os recursos tecnológicos utilizados para a modelagem e o desenvolvimento da aplicação, assim como a criação do *game design* e da arte, criação e descrição dos componentes e os testes realizados. O capítulo também se destina a fazer uma demonstração do protótipo do jogo, utilizando os componentes desenvolvidos, e a mostrar os resultados de uma avaliação do sistema e do jogo, feita por três possíveis usuários.

Capítulo 6: Conclusões e trabalhos futuros

Serão apresentadas conclusões sobre o projeto e o jogo, os resultados apresentados e considerações finais sobre o trabalho desenvolvido.

2 REVISÃO BIBLIOGRAFICA

Este capítulo apresenta os seguintes assuntos: Redes sociais, a dinâmica das redes sociais, Facebook, jogos e *multiplayer*.

2.1 REDES SOCIAIS

Segundo Castells (2000), a tecnologia das redes de informação começou a ter um grande progresso no início dos anos 90, que determinariam quatro grandes tendências que iriam fortalecer a internet: digitalização da rede de telecomunicações; desenvolvimento da transmissão em banda larga; uma melhora significativa no desempenho de computadores conectados pela rede; e avanços tecnológicos em hardware e software.

Segundo Marteleto (2001, p. 72):

> O conceito de redes é tributário de um conflito permanente entre diferentes correntes nas ciências sociais, que criam os pares dicotômicos – indivíduo/sociedade; ator/estrutura; abordagens subjetivistas/objetivistas; enfoques micro ou macro da realidade social – colocando cada qual a ênfase analítica em uma das partes.

Com a popularização da internet em todo o mundo depois dos anos 2000, que teve um aumento significativo de presença no trabalho e na casa das pessoas, os serviços de comunicação e entretenimento começaram a se fortalecer e sentir a necessidade de algo que causasse mais impacto e fizesse com que os internautas pudessem obter uma grande variedade de informação e diversão. Com isso surgiram as redes sociais.

Conforme Silva e Ferreira (2007) consideram, rede social é um conjunto de pessoas (ou empresas, ou qualquer outra entidade socialmente criada) que estão interligadas por um conjunto de relações sociais entre si, como amizade, relações de trabalho, trocas comerciais ou de informações.

As redes sociais sempre existiram porém de forma um pouco diferente, pois no passado as sociedades eram mais coesas e a forma de conseguir informação e interagir eram diferentes. Com o avanço tecnológico, a sociedade passou a usufruir das vantagens que a

internet oferece, podendo acessar conteúdos de qualquer lugar, sem a necessidade de estar presente com outras pessoas para discutir sobre um assunto ou trocar informação.

Conforme Acioli (2007) afirma, as redes nascem num espaço informal de relações sociais, porém os efeitos que ela pode causar são visíveis além do espaço entre as relações com o Estado, a sociedade e outras instituições representativas.

Com isso, as redes sociais surgiram com o intuito de criar interações informais entre indivíduos em suas relações cotidianas. Elas unem interesses, necessidades e objetivos em comum entre as pessoas, para criar um ambiente onde todos podem escrever sobre qualquer tema ou situação. Além disso as redes sociais possuem uma variedade enorme de produtos dos quais fazem os usuários gastarem cada vez mais tempo do dia interagindo.

Conforme Castells (1999), o desenvolvimento global permite aos países e as suas populações, a possibilidade de funcionar produtivamente na economia global e na sociedade em rede. Isto significa que a difusão entre tecnologias de informação e comunicação cheguem em todo o mundo. Mas também significa uma maior necessidade de possuir a produção de recursos humanos necessários para operar neste sistema, e a distribuição de capacidade de gerar conhecimento e informação.

Dentro desses conceitos de redes sociais surgem os software sociais. Segundo Machado e Tijiboy (2005), os softwares sociais são programas que funcionam como mediadores sociais. Estes favorecem a criação de redes de relacionamentos com o intuito de juntar pessoas em seu círculo de relacionamentos, conhecer outras que compartilhem os mesmos interesses e discutir temas variados, construindo diferentes elos.

2.1.1 Relações sociais

Segundo Recuero (2009, p. 37):

> A ideia de relação social é independente do seu conteúdo. O conteúdo de uma ou várias interações auxilia a definir o tipo de relação social que existe entre dois interagentes.

O mundo é composto por redes parcialmente conectadas e as redes sociais fornecem um mecanismo flexível de organização social. A relação social pelo computador

traz aspectos importantes, como o distância entre as pessoas envolvidas na construção dessa relação. Os estudos dessas relações devem seguir alguns pontos, conforme Wasserman e Faust (1994):

a) mutualidade dos laços ou conexões;

b) proximidade dos membros do subgrupo;

c) frequência dos laços dentre os membros;

d) frequência dos laços entre os membros em relação aos não-membros do grupo.

De acordo com Wasserman e Faust (1994), a conexão apresentada entre dois atores em uma rede social é denominada laço social, que é composto por relações sociais, que por sua vez, são constituídas por interações sociais.

Os laços sociais podem ser fortes e fracos. Segundo Recuero (2009, p. 41), os laços fortes são aqueles que se caracterizam pela intimidade, pela proximidade e pela intencionalidade em criar e manter uma conexão entre duas pessoas e, por outro lado, os laços fracos caracterizam-se por relações esparsas, que não traduzem proximidade e intimidade.

Esses laços sociais constituídos entre os atores, são difíceis de ser percebidos na internet. No entanto, a partir da observação sistemática das interações, é possível perceber elementos como o grau de intimidade entre os interagentes.

2.2 A DINÂMICA DAS REDES SOCIAIS

A rede social está ligada ao relacionamento entre entretenimento e interação dos seus usuários. Todo o processo desde o seu início está ligado a pessoas se conhecerem, trocarem informações, manter contato e se divertirem. Devido ao grande número de usuários, as redes sociais também recebem um grande volume de propagandas, onde muitas empresas estão investindo e conseguindo divulgar a sua marca e seus produtos.

É evidente que os usuários das redes sociais podem ajudar um produto a ser divulgado, porém, nem sempre é colaborativo esse tipo de atitude. A dinâmica que as redes sociais alcançaram, permite que opiniões e questionamentos sobre diversos tipos de assuntos, sejam muito influentes, atingindo um grande grupo de pessoas a compartilharem dessa mesma opinião.

Conforme CASTRO (2012, p. 139):

> A capacidade de lidar de modo eficiente com o consumidor-internauta que expressa publicamente sua opinião sobre marcas, produtos e serviços já há muito tem sido considerada atributo indispensável no mundo dos negócios. O monitoramento de redes sociais faz parte de estratégias corporativas nos mais diversos segmentos de mercado. É preciso saber estar presente na rede, assim como também interagir a tempo e de forma adequada com a opinião pública na arena digital.

As empresas têm usado muito a mídia social para divulgar seus produtos. São criadas páginas para fãs das marcas/produtos, são colocados vídeos dentro das paginas principais das redes, dentro dos jogos sociais e também conseguem criar uma lista de usuários/fãs para poder fazer sorteios e promoções.

A Figura 1 mostra o potencial que uma marca consegue chegar em divulgação na rede social que, além de não conter custo direto para a empresa, atinge um número muito alto de pessoas que recebem informações e divulgam a marca.

Figura 1 – A exposição das marcas nas redes sociais

Fonte: *fan-page* Coca-Cola no facebook.com (2013).

As dinâmicas que as redes sociais utilizam são bem simples como forma de atrair cada vez mais pessoas para a sua rede. Permitir que qualquer um tenha uma conta, sem precisar gastar nada para isso, poder acessar a sua conta de qualquer lugar, inclusive em dispositivos móveis, compartilhar fotos, vídeos e pensamentos, permitindo que seja escolhido

o tipo de privacidade (apenas amigos, toda a rede ou algumas pessoas especificas), jogar os jogos sociais, comentar e criar grupos de discussões sobre qualquer tipo de assunto.

Por outro lado, o sistema por trás da rede é bem complexo, devido à necessidade de se ter uma alta confiabilidade dos dados que trafegam dentro da rede e à estrutura de servidores para comportar o alto número de usuários postando conteúdo e acessando ao mesmo tempo em todo o mundo.

Algumas funcionalidades oferecidas pelas principais aplicações de redes sociais *online* podem ser enumeradas a seguir, conforme Benevenuto, Almeida e Silva (2011):

a) perfis dos usuários: uma página individual que oferece uma descrição. Perfis podem ser utilizados não só para identificar o indivíduo no sistema, mas também para identificar pessoas com interesses em comum e articular novas relações;

b) atualizações: são formas efetivas de ajudar usuários a descobrir conteúdo. As redes sociais geralmente fazem as atualizações aparecerem imediatamente na página principal para incentivar outras pessoas a verem e comentarem;

c) comentários: a maior parte das aplicações de redes sociais *online* permitem que usuários comentem o conteúdo compartilhado por outros;

d) avaliações: o conteúdo compartilhado por um usuário pode ser avaliado por outros usuários;

e) listas de favoritos: permitir que usuários selecionem e organizem seu conteúdo. Listas de favoritos ajudam usuários a gerenciar seu próprio conteúdo e podem ser úteis para recomendações sociais;

f) listas de mais populares: listas baseadas em avaliações ou outras estatísticas do sistema relativas ao conteúdo;

g) metadados: usuários associam metadados, como título, descrição e *tags*, ao conteúdo compartilhado. Metadados são essenciais para recuperação de conteúdo em redes sociais *online*.

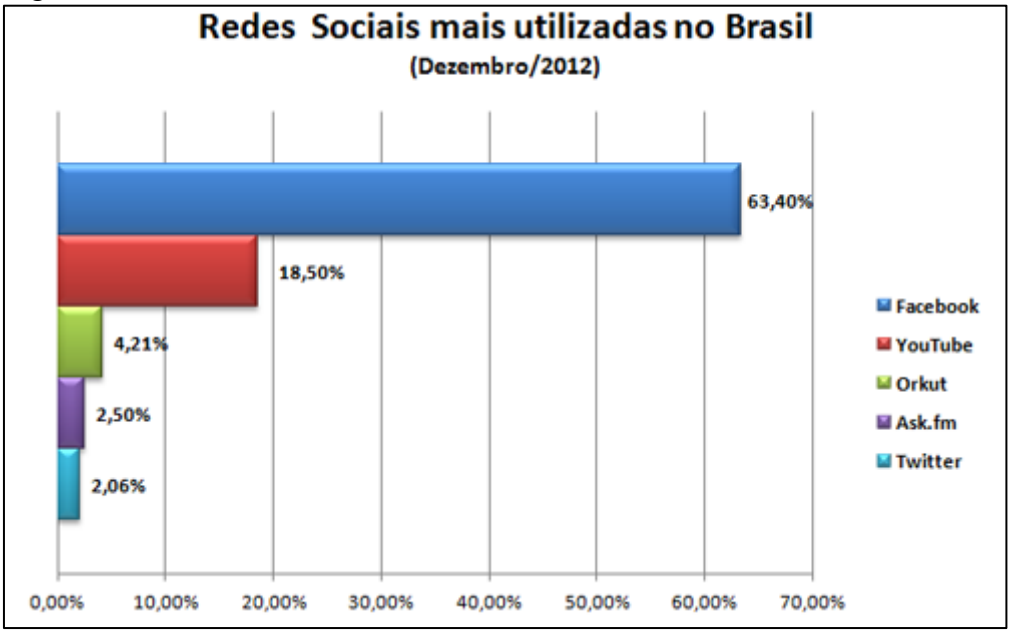
2.2.1 Dados gerais

2004 pode ser considerado o ano das redes sociais, pois nesse período foram criadas várias delas, como, por exemplo, o Facebook, Flickr e Orkut. No final de 2012, o site

de monitoramento de rede PINGDOM mostrou os resultados em números de como as redes sociais estão cada vez mais crescendo no mundo da tecnologia. O Facebook possui mais de 1 bilhão de usuários ativos, o Twitter tem mais de 200 milhões, o Linked In com mais de 187 milhões e o Google+ com mais de 135 milhões de usuários ativos.

O Brasil hoje é o 7º mercado de internet do mundo. Possui mais de 85% dos internautas utilizando alguma rede social. A Figura 2, mostra os dados das redes sociais mais acessadas no Brasil até o final do ano de 2012.

Figura 2 – As redes sociais em números no Brasil



Fonte: Techtudo (2012)

Conforme os números apresentados, fica claro que o mercado das redes sociais está aquecido. Os investimentos das empresas para conseguir entrar nesse mercado têm sido cada vez maior, ajudando a criar mais empregos em várias áreas de atuação desse tipo de segmento.

2.3 FACEBOOK

O Facebook é uma rede social que surgiu em 2004, feita por universitários de Harvard. No início, ele era disponível apenas para universitários da própria universidade. Em

pouco tempo, a rede teve uma grande quantidade de acessos e começou a se popularizar. A ideia inicial era a mesma que a maioria das redes sociais têm, como compartilhar fotos e ter um espaço virtual para interagir com os amigos e conhecer novas pessoas. Só que o Facebook não ficou apenas nisso. Aos poucos, ele foi inovando o seu site com várias funcionalidades que atraiam mais e mais pessoas, fazendo com que o mercado ficasse de olho nele e começasse a investir nessa rede social.

Conforme Kirkpatrick (2011), o Facebook muda a forma como as pessoas se comunicam e interagem, como os comerciantes vendem seus produtos, como os governos chegam aos cidadãos e até como as empresas operam.

Aos poucos, as novidades foram surgindo, como aplicativos dentro da rede, que permitem os usuários colocarem conteúdo externo dentro de uma página do Facebook, as "*fan pages*", que são direcionadas para empresas, marcas ou produtos que desejam interagir com seus clientes e a opção de poder postar vídeos e músicas, criar eventos e convidar as pessoas.

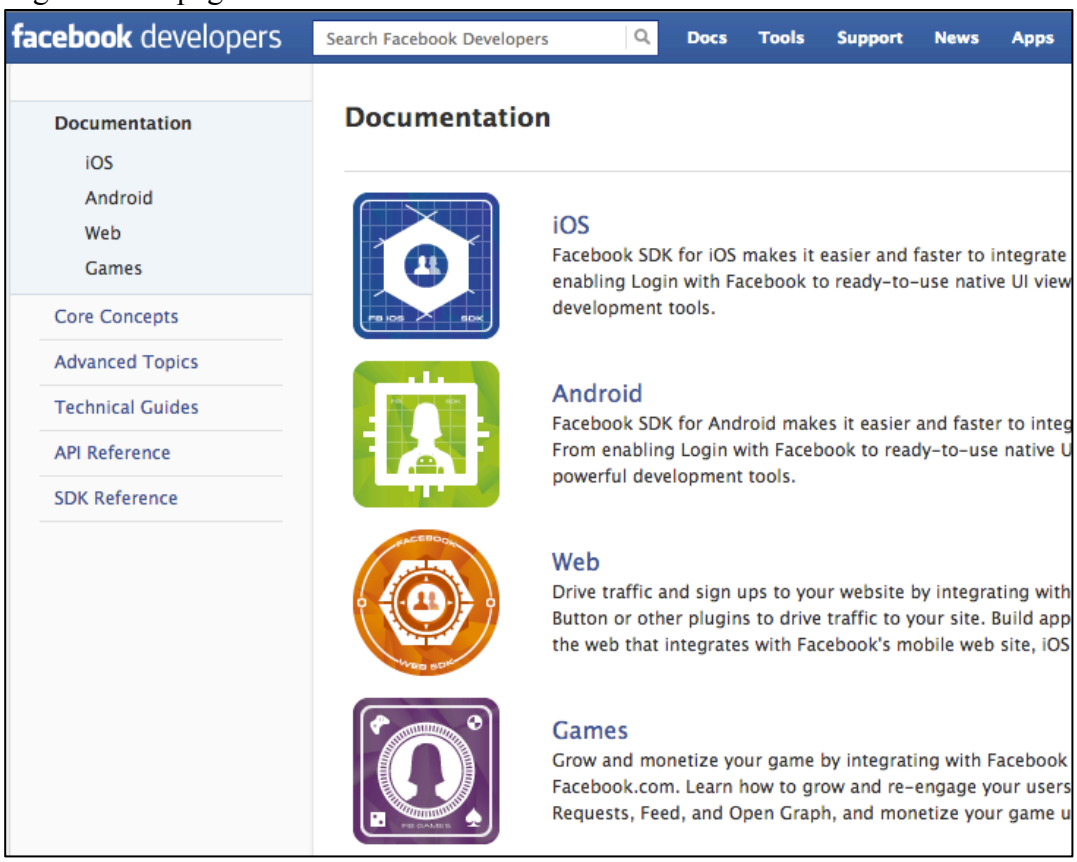
O Brasil hoje é o segundo país com o maior número de usuários ativos no Facebook, com mais de 47 milhões e fica atrás apenas dos Estados Unidos. E com o surgimento dos jogos dentro do Facebook, cada vez mais pessoas estão se tornando usuários ativos e gastando mais tempo dentro da rede.

2.3.1 Desenvolvendo para Facebook

O Facebook criou uma página para colocar toda a documentação necessária para que os desenvolvedores possam criar aplicativos que utilizem os recursos dele. Para poder acessá-la, o desenvolvedor precisa ter uma conta no Facebook.

A Figura 3 mostra como é organizada a página de desenvolvedores do Facebook.

Figura 3 – A página de desenvolvedores do Facebook



Fonte: developers.facebook.com (2013).

O Facebook disponibiliza a documentação para desenvolvimento *mobile* (Android e iOS), web e de jogos. Além de disponibilizar a documentação de referencia da sua API e dos SDKs para Javascript, PHP, iOS e Android.

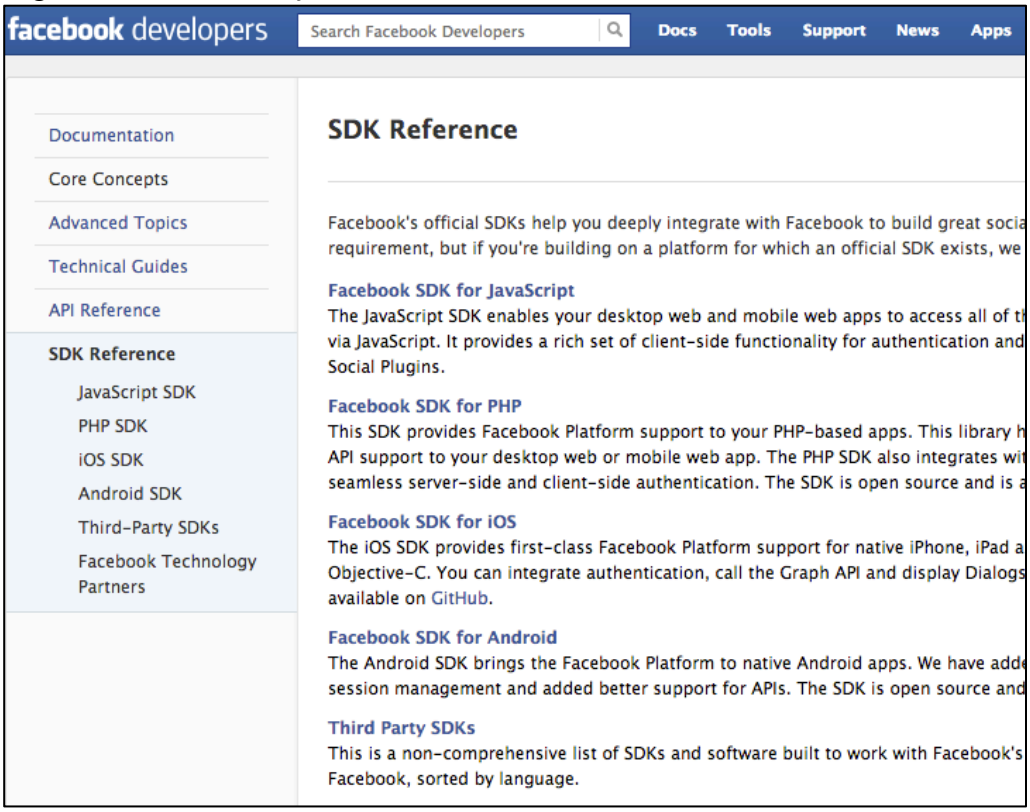
A API, cujo significado é Interface de Programação de Aplicativos, é um conjunto de rotinas e padrões estabelecidos por um determinado software, para que sejam utilizadas as suas funcionalidades por aplicativos que não precisam se importar com os seus detalhes de implementação e apenas usar os seus serviços. A API do Facebook é muito rica e bastante consistente, fazendo com que várias plataformas interajam em diversos tipos de aplicativos.

SDK é um Kit de Desenvolvimento de Softwares, e o Facebook disponibiliza o SDK para PHP que é usado para criar configurações e funcionalidades na parte do servidor, ou *server-side*, do qual pode acessar funcionalidades do Facebook, através de chamadas da API. O SDK do PHP pode trabalhar em conjunto com o SDK do Javascript para gerenciamento de sessões entre cliente e o aplicativo, pois o Javascript interage com o usuário sem a necessidade de o script passar pelo servidor, realizando comunicação assíncrona e alterando o conteúdo do que está sendo exibido.

Além disso, o Facebook ainda possui em sua documentação o SDK para iOS e Android, que possui um conjunto de referencias para o desenvolvimento utilizando a sua API *mobile*, e também referencias para outros tipos de linguagem de programação que utilizem a API do Facebook, como Java, .Net (desenvolvimento em C#), Flash (desenvolvimento em ActionScript).

A Figura 4 mostra como é organizada a página de documentação dos SDKs que o Facebook utiliza.

Figura 4 – Documentação do Facebook



Fonte: developers.facebook.com (2013).

2.4 JOGOS

Conforme Adams (2009), um jogo é um tipo de atividade lúdica, realizada no contexto de uma realidade simulada, em que os participantes tentam alcançar, de forma arbitrária, um objetivo não trivial, agindo de acordo com as regras.

Os jogos são um instrumento milenar de diversão, que podem envolver estratégia e informação, além de desenvolver habilidades de raciocínio, inteligência espacial, memória, coordenação motora, entre outras competências. Segundo Juul (2012), a tração pelos jogos é uma experiência subjetiva da qual varia de pessoa para pessoa, dependendo da experiência vivida com jogos anteriores, gostos pessoais, e se a pessoa está disposta a dar ao jogo o tempo que ele pede.

Segundo Crawford (1984), os jogos são uma parte fundamental da existência humana. Um jogo digital, segundo Crawford, é caracterizado por quatro aspectos principais:

a) representação: um sistema formal fechado que representa um subconjunto deliberativamente simplificado da realidade;

b) interação: permite ao participante explorar elementos interconectados e observar relações de causa e efeito entre os mesmos;

c) conflito: ao buscar os objetivos propostos, o jogador deve enfrentar obstáculos ativos ou, no mínimo, dinâmicos;

d) segurança: uma forma segura de experimentar aspectos da realidade representada (ultra-realidade, realidade aumentada). Essa é uma definição um tanto restritiva e mais próxima do conceito seguido pela indústria de jogos digitais e diversões eletrônicas em geral.

O desenvolvimento de um jogo é um processo muito complexo, depende de várias áreas interligadas e profissionais com um bom conhecimento técnico e das tendências que prendem um jogador a se identificar e se divertir com o jogo.

Segundo Demasi (2005), o mercado de jogos é extremamente competitivo, bilhões de dólares são investidos e o público é extremamente exigente. Desenvolver jogos lida com quase todas as áreas da computação, e não apenas programação e computação gráfica. Existem alguns segmentos para a programação de jogos, cada um com as suas características próprias, como consoles, computadores e dispositivos móveis.

A criação de um jogo exige que as pessoas envolvidas atravessem várias fases até que o jogo seja publicado, como definir uma ideia, criar um *game designer*, ou seja, planejar o jogo, criar um roteiro, esboçar as artes, programar através de códigos, criar efeitos sonoros e trilha sonora, passar por fases de testes e divulgação.

Para que a criação de um jogo siga o processo correto, é preciso além de ter um projeto bem definido, um documento de onde toda a equipe vai se guiar. Esse documento é chamado de GDD. Nele são descritas todas as características do jogo, assim como o

gameplay, que é onde vai ser definida toda a jogabilidade, ou seja, as regras do jogo e o balanceamento das regras. É nessa parte que o jogo deve proporcionar a diversão e os desafios interessantes. Esta parte do documento é de grande importância para guiar os programadores.

Segundo Rollings e Morris (2004), existe algo mais importante para um jogo do que o *gameplay*, a interatividade. Ela é o conjunto de tudo que o jogador pode interagir e modificar no meio ambiente do jogo. O *gameplay* é um tipo de interatividade que junto com um conjunto de interações formam a alma do jogo.

Rolling e Morris (2004, p. 111) exemplificam alguns tipos de interação:

a) afetar o próprio mundo do jogo, seja alterando configurações ou uma parte do jogo;

b) controlar as ações dos personagens ou um grupo de personagens;

c) influenciar nas ações dos personagens, como, por exemplo, trocar de arma;

d) decidir o que e quem seguir.

Com os avanços tecnológicos, os jogos passaram a ter cada vez mais complexidade, a qualidade gráfica e as mecânicas ficaram mais parecidas com a realidade e os roteiros ficaram cada vez mais parecidos com os roteiros dos filmes. Os softwares de criação artística possuem mais recursos e exigem cada vez mais dos computadores para se conseguir trabalhar neles.

Considerando o aspecto computacional que as aplicações requerem, Battaiola (2000) comenta que é necessária a adoção de sofisticadas técnicas que representam o "estado da arte" das pesquisas em Ciência da Computação. Algumas dessas pesquisas estão relacionadas com a análise de algoritmos, computação gráfica, redes de computadores e inteligência artificial.

2.4.1 Classificação dos jogos

Os jogos possuem uma classificação por gêneros, assim como os livros e os filmes. Segundo Adams (2009), o gênero é uma categoria dos jogos caracterizada por um determinado conjunto de desafios, independentemente da definição ou do conteúdo do mundo

do jogo. Assim, os gêneros dos jogos foram classificados conforme Adams (2009, p. 70-71) como:

a) ação: jogos que incluem desafios físicos, dos quais também podem incorporar *puzzles* (quebra-cabeças), corrida e uma variedade de desafios com conflitos;

b) estratégia: incluem estratégias, táticas e alguns desafios logísticos;

c) rpg[4]: envolvem tática, logística e desafios de exploração. Geralmente envolvem coleta e comércio de itens e também incluem *puzzles* e desafios conceituais;

d) simulação: incluem jogos de esporte e simulação de veículos, utilizando principalmente desafios de física e táticas;

e) construção e gestão: oferecem desafios conceituais e econômicos dentro dos jogos;

f) aventura: fornecem exploração de cenários e resolução de *puzzles*;

g) *puzzles*[5]: oferecem desafios lógicos e conceituais.

Esses gêneros podem também ser combinados, gerando os jogos híbridos, que mesclam entre dois ou mais gêneros para torná-los mais atrativos aos jogadores. Com a entrada dos jogos sociais no mercado, outro gênero que ficou bem conhecido foram os *advergames*, que utilizam a estratégia de comunicação mercadológica dentro dos jogos como ferramenta para divulgar e promover marcas, produtos, organizações e/ou pontos de vista.

Além dos gêneros, os jogos também possuem outro tipo de classificação, que segundo Adams (2009, p. 14) são as competições e cooperações. As competições ocorrem quando o jogo possui um conflito de interesses entre os jogadores e as cooperações ocorrem quando esses interesses convergem entre si. Os tipos de competição e cooperação podem ser classificados como:

a) competição entre dois jogadores;

b) *multiplayer* competitivo, ou seja, "cada um por si" em um mesmo cenário

c) *multiplayer* cooperativo, onde todos os jogadores se ajudam com um objetivo em comum;

d) baseado em times, onde os membros de um time cooperam entre si competindo contra um outro time;

e) *single-player*, ou seja, o jogador contra a máquina ou a situação do jogo;

---

[4] Conhecido como um jogo de interpretação de personagens onde os jogadores assumem os papéis de personagens e criam narrativas colaborativas.

[5] Tipo de jogo onde um jogador deve resolver um problema proposto utilizando o raciocínio lógico.

f) modos de competições híbridas, onde o jogo permite cooperação em alguns momentos, porém, de forma global, o jogo é de competição.

2.4.2 Processos de elaboração dos jogos

Como em qualquer outro software, os jogos também passam por um processo de desenvolvimento que requer documentação e colaboração de todas as áreas envolvidas. Para a integração da equipe, é necessário um gerente de projetos que faça a ligação de todos os elementos em harmonia. Nesse processo de desenvolvimento de um jogo, Clua e Bittencourt (2005) citam as seguintes etapas:

a) confecção do *Design Bible*;

b) produção de áudio e imagens;

c) modelagem 3D;

d) integração da parte artística com a programação.

A confecção do *Design Bible* é uma espécie de manual do jogo para que todas as áreas envolvidas possam se guiar para o desenvolvimento do jogo. É necessário que as áreas saibam a fundo todos os aspectos do jogo, conhecer o que deve ser feito para atingir as expectativas dos usuários.

Quando a etapa de documentação termina e as equipes passam a conhecer o que é o jogo, entra a parte de definir a tecnologia a ser usada, como as *engines* de criação de jogos ou até mesmo criar uma *engine* própria.

2.4.3 Jogos no Facebook

Os jogos para Facebook, os chamados “jogos sociais”, surgiram a partir da criação de um framework para que os desenvolvedores pudessem criar aplicações da qual conseguisse se comunicar com recursos internos do Facebook. Todos os recursos que o Facebook

disponibiliza para os desenvolvedores possui uma vasta documentação a fim de um melhor entendimento da sua estrutura.

Segundo Déda e Zagalo (2010, p. 11):

> O jogo torna-se uma ferramenta de engajamento social, pois através dela podem-se conhecer outros jogadores que juntos estimulam formas de compartilhamentos inseridos em uma relação social, que prioriza a confiabilidade e a lealdade na suas ações expostas, sendo a socialização a meta matriz do jogo na perspectiva de um contexto interativo.

A estrutura que o Facebook possui para o desenvolvimento atrai cada vez mais empresas para lançarem seus jogos sociais dentro da rede. A facilidade de promover a marca e conseguir inúmeros jogadores são características que a rede social possui para conseguir cada vez mais pessoas investindo nesse tipo de jogo.

Os jogos não possuem custo para os usuários começarem a jogar, porém a maioria usa a estratégia de fazer com que esses usuários venham a gastar com o passar do tempo. Recuero (2009) entende que os gastos dos usuários acabam ocorrendo por algum motivos:

a) os jogos fazem com que as pessoas queiram cada vez mais se superar e passar o tempo se divertindo, porém chega em um ponto do jogo que o jogador precisa esperar um determinado tempo para poder voltar a jogar, ou pagar, para poder voltar a jogar de forma imediata;

b) devido ao senso de disputa que as pessoas possuem, os jogos sociais na sua maioria possuem um ranking que pode ser compartilhado na página principal, fazendo com que o usuário queira evoluir mais rápido no jogo e ser o melhor no seu grupo de amigos;

c) existem itens dentro dos jogos que só podem ser usados pelos jogadores que pagarem pelo item.

Devido ao grande número de usuários que gastam dinheiro dentro dos jogos sociais é que cada vez mais é investido dinheiro nesse mercado. De acordo com a GAMEREPORTER, o *PayPal*, serviço de transações comerciais *online*, mostra que mais de 12 milhões de usuários pagam mensalmente para jogar no Facebook.

2.4.4 *Engines*

Conforme Batista e Lima (2009), *game engine*, ou motor de jogo, pode ser definido como um conjunto de funcionalidades reutilizáveis em alto nível que auxiliam o desenvolvedor a implementar as tarefas mais comuns de um jogo. As *engines*, dentro da área de jogos, são softwares que se encarregam por controlar os modelos a serem renderizados, tratar das entradas de dados do jogador e todos os processos de baixo nível e se entender com os hardwares gráficos. Algumas características que, segundo Clua e Bittencourt (2005), as *engines* devem possuir são:

a) permitir o uso da mesma para criação de inúmeros jogos diferentes;

b) o reaproveitamento de código que já foi desenvolvido anteriormente;

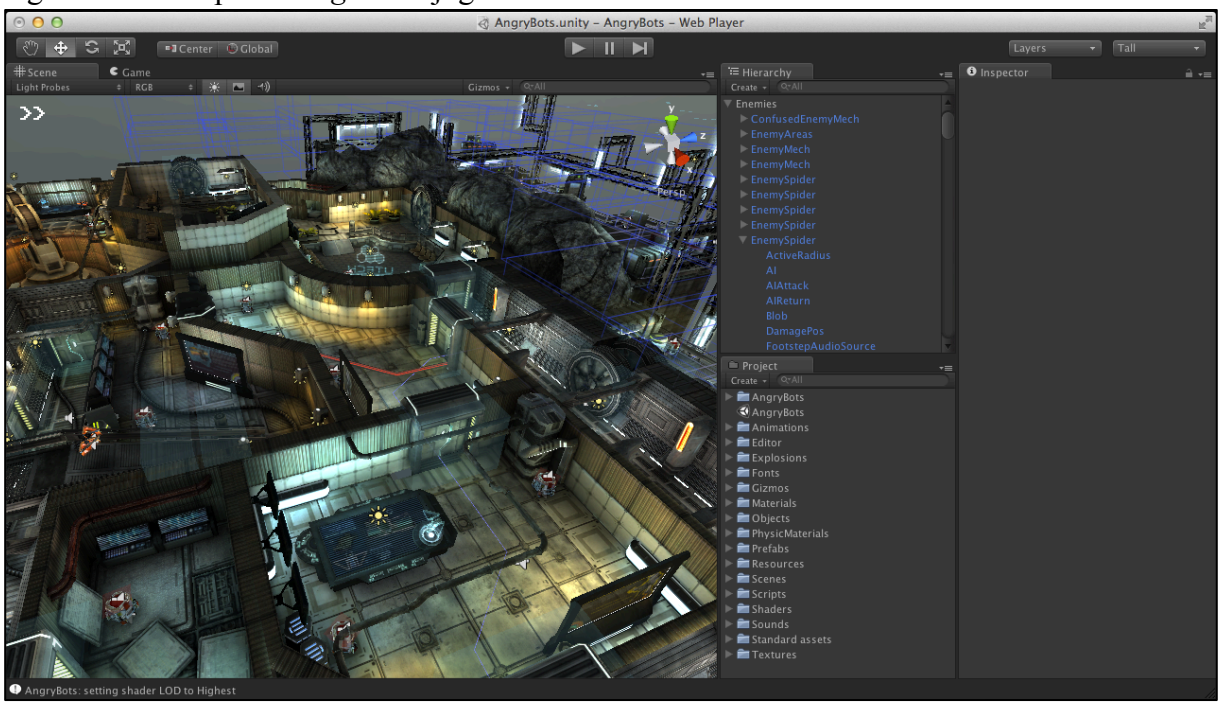
c) fazer com que os programadores não precisem manipular APIs e sim utilizar as próprias APIs dentro do ambiente da *engine;*

d) facilitar a integração entre os códigos gerados pelos programadores e a parte de modelagem 3D, devido à *engine* geralmente possuir editores de cena.

A escolha da *engine* é muito importante para que o projeto do jogo alcance os resultados desejados. As *engines* possuem características próprias para determinados tipos de projeto. Algumas são melhores para utilizar com 2D, outras com 3D e também levar em conta a sua integração com os softwares específicos para modelagem e animação. A linguagem de programação que ela utiliza também é importante, pois na hora de decidir qual utilizar, precisa ser de acordo com as qualificações técnicas dos programadores.

A Figura 5, mostra um exemplo de *engine* utilizada para jogos, a *Unity 3d*.

Figura 5 – Exemplo de *engine* de jogos



Fonte: unity3d.com (2013).

Para a escolha da *engine*, Clua e Bittencourt (2005) levantam mais alguns pontos que devem ser levados em consideração. São eles:

a) o orçamento disponível para a produção do jogo, pois existem *engines* muito caras e *engines* baratas. Isso faz diferença na quantidade de recursos que a *engine* possui e também no suporte que a empresa que criou a *engine* vai dar;

b) a plataforma que o jogo irá rodar, pois existem várias plataformas, como por exemplo PC, X-BOX e Playstation, e algumas *engines* só fazem para algumas plataformas específicas;

c) documentação oferecida pela *engine* para que os profissionais que irão utilizá-la tenham uma boa referência de pesquisa.

2.5 *MULTIPLAYER*

Os jogos *multiplayer*, ou jogos multijogadores, são jogos que oferecem aos jogadores a possibilidade de poderem jogar ao mesmo tempo dentro de um mesmo cenário e

interagindo entre si e com os componentes do jogo. Segundo Adams (2009, p. 3), esse tipo de jogo requer também cooperação social entre os participantes e todos devem ter as mesmas chances de ganhar. A intenção desse tipo de jogo é criar um ambiente de disputa, cooperação ou rivalidade, e proporcionar uma forma de comunicação social que geralmente está ausente em jogos do tipo *single player*, ou jogo eletrônico para um jogador.

O trabalho de um jogo *multiplayer* é replicar em vários clientes (jogadores) o mesmo jogo. Isso ocorre no servidor que gerencia tudo que acontece no jogo e envia para os clientes, assumindo que seja um sistema cliente-servidor. O primeiro passo para a criação de um jogo *multiplayer* é criar um servidor. O servidor pode ser um dos jogadores, que no caso estaria esperando uma conexão com o outro jogador ou em um servidor dedicado.

É fundamental saber que nos jogos *multiplayer* a autoridade é sempre o servidor, pois ele controla tudo que deve acontecer e manter os clientes atualizados. Isso faz com que sejam minimizadas as chances de alguém tentar burlar o sistema. Como exemplo de controle do servidor, podemos citar que se um cliente recebe um pacote do servidor informando uma determinada posição no cenário e esta não é a atual do jogador, este vai ser deslocado até a posição informada pelo pacote vindo do servidor.

Esse tipo de jogo chama muito a atenção de empresas que pretendem patrocinar o desenvolvimento de jogos, devido a grande quantidade de usuários ativos por dia e o tempo que se é gasto dentro deles. Por exemplo, os jogos do tipo MMO permitem milhares de pessoas jogarem nesse mundo virtual dinâmico ao mesmo tempo.

A Figura 6 mostra um exemplo de jogo *multiplayer*.

Figura 6 – Exemplo de um jogo *multiplayer*

Fonte: blizzard.com (2013).

Outro grande desafio dos jogos *multiplayer* é o *lag*, que é um termo usado para mostrar o tempo que leva para que uma ação de um jogador ocorra de verdade e apareça para todos os jogadores da rede.

2.5.1 Socialização dos jogos *multiplayer*

Os jogos mais convencionais são os jogos *multiplayer*, pois eles instigam uma atividade social. As pessoas gostam de jogar jogos juntas e a tecnologia dá-lhes muitas maneiras de fazê-lo. Segundo Adams (2009, p. 26-27) as formas de socializar dentro dos jogos podem ser das seguintes formas:

a) *multiplayer* local, onde duas ou mais pessoas jogam juntos em um mesmo lugar. É um tipo clássico de jogo de console jogado por mais de uma pessoa, onde cada jogador tem seu próprio controlador, mas todos olham para a mesma tela e todos os jogadores veem o mesmo mundo de jogo em conjunto;

b) *multiplayer* em rede, que refere-se a pessoas jogando contra ou em conjunto com outras pessoas em uma rede em locais físicos diferentes. Esta é a forma

como as pessoas jogam jogos através da internet. Dessa forma, os jogadores podem se comunicar uns com os outros através de uma conexão por voz ou mensagens instantâneas dentro do jogo;

c) LAN em grupos, que são eventos em que um grupo de pessoas estão juntas em um mesmo local, mas cada jogador tem o seu próprio computador ligado aos outros por uma rede de área local (LAN). Dessa forma, eles podem falar uns com os outros, mas eles não podem ver as telas uns dos outros.

Adams (2009, p. 103) afirma que os jogos *multiplayer* evocam a mais ampla variedade de emoções, porque os jogadores se socializam com pessoas reais e fazem amigos quando jogam.

2.5.2 Desafio dos jogos *multiplayer*

Os jogos *multiplayer* possuem desafios técnicos, bem como as responsabilidades permanentes de fornecimento de novos conteúdos e serviços aos jogadores. Algumas questões técnicas devem ser tratadas diferentes para jogos *online* e jogos locais.

Dentre esses desafios, Adams (2009, p. 594-596) destaca alguns como primordiais para um bom desenvolvimento dos jogos *multiplayer*, como o modelo de comunicação, a latência, perda de pacotes de comunicação, não deixar os jogadores perderem a atração pelo jogo, controlar o mau comportamento dos jogadores dentro da comunidade do jogo e atendimento para solucionar problemas do jogo.

Os modelos de comunicação podem ser o cliente/servidor, onde cada jogador executa um programa chamado de cliente no seu computador que se comunica com um programa central, o servidor, em um computador de propriedade de uma empresa que presta o serviço do jogo ou o modelo ponto-a-ponto que envolve a comunicação direta entre os computadores dos jogadores em um mesmo ambiente.

A internet não faz quaisquer garantias sobre quanto tempo um determinado pacote de dados vai demorar para chegar de um ponto a outro. Um jogador com uma conexão mais rápida, traduz em uma vantagem de jogo, fazendo com que os jogadores com conexões de alta velocidade tenham mais chances de ganhar o jogo. Porém, esse tipo de problema, não é algo

que está nas mãos dos desenvolvedores devido a dependência de fatores externos dos jogadores.

Segundo Adams (2009, p. 595, tradução do autor) afirma: “para alguns jogadores, o jogo é uma forma de escapismo que os leva para um lugar mágico, e eles querem que ele fique mágico enquanto eles estão lá.”. É importante que nada ocorra no jogo para que essa magia se perca, porém, os jogos *online* contam com a interação de jogadores de todas as partes do mundo e com diferentes culturas, podendo surgir problemas de mau comportamento. Em um jogo *online* o jogador deve aceitar que seu mundo imaginário inclui um grande número de pessoas reais.

A produção contínua de conteúdo é primordial para a permanência dos jogadores. Os jogos *multiplayer* ganham dinheiro através de publicidade, micro pagamentos ou assinaturas. Para manter as pessoas interessadas, é necessário a produção de novos conteúdos em uma base contínua, além de prover ajuda imediata, seja para resolver problemas técnicos ou explicar a interface do usuário, responder a perguntas sobre o conteúdo do jogo e fazer valer a justiça, investigando e punindo o mau comportamento de outros jogadores.

2.5.3 Protocolos de comunicação

O protocolo para comunicação de jogos *multiplayer* costuma ser o protocolo UDP. Esse protocolo, que apesar de não ser confiável, ou seja, não garante entrega de pacotes e nem garante que cheguem na ordem correta, favorece e é usado nos jogos devido a recuperação dos dados de envio em um próximo pacote. Por exemplo, se algum pacote enviado seja perdido, a informação vai ser recuperada no próximo pacote.

Segundo Morimoto (2005), o UDP é o protocolo irmão do TCP. A diferença básica entre os dois protocolos é que o TCP é um protocolo orientado à conexão. Ele inclui vários mecanismos para iniciar e encerrar a conexão, além de negociar tamanhos de pacotes e permitir a retransmissão de pacotes corrompidos. No TCP todo esse processo é feito com muito cuidado para garantir que os dados realmente cheguem inalterados. O UDP foi feito para transmitir dados pouco sensíveis, como streaming de áudio e vídeo. No UDP não existe checagem e nem confirmação dos pacotes. Os dados são transmitidos apenas uma vez e caso

os pacotes cheguem corrompidos, são simplesmente descartados, sem que o emissor saiba do problema.

A ideia do protocolo UDP é transmitir dados focando no maior desempenho possível. Segundo Tanenbaum (2003, p. 400) esse protocolo não realiza controle de fluxo, controle de erros ou retransmissão após a recepção de um segmento incorreto. Os programas que utilizam portas UDP recorrem também à uma porta TCP para enviar as requisições de dados e também precisa ser checado periodicamente se o cliente ainda está *online* para saber se deve parar ou não de enviar os dados.

O Quadro 1 mostra uma comparação entre os protocolos TCP e UDP.

Quadro 1 – Diferenças entre os protocolos TCP e UDP

RECURSOS	TCP	UDP
Confiança/Segurança	Orientado à conexão. Entrega garantida.	Sem conexão. Entrega não garantida.
Ordenação dos pedidos	É garantida a ordem de recebimento das mensagens.	Não é garantida a ordem de recebimento das mensagens.
Peso do Protocolo	Pesado, devido à elevada informação no cabeçalho das mensagens.	Leve, devido à pouca informação no cabeçalho das mensagens.
Pacotes	Os dados são transmitidos do forma sequencial, sem distinção de início e fim do pacote. Podem existir múltiplos pacotes por chamada.	Datagramas, um pacote por uma chamada de leitura.

Fonte: Adaptado de Tanenbaum (2003).

3 MÉTODO

Este capítulo se destina a apresentar os procedimentos executados para a realização do projeto, classificando-o quanto ao tipo de pesquisa, mostrando as etapas necessárias para alcançar os objetivos propostos e as delimitações do projeto.

3.1 CARACTERIZAÇÃO DO TIPO DE PESQUISA

Segundo Gil (2008, p. 43), a pesquisa é um processo formal e sistemático de desenvolvimento do método científico, com o objetivo fundamental de descobrir respostas para os problemas, mediante o emprego de procedimentos científicos.

O tipo de pesquisa feita para a confecção desta monografia é classificada como "pesquisa aplicada". Segundo Silva e Menezes (2005, p. 20), a natureza desse tipo de pesquisa tem como objetivo "gerar conhecimentos para aplicação prática e dirigidos à solução de problemas específicos.". Este projeto se propõe a criar pacotes de códigos facilitadores para desenvolvedores de jogos que utilizam a *engine Unity 3D* para redes sociais, assim como agilizar o processo tanto de desenvolvimento como de aprendizado na integração dos jogos com as redes sociais.

Quanto aos objetivos da pesquisa, esta monografia pode ser classificada como "pesquisa exploratória", pois ela visa proporcionar um maior entendimento do problema utilizando o desenvolvimento de ideias e hipóteses, assim como Gil (2008, p. 43) conceitua, que a principal finalidade é desenvolver, esclarecer e modificar conceitos e ideias, com vista na formulação de problemas mais precisos.

3.2 ATIVIDADES METODOLÓGICAS

De acordo com Silva e Menezes (2005, p.22), a pesquisa deve obedecer aos critérios de coerência, consistência, originalidade e objetivação. O planejamento de uma pesquisa dependerá basicamente de três fases:

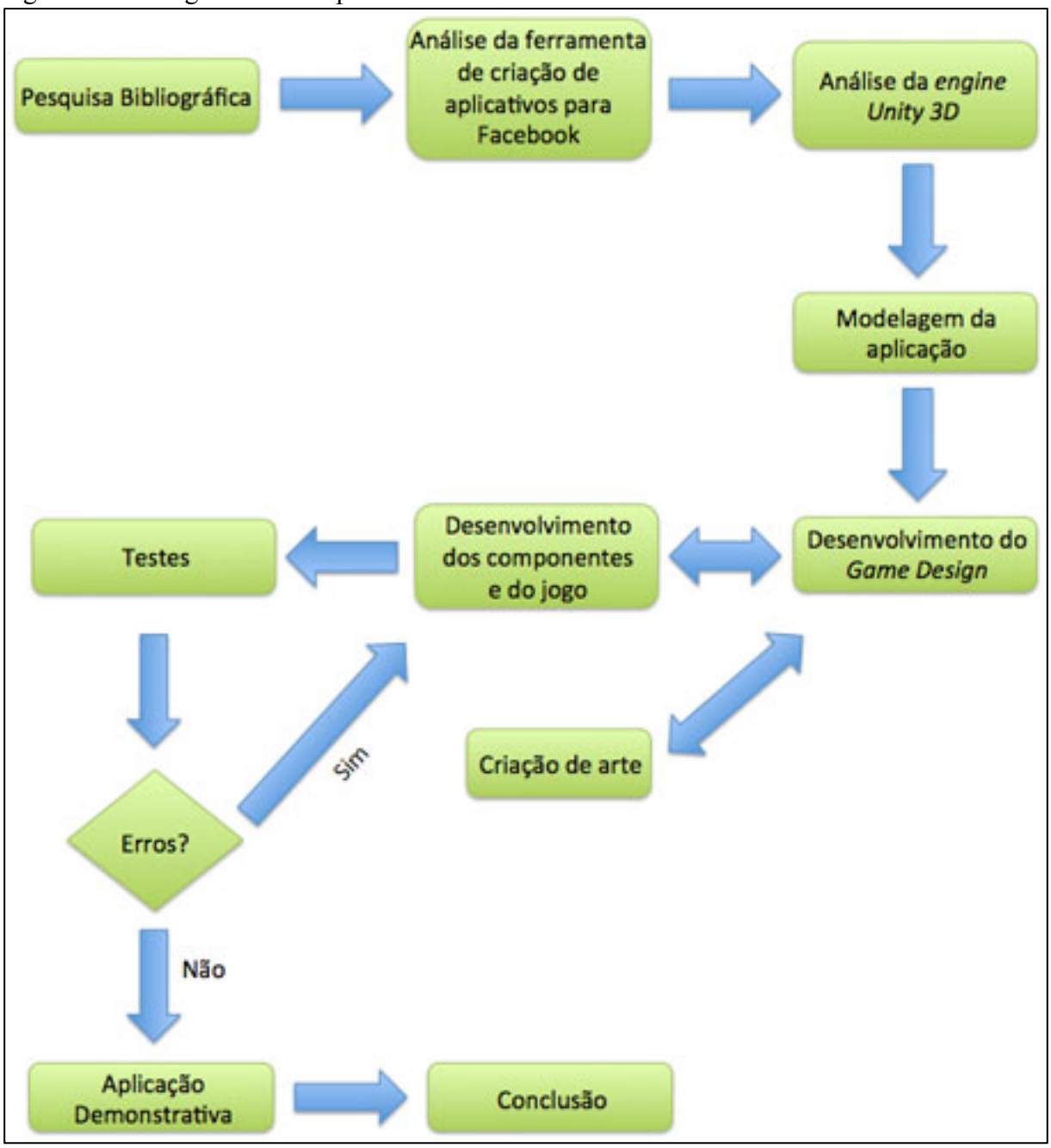
a) fase decisória: referente à escolha do tema, à definição e à delimitação do problema de pesquisa;

b) fase construtiva: referente à construção de um plano de pesquisa e à execução da pesquisa propriamente dita;

c) fase redacional: referente à análise dos dados e das informações obtidas na fase construtiva. É a organização das ideias de forma sistematizada visando à elaboração do relatório final.

O desenvolvimento do projeto está proposto em etapas apresentadas no fluxograma da Figura 7.

Figura 7 – Fluxograma das etapas de desenvolvimento do trabalho



Fonte: Elaboração do autor (2013).

As etapas apresentadas, foram desenvolvidas conforme o detalhamento a seguir:

a) a pesquisa bibliográfica está apresentada no Capítulo 2, onde foram pesquisadas informações já publicadas em livros e artigos científicos sobre redes sociais e jogos, assim como as *engines* de desenvolvimento e o modo *multiplayer* para jogos;

b) análise, estudo e testes sobre a ferramenta disponibilizada pelo Facebook para desenvolvimento de aplicativos e utilização dos recursos disponíveis dentro da rede;

c) análise, estudo e testes com a *engine* de jogos *Unity 3D* para comunicação e desempenho do jogo dentro do Facebook;

d) a modelagem da aplicação irá utilizar alguns diagramas definidos pela UML[6], com o levantamento dos requisitos funcionais e não funcionais e os casos de uso, além de conter as decisões de *design,* como a linha de arte, definição de objetos 2D e 3D e como será o ambiente do jogo;

e) A criação das regras do jogo, ou seja, o *Game Design*, para que sejam definidas a linha de arte e o tema principal do jogo. Essa etapa fica em constante comunicação com a criação de arte e o desenvolvimento dos componentes, pois, conforme alguns pontos forem sendo definidos, a implementação é imediata para poder se dar continuidade a criação tanto de arte como das regras;

f) a etapa de desenvolvimento dos componentes fica em constante comunicação com a criação do jogo, resultando em testes com um cenário real de jogo.

g) serão realizados testes de cada funcionalidade implementada;

h) quando o jogo estiver funcionando como esperado, será gerada uma *build*[7] para ser publicada no Facebook como um aplicativo em teste (modo *Sandbox*[8]).

3.3 PROPOSTA DA SOLUÇÃO

A proposta da solução, conforme a Figura 8, é um pacote, ou conjunto, de códigos em que fazem a comunicação entre as linguagens de programação PHP e Javascript com a API do Facebook, retornando dados para a *Unity 3D* interpretar e utilizar dentro do jogo. Essa comunicação também utilizará o banco de dados MySQL para buscar os dados tanto do Facebook como os dados salvos pelos jogadores dentro do jogo.

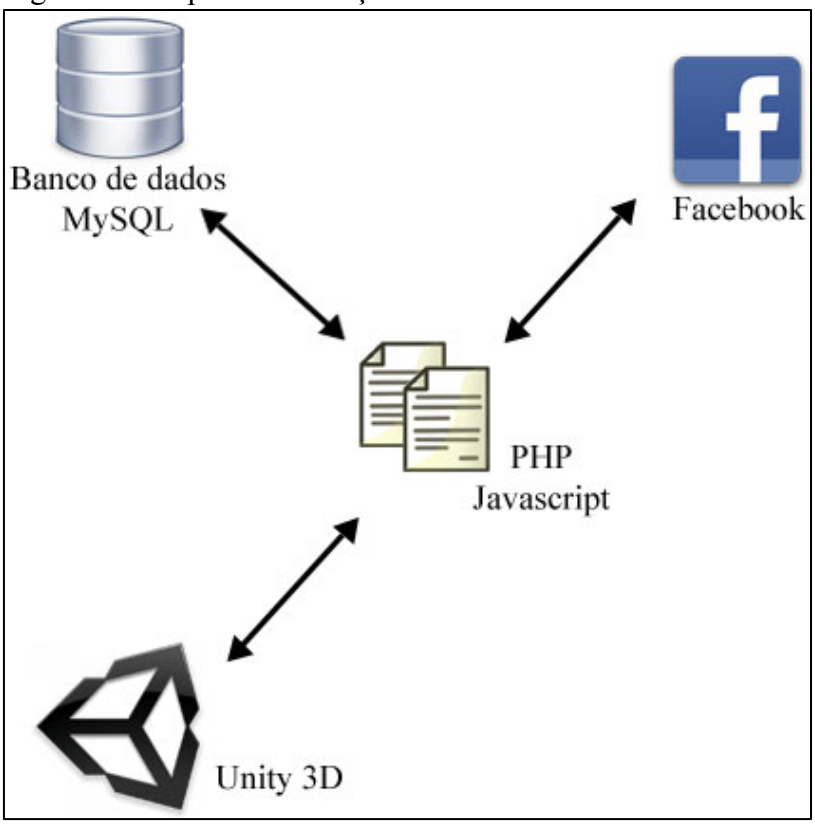
---

[6] É uma linguagem de modelagem que permite os desenvolvedores visualizem os produtos de seus trabalhos em diagramas padronizados.

[7] Versão compilada de um *software* que contém um conjunto de recursos.

[8] Ambiente de desenvolvimento *online* do qual o conteúdo pode ser testado sem afetar o sistema original.

Figura 8 – Proposta da solução



Fonte: Elaboração do autor (2013).

Ainda dentro da proposta, foram criados módulos de comunicação *multiplayer* dentro da *Unity 3D*.

3.4 DELIMITAÇÕES

A solução proposta nesta monografia tem a finalidade de gerar pacotes de códigos para que estudantes e profissionais da área de jogos e redes sociais possam obter ganhos em tempo de desenvolvimento e em conhecimento da comunicação entre os jogos e as redes sociais que irão desenvolver.

O jogo não será desenvolvido por completo, e sim um módulo com as funcionalidades básicas de um jogo, assim como os recursos mais usados em jogos dentro do Facebook. A única rede social envolvida no desenvolvimento do jogo é o Facebook.

Dentro da parte de *multiplayer*, as soluções utilizadas são as básicas para a comunicação entre cliente-servidor e cliente-cliente, do ponto de vista de não utilizar um

servidor dedicado ou outros recursos para se fazer um jogo massivo, ou seja, que possua milhares de pessoas ao mesmo tempo utilizando os recursos do servidor.

4 MODELAGEM DO SISTEMA

Neste capítulo são apresentadas as atividades de análise, estudo e testes sobre a ferramenta disponibilizada pelo Facebook e a *engine* de jogos *Unity 3D*, além de apresentar a modelagem da aplicação proposta. São expostos os requisitos funcionais e não funcionais e o diagrama de casos de uso.

4.1 FACEBOOK SDK

O Facebook SDK fornece um rico conjunto de funcionalidades para acessar a sua API do lado do servidor. Ele é utilizado para realizar as operações como administrador do aplicativo, além de oferecer todos os recursos necessários para conexão direta do aplicativo com a sua base de dados e outras funcionalidades utilizando linguagens de programação.

Segundo Graham (2008), a plataforma do Facebook é composta por cinco componentes: a linguagem de marcação[9] derivada do HTML (Facebook *Markup Language*), uma *Rest*[10] API para lidar com comunicação entre o Facebook e a sua aplicação, uma linguagem de estilo SQL para interagir com os dados do Facebook (Facebook *Query Language*), uma linguagem de *scripting* (Facebook JavaScript), e um conjunto de bibliotecas de cliente para diferentes linguagens de programação.

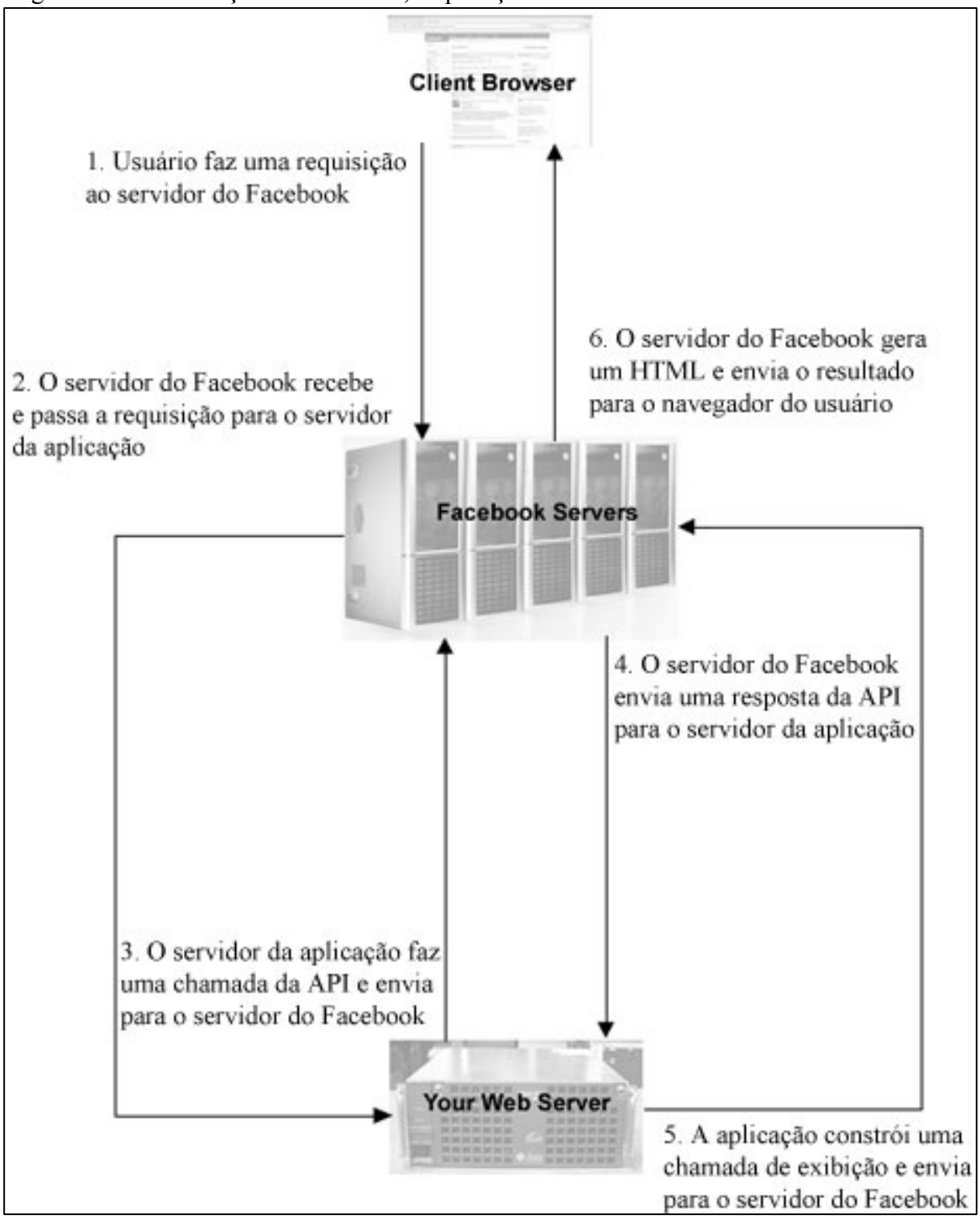
A Figura 9, mostra como é feita a comunicação entre o usuário, a aplicação e o Facebook. Isto ocorre toda vez que a aplicação requisita recursos dos servidores do Facebook.

---

[9] É um conjunto de códigos aplicados a um texto ou a dados, com o fim de adicionar informações particulares sobre esse texto ou dado, ou sobre trechos específicos.

[10] Técnica de engenharia de software para sistemas hipermídia (reunião de várias mídias num suporte computacional) que descreve qualquer interface web simples que utiliza XML e HTTP (ou JSON, ou texto puro) sem as abstrações adicionais dos protocolos baseados em padrões de trocas de mensagem.

Figura 9 – Comunicação entre usuário, a aplicação e o Facebook



Fonte: Adaptado de Graham (2008, p. 16).

Com isso, Graham (2008) orienta que é preciso considerar essas interações constantemente ao desenvolver uma aplicação, devido ao cuidado que se deve tomar para não se fazer chamadas desnecessárias para a API do Facebook, que pode gerar um problema de lentidão na aplicação.

4.1.1 Criando a aplicação no Facebook

Uma vez entendido o fluxo de comunicação entre o aplicativo e o Facebook e as principais informações da sua plataforma, é importante criar a aplicação dentro do Facebook e configurá-la.

Para criar o aplicativo, o Facebook disponibiliza uma página para desenvolvedores (https://developers.facebook.com) que contém além da possibilidade de criar aplicativos, toda a documentação necessária para aprender a utilizar sua plataforma e toda a parte de gerenciamento da aplicação.

Ao criar a aplicação, o usuário é perguntado sobre o nome que o aplicativo irá receber e o *namespace*[11] da aplicação, além de oferecer um provedor de *cloud service* (servidor de computadores dedicados a aplicação que garantem confiança e estabilidade) para facilitar desenvolvedores que não possuem um servidor definido.

Uma vez criada a aplicação no Facebook, o usuário passa a ter um painel administrador, onde possui um identificador/chave da aplicação e um segredo único em código. Nesse painel podem ser feitas várias configurações, acompanhar os dados e estatísticas do aplicativo dentro da rede e definir usuários para testar a aplicação, enquanto ela não é publicada.

A aplicação funciona dentro de uma página HTML que é carregada dentro do Facebook como um *Canvas Application* (uma tela em branco da qual executa o aplicativo), isso resulta no aplicativo estar sendo exibido dentro do padrão Facebook.

4.1.2 Facebook SDK para Javascript

Esse projeto irá utilizar o SDK do Facebook para Javascript. A ferramenta de desenvolvimento para programar os códigos em HTML e Javascript será o Eclipse SDK 3.7.

Os primeiros passos para estar com a aplicação ligada ao Facebook é criar um HTML dentro do servidor, que será o responsável pela configuração inicial da aplicação.

---

[11] Usado para acessar o aplicativo através de URL (ex: http://apps.facebook.com/namespace) e para acessar ações e objetos customizados da aplicação.

A Figura 10 mostra o código de um exemplo simples, executado com sucesso, da aplicação conectando com o Facebook e buscando o identificador do usuário, que será necessário para registrar esse jogador no banco de dados, além de garantir que o usuário está conectado no Facebook.

Figura 10 – Código de inicialização do Javascript SDK

```
<div id="fb-root"></div>

<script src="//connect.facebook.net/en_US/all.js"></script>

<script>
    // Initialize the JS SDK
    FB.init({
        appId: '1234567890,
        status: true,
        frictionlessRequests: true,
        cookie: true,
        oauth: true
    });

    FB.getLoginStatus(function(response) {
        uid = response.authResponse.userID ? response.authResponse.userID : null;
    });
</script>
```

Fonte: Elaboração do autor (2013).

Segundo Graham (2008), os métodos de API do Facebook são divididos em grupos lógicos de chamadas. Essas chamadas de API são onde ocorrem as interações com a plataforma. Não é necessário usar a biblioteca de cliente disponibilizada pelo Facebook, porém, ela torna as coisas mais rápidas a fim de interagir com a plataforma.

A Figura 11 apresenta o código de teste para buscar as informações do usuário, como nome e foto, e dos seus amigos no Facebook, com nome, sobrenome e foto. Obter os dados dos amigos do usuário é muito importante para a aplicação, pois eles podem ser usados para convidar os amigos, visualizar ranking, entre outras coisas.

Figura 11 – Código de busca dos dados do usuário

```
function UserData(Void)
{
    FB.api('/me?fields=first_name', function(response) {
        gPlayerName = response.first_name;
        imageURL = 'https://graph.facebook.com/' + uid + '/picture?width=128&height=128';
        arrayFriends += uid + "#" + gPlayerName + "#" +  imageURL + ";";
    });

    FB.api('/me/friends?fields=id,first_name,last_name', function(response) {
        if(response.data.length > 0)
        {
            for(var i = 0; i < response.data.length; i++)
            {
                gFriendID = response.data[i].id;
                gFriendName = response.data[i].first_name;
                gFriendLastName = response.data[i].last_name;
                imageFriendURL = 'https://graph.facebook.com/' + gFriendID +
                                 '/picture?width=128&height=128';
                arrayFriends += gFriendID + "#" + gFriendName + "#" +
                                gFriendLastName + "#" + imageFriendURL + ";";
            }
        }
    });
}
```

Fonte: Elaboração do autor (2013).

4.2 UNITY 3D

A *engine* de jogos *Unity 3D* é hoje uma das mais poderosas do mercado. Com ela é possível criar jogos para diferentes plataformas como PC, Mac, Web, Android, iOS, Playstation, Wii e Xbox.

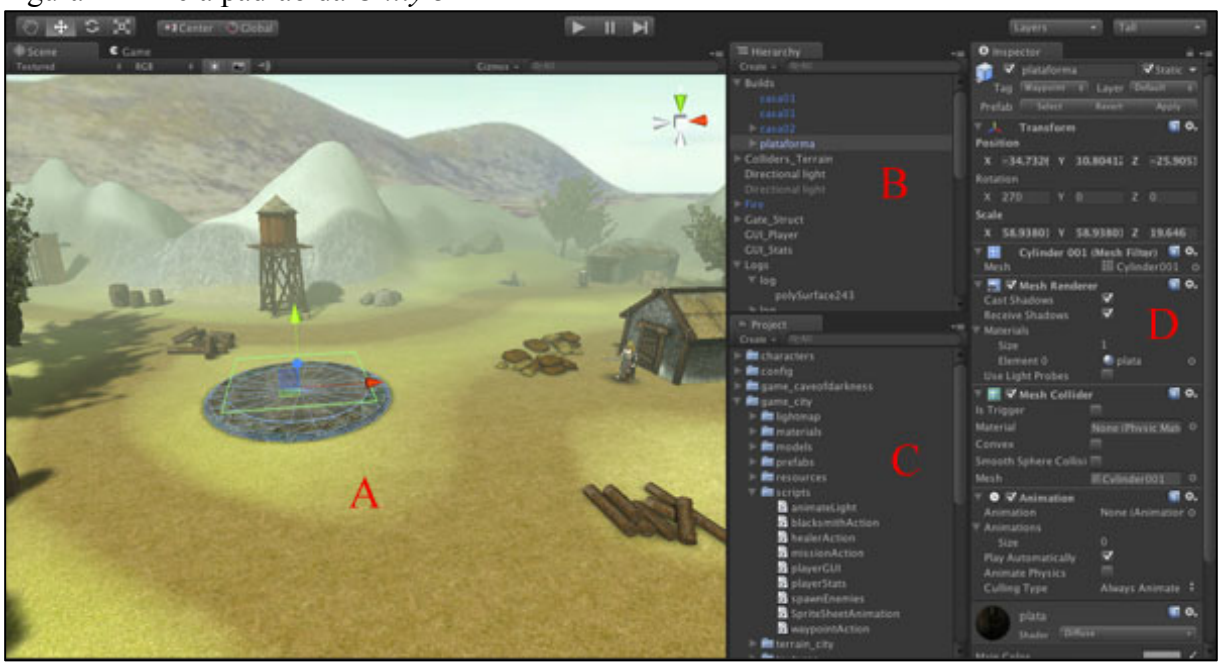
De acordo com Blackman (2011), a *Unity 3D* oferece um excelente ponto de entrada para o desenvolvimento dos jogos, equilibrando recursos e funcionalidades com um preço acessível, além de ter uma versão gratuita para iniciar o aprendizado/desenvolvimento do jogo.

Esse projeto utiliza a *engine* para gerar um jogo que rode em um navegador de internet, para poder rodar dentro do Facebook. Uma pequena desvantagem da *engine* para funcionar na *web* é a necessidade de o usuário ter que instalar um *plugin*[12] para que o jogo funcione no navegador.

[12] Programa de computador desenvolvido para adicionar funções a outros programas maiores, provendo alguma funcionalidade especial ou muito específica.

A Figura 12 mostra a estrutura da *Unity 3D* dividida em 4 aspectos básicos de visualização, onde "A" é a cena onde os objetos são colocados dentro do cenário do jogo, "B" contém a hierarquia de objetos dentro da cena, "C" possui os arquivos fonte utilizados pelo jogo, como por exemplo, os *scripts*, os modelos 3D, as imagens e os sons, e "D" é o inspetor dos elementos, para alterar as características de um determinado objeto.

Figura 12 – Tela padrão da *Unity 3D*



Fonte: Adaptado de unity3d.com (2013).

A *Unity 3D* trabalha com 3 tipos de linguagem de programação: C#, Javascript e BOO. Em um mesmo projeto, podem ser utilizadas mais de uma das 3 linguagens. Esse projeto utiliza C# e Javascript.

O primeiro teste/estudo realizado foi criar um cenário inicial do jogo para poder testar o desempenho e adicionar alguns *scripts* para conexão com o Facebook.

Com relação ao desempenho, foram testados alguns tipos de objetos 3D e luminosidade da cena. Um jogo dentro do Facebook não deve ser muito pesado, ou seja, exigir muito do processador e da placa de vídeo do computador, para poder atingir uma quantidade maior de jogadores, que possuem diversos tipos de configuração de computador. Foi identificado que se deve ter cuidado com alguns pontos:

a) quanto mais os objetos 3D são detalhados e, consequentemente, o número de polígonos é maior, o projeto vai ficando mais pesado, exigindo maior desempenho do computador.

b) a iluminação da cena também possui alguns aspectos a serem considerados. A *Unity 3D* possui um tipo de iluminação em tempo real, do qual vai se alterando conforme a posição que a câmera está em relação ao cenário. Porém, existe uma forma de "fixar" as sombras que a luz gera no cenário, utilizando alguns recursos como o *lightmapping*, que pinta, literalmente, as sombras nas texturas dos objetos que estão recebendo a sombra. Com isso, pode-se utilizar essa técnica para poder reduzir a quantidade de processamento da iluminação da cena em tempo real.

c) em relação ao tamanho em *megabytes* do jogo, a *Unity 3D* possui algumas técnicas de dividir o jogo em "pedaços", para que o jogador só precise fazer *download* do que está sendo usado no momento do jogo, e dividir o resto no *AssetBundles* (pacote de recursos).

Após esses testes, foram realizados testes de comunicação da *Unity 3D* com o Facebook. Como é um projeto para a *web*, será necessário a utilização de alguns recursos externos para garantir a comunicação entre o jogo, o banco de dados e o Facebook. Foram escolhidos o PHP e o MySQL.

Os primeiros testes de *scripts* realizados com sucesso da comunicação da *Unity 3D* com o PHP e o banco de dados MySQL, foram feitos utilizando uma classe em C# na *Unity 3D*, a WWWForm, que faz uma requisição a uma página da *web* e aguarda um resultado de sucesso ou de erro.

A comunicação entre o PHP e o MySQL ocorreu utilizando a função do PHP mysql_connect() passando o endereço do servidor, nome de usuário e senha. Foi criado um banco de dados no MySQL com algumas informações de usuário para serem feitos os testes.

O processo de comunicação entre o Facebook e a *Unity 3D*, foi feito em duas etapas:

a) um *script* C# da *Unity 3D* chama uma função utilizada apenas para *web*, *Application.ExternalCall()*, que chama uma função dentro do HTML do navegador em Javascript.

b) o Javascript recebe essa chamada e realiza a comunicação com o Facebook e retorna o resultado chamando uma função que está dentro do *script* da *Unity 3D*.

c) a *Unity 3D* recebe o resultado e processa os dados conforme necessário.

A Figura 13 mostra as fases dessa comunicação.

Figura 13 – *Scripts* de comunicação *Unity 3D* e Facebook Javascript

Unity 3D - C# script

```
void Start()
{
    //Call javascript web to inform that unity is ready
    Application.ExternalCall("UserData");
}
```

Web - Javascript

```
function UserData(Void)
{
    FB.api('/me?fields=first_name', function(response) {
        gPlayerName = response.first_name;
        imageURL = 'https://graph.facebook.com/' + uid + '/picture?width=128&height=128';
        arrayFriends += uid + "#" + gPlayerName + "#" +  imageURL + ";";
    });

    FB.api('/me/friends?fields=id,first_name,last_name', function(response) {
        if(response.data.length > 0)
        {
            var i;
            for(i = 0; i < response.data.length; i++)
            {
                gFriendID = response.data[i].id;
                gFriendName = response.data[i].first_name;
                gFriendLastName = response.data[i].last_name;
                imageFriendURL = 'https://graph.facebook.com/' + gFriendID + '/picture?width=128&height=128';
                arrayFriends += gFriendID + "#" + gFriendName + "#" + gFriendLastName + "#" + imageFriendURL + ";";

                if(i == response.data.length - 1)
                    GetFacebookFriends();
            }
        }
    });
}

function GetFacebookFriends()
{
    if(arrayFriends.length > 0)
        GetUnity().SendMessage("FacebookUI", "friendsData", arrayFriends);
}
```

Unity 3D - C# script

```
void friendsData(string friends)
{
    //Return from facebook whith friends data
    dataFacebook = friends.Split(";"[0]);
}
```

Fonte: Elaboração do autor (2013).

A conclusão obtida é que tanto a API do Facebook como a *Unity 3D* possuem uma documentação muito rica, facilitando os desenvolvedores a usarem seus recursos de forma clara e ainda possuem uma comunidade muito grande de desenvolvedores criando soluções para aplicações e ajudando os iniciantes.

Com esses primeiros testes realizados, algumas incertezas do projeto foram sanadas. A comunicação entre o Facebook e a *Unity 3D*, com o intermédio do PHP, funciona muito bem, e com poucas linhas de código, já é possível chegar à conclusão de que pode-se construir uma aplicação utilizando as duas tecnologias.

4.3 DEFINIÇÃO DE REQUISITOS

Segundo Bezerra (2008), a definição dos requisitos corresponde à etapa de compreensão do problema aplicado ao desenvolvimento do software. Existem basicamente dois tipos de requisitos, os funcionais e os não funcionais. Com isso, esta seção trata dos dois tipos de requisitos do sistema.

4.3.1 Requisitos funcionais

De acordo com Bezerra (2008), os requisitos funcionais são os que definem as funcionalidades do sistema. Os requisitos funcionais podem ser cálculos, detalhes técnicos, manipulação de dados e de processamento, entre outras funcionalidades específicas que o sistema deverá ser capaz de realizar.

A Figura 14 apresenta de forma visual os requisitos funcionais do projeto proposto.

Figura 14 – Requisitos funcionais

RF01	Cadastrar o usuário
RF02	Usuário escolher um personagem para o jogo
RF03	Salvar as informações do usuário
RF04	Criação da interface (GUI)
RF05	Convidar amigos do Facebook para o jogo
RF06	Compartilhar no Facebook etapas ocorridas no jogo
RF07	Gerenciar o balanceamento do jogo
RF08	Criar ações do jogo

RF09	Escolha de habilidades do jogador
RF10	Salvar status do jogo
RF11	Criar AI de controle
RF12	Gerar itens aleatórios para o usuário
RF13	Gerar inimigos aleatoriamente no cenário
RF14	Gerenciar nível e equipamentos do jogador
RF15	Gerenciar a comunicação entre os usuários jogando em rede

Fonte: Elaboração do autor (2013).

O Quadro 2 contém as informações quanto à descrição de cada requisito.

Quadro 2 – Descrição dos requisitos funcionais

RF01 - Cadastrar o usuário **Descrição:** Ao usuário acessar o jogo através do Facebook pela primeira vez, o sistema irá registar esse novo usuário, sincronizando seus dados com a rede social.
RF02 - Usuário escolher um personagem para o jogo **Descrição:** Na primeira vez que o usuário acessa o jogo, será permitida a escolha de um personagem. Uma vez escolhido, será possível sempre visualizar as estatísticas de evolução do jogador no jogo.
RF03 - Salvar as informações do usuário **Descrição:** O sistema sempre estará se atualizando com os dados do jogador durante o período em que o mesmo está realizando ações dentro do jogo.
RF04 - Criação da interface (GUI[13]) **Descrição:** Criação da interface contendo todos os elementos de interação do usuário com o sistema.
RF05 - Convidar amigos do Facebook para o jogo **Descrição:** O sistema permitirá que o usuário possa enviar convites para os amigos do Facebook conhecerem o jogo.
RF06 - Compartilhar no Facebook etapas ocorridas no jogo **Descrição:** O sistema deverá estar preparado para apresentar ao usuário a opção de compartilhar algum evento dentro do jogo no Facebook.
RF07 - Gerenciar o balanceamento do jogo **Descrição:** O sistema deverá cuidar do balanceamento do jogo, garantindo que o jogo não se torne fácil ou difícil demais, ou seja, balancear o surgimento de itens, as evolução das habilidades do jogador e dos inimigos.
RF08 - Criar ações do jogo **Descrição:** Criar as ações que serão a base da interatividade entre o usuário e os elementos do cenário.
RF09 - Escolha de habilidades do jogador **Descrição:** O usuário irá selecionar as habilidades do seu personagem da forma que desejar, sempre que o mesmo alcançar um novo nível no jogo.
RF10 - Salvar status do jogo

[13] É um tipo de interface para o usuário que permite a interação com dispositivos digitais, através de elementos gráficos como ícones e outros indicadores visuais.

Descrição: Salvar o status atual do jogo, permitindo ao usuário continuar o jogo em outra ocasião, sem a perda dos dados.
RF11 - Criar AI[14] de controle **Descrição:** Criar AI de controle que será encarregada de manipular e controlar todas as mudança de estados e comportamentos dos elementos dentro do jogo.
RF12 - Gerar itens aleatórios para o usuário **Descrição:** O sistema será capaz de gerar itens aleatórios para o usuário durante o jogo, garantindo que não haja favorecimento dos resultados obtidos, assim como tornar surpreendente o surgimento dos itens para o jogador.
RF13 - Gerar inimigos aleatoriamente no cenário **Descrição:** O sistema irá gerar aleatoriamente a quantidade de inimigos em pontos definidos do cenário, assim como a força e quantidade de vida deles.
RF14 - Gerenciar nível e equipamentos do jogador **Descrição:** O sistema deverá gerenciar e salvar as informações no banco de dados, toda vez que ocorrer alguma alteração de: • nível do personagem; • estatísticas do personagem; • obtenção ou perda de equipamentos;
RF15 - Gerenciar a comunicação entre os usuários jogando em rede **Descrição:** O sistema cuidará da conexão entre os jogadores que desejarem jogar juntos, proporcionando a comunicação entre as máquinas e elementos do jogo.

Fonte: Elaboração do autor (2013).

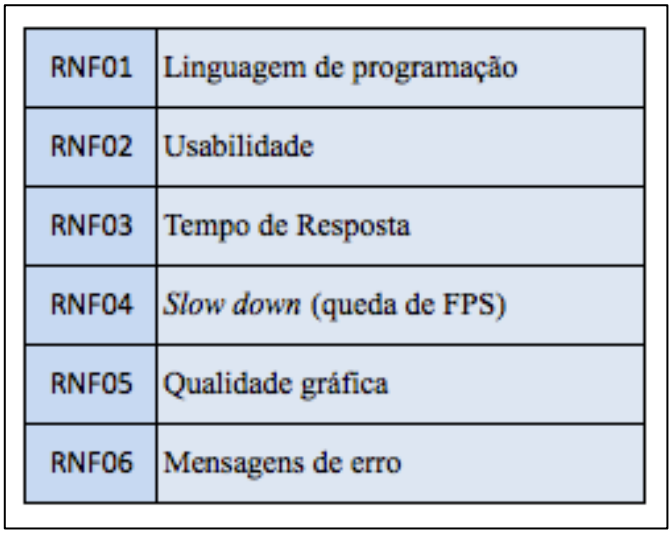
4.3.2 Requisitos não funcionais

De acordo com Bezerra (2008), os requisitos não funcionais declaram as características de qualidade que o sistema deve possuir e que estão relacionadas às suas funcionalidades, como desempenho, segurança, portabilidade, usabilidade e confiabilidade.

[14] Ramo da ciência da computação que se ocupa em buscar métodos ou dispositivos computacionais que possuam a capacidade racional do ser humano de resolver problemas.

A Figura 15 apresenta de forma visual os requisitos não funcionais do projeto proposto.

Figura 15 – Requisitos não funcionais



RNF01	Linguagem de programação
RNF02	Usabilidade
RNF03	Tempo de Resposta
RNF04	*Slow down* (queda de FPS)
RNF05	Qualidade gráfica
RNF06	Mensagens de erro

Fonte: Elaboração do autor (2013).

O Quadro 3 contém as informações quanto à descrição de cada requisito.

Quadro 3 – Descrição dos requisitos não funcionais

RNF01 - Linguagem de programação **Descrição:** O jogo será desenvolvido para atuar dentro do Facebook, utilizando: • linguagem para o navegador: HTML e Javascript; • a comunicação entre os dados do jogo e o banco de dados utilizará a linguagem PHP e o banco de dados MySQL; • o jogo será programado na linguagem C# (C *sharp*).
RNF02 - Usabilidade **Descrição:** O jogo deverá oferecer ao usuário a possibilidade de usufruir de suas funcionalidades principais de maneira rápida e fácil.
RNF03 - Tempo de Resposta **Descrição:** O tempo de resposta das requisições do jogo poderão variar conforme a velocidade de conexão do usuário, devido ao jogo ser *online.* A média aceitável é entre 5 e 10 segundos.
RNF04 - *Slow down* (queda de FPS[15])

[15] É a unidade de medida da cadência de um dispositivo audiovisual qualquer, significando o número de imagens que tal dispositivo registra, processa ou exibe por unidade de tempo.

Descrição: O jogo deverá apresentar atrasos aceitáveis ao usuário, em relação às exigências mínimas de configuração da máquina para jogar um jogo em 3D, ou seja, um computador com processador Core 2 Duo, memória de 4Gb e uma placa de vídeo NVIDIA GeForce 320M 256Mb.
RNF05 - Qualidade gráfica **Descrição:** O jogo deverá apresentar qualidade em termos gráficos, com visual agradável aos olhos do usuário.
RNF06 - Mensagens de erro **Descrição:** As mensagens de falha e sucesso devem ser sempre mostradas para o usuário, permitindo ele ter uma percepção do que está acontecendo no jogo. Assim como sempre mostrar para o usuário o motivo do erro, seja do sistema ou de conectividade com a *internet*.

Fonte: Elaboração do autor (2013).

4.4 MODELAGEM DOS CASOS DE USO

Conforme Bezerra (2002, p. 45):

> O modelo de casos de uso é uma representação das funcionalidades externamente observáveis do sistema e dos elementos externos ao sistema que interagem com ele.

Essa modelagem é composta pelos casos de uso, seus atores e os relacionamentos entre eles. Sendo assim, esse tópico trata das regras de negócio, a lista e o diagrama dos casos de uso e a definição dos atores do sistema.

4.4.1 Regras de negócio

De acordo com Bezerra (2002), as regras de negócio são de grande importância nos processos organizacionais, pois descrevem como a organização funciona. O Quadro 4 apresenta as regras de negócio do projeto proposto.

Quadro 4 – Descrição das regras de negócio

RN01 - Iniciar o jogo **Descrição:** O jogador precisa ter uma conta no Facebook para jogar.
RN02 - Permissão **Descrição:** É necessário que o jogador permita que o jogo acesse suas informações de conta do Facebook ao instalar o jogo.
RN03 - Personagem **Descrição:** O jogador precisa escolher um, entre três personagens, existentes no jogo. Cada um possui os atributos iniciais diferenciados. Um personagem possui mais força, o outro mais destreza e o último mais inteligência.
RN04 - Atributos dos personagens **Descrição:** O personagem possui 4 tipos de atributos: força, destreza, vitalidade e inteligência. Esses atributos se modificam durante o jogo conforme o jogador passa de nível, podendo escolher aleatoriamente, em um total de cinco, qualquer um dos atributos para serem somados os pontos.
RN05 - Experiência do personagem **Descrição:** A experiência do jogador aumenta conforme ele mata os inimigos. Cada inimigo possui um mínimo e máximo de experiência que é gerado para o jogador. Para alterar a experiência do jogador, é obtido um valor aleatório entre o mínimo e o máximo que o inimigo possui.
RN06 - Níveis do personagem **Descrição:** Os níveis que o jogador alcança são calculados a partir do total de experiência obtido. Inicialmente, para alcançar o segundo nível, o valor é de 200. Assim, para alcançar os próximos níveis, multiplica-se o valor atual por dois, por exemplo: 200 x 2 = 400. Depois 400 x 2 = 800. E assim por diante.
RN07 - Cálculo dos atributos **Descrição:** O peso para os cálculos do personagem são feitos conforme a sua especialidade no atributo. O personagem que usa mais força, terá peso 2 para o atributo força, 1,2 para destreza, 1 para inteligência e 1,5 para vitalidade. O personagem que usa mais destreza, terá peso 2 para o atributo destreza, 1,5 para inteligência, 1 para força e 1,2 para vitalidade. E o personagem que usa mais inteligência, terá peso 2 para o atributo inteligência, 1,5 para destreza, 1,2 para força e 1 para vitalidade.
RN08 - Cálculo de dano realizado pelo jogador

Descrição: A quantidade de dano que o personagem do jogador possui, varia de acordo com os seus atributos. Através do cálculo:

Dano = (valor do atributo de força x (peso) + valor do atributo de destreza x (peso) + valor do atributo de inteligência x (peso)) / 3

- onde o peso é diferente para cada tipo de personagem

Com isso, se obtém o máximo de dano que o ataque irá gerar. Porém, com o valor obtido do cálculo, é feito uma fórmula para pegar um valor aleatório desse resultado, que será definida da seguinte forma: o mínimo de dano será o máximo multiplicado por 0,6. Assim, é obtido um valor aleatório entre o mínimo e o máximo.

RN09 - Dano sofrido pelo jogador

Descrição: O cálculo de resistência do jogador aos ataques dos inimigos é feito com base nas estatísticas que ele possui. O cálculo de defesa do jogador é:

Defesa = (valor do atributo de força x (peso) + valor do atributo de destreza x (peso) + valor do atributo de vitalidade x (peso)) / 3

- onde o peso é diferente para cada tipo de personagem

O dano gerado pelo inimigo é obtido de um valor aleatório entre o máximo de dano do inimigo e o mínimo (calculado pelo máximo x 0,6).

Sendo assim, o cálculo do dano sofrido pelo jogador é:

Dano sofrido = dano gerado pelo inimigo - (defesa do jogador x 0.01)

Se o resultado for menor que 1, o valor do dano será o original dado pelo inimigo dividido por 2.

RN10 - Itens do jogo

Descrição: Os itens que o jogador pode obter durante a partida são: dinheiro, vida, mana (poção mágica), armadura, elmo (capacete), escudo, bota, ombreira, arma, anel e amuleto.

RN11 - Geração dos itens durante o jogo

Descrição: Os itens gerados para o jogador durante a partida, ocorrem de forma aleatória quando qualquer baú é aberto ou inimigo é morto. Eles são gerados obedecendo uma porcentagem para balancear a frequência com que eles surgem. As porcentagens são:

- dinheiro = 20%;
- mana = 19%;
- vida = 16%;
- arma = 13%;
- escudo = 10%;

• armadura = 9%; • anel = 8%; • amuleto = 5%; O dinheiro, ao ser sorteado, também precisa ser sorteado de forma aleatória qual o seu valor. Esse valor é calculado a partir da seguinte fórmula: $$\text{Dinheiro} = \text{aleatório entre } [50 \times (1.2)^{\text{nível} - 1} \text{ e } 200 \times (1.2)^{\text{nível} - 1}];$$
RN12 - Inimigos **Descrição:** O mapa possui áreas determinadas para que os inimigos possam surgir. A quantidade de inimigos que surgem nessas áreas acontece de forma aleatória entre o mínimo e o máximo estabelecido para a área.
RN13 - Danos e vida dos inimigos **Descrição:** Os inimigos possuem características de ataque e quantidade de vida particulares para cada um deles. O dano de ataque deles é calculado de forma a pegar um valor aleatório entre o mínimo e o máximo de dano que ele pode causar. A quantidade de vida também é um número aleatório entre um máximo e um mínimo que ele pode ter.
RN14 - Morte do personagem **Descrição:** Quando o personagem morre, a experiência e o dinheiro dele diminuem segundo um cálculo baseado no nível atual: Experiência = experiência atual - (experiência atual x 0.03) Dinheiro = dinheiro atual - (dinheiro atual x 0.03) A vida e a mana do jogador voltam para 100% e ele renasce no início do mapa, porém, o estado do jogo não se altera.
RN15 - Fim do jogo **Descrição:** O jogo não possui um *game over* (fim da partida). Ao personagem morrer durante a partida, ele retorna ao ponto inicial do mapa do jogo. O jogo também não possui um final, se o jogador terminar de cumprir suas missões, ele pode continuar jogando com os amigos e refazendo as missões, porém os níveis de dificuldade aumentam.
RN16 - Servidor **Descrição:** O servidor, ou *server-side*, será o responsável por controlar todas as ações do jogo que dependem da geração de valores aleatórios e dados contido no banco de dados.

Fonte: Elaboração do autor (2013).

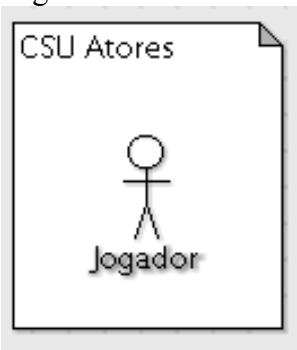
4.4.2 Casos de uso

De acordo com Fowler (2005), os casos de uso são uma técnica para captar os requisitos funcionais de um sistema, servindo como uma descrição das interações entre os usuários do sistema e o próprio sistema, fornecendo uma narrativa de como o sistema será utilizado.

Os componentes do modelo de casos de uso são: os casos de uso (funcionalidades do sistema), os atores (interagem com o sistema) e os relacionamentos entre eles, definindo como utilizar uma funcionalidade do sistema, sem entrar em detalhes internos de estrutura ou implementação.

Na terminologia da UML, qualquer elemento externo que interage com o sistema é denominado ator. Segundo Bezerra (2002), "externo" indica que os atores não fazem parte do sistema e "interagir" significa que o ator troca informações com o sistema. O sistema contém diversos casos de uso e pode conter mais de um ator. A seguir, serão apresentados o ator e os casos de uso do sistema proposto.

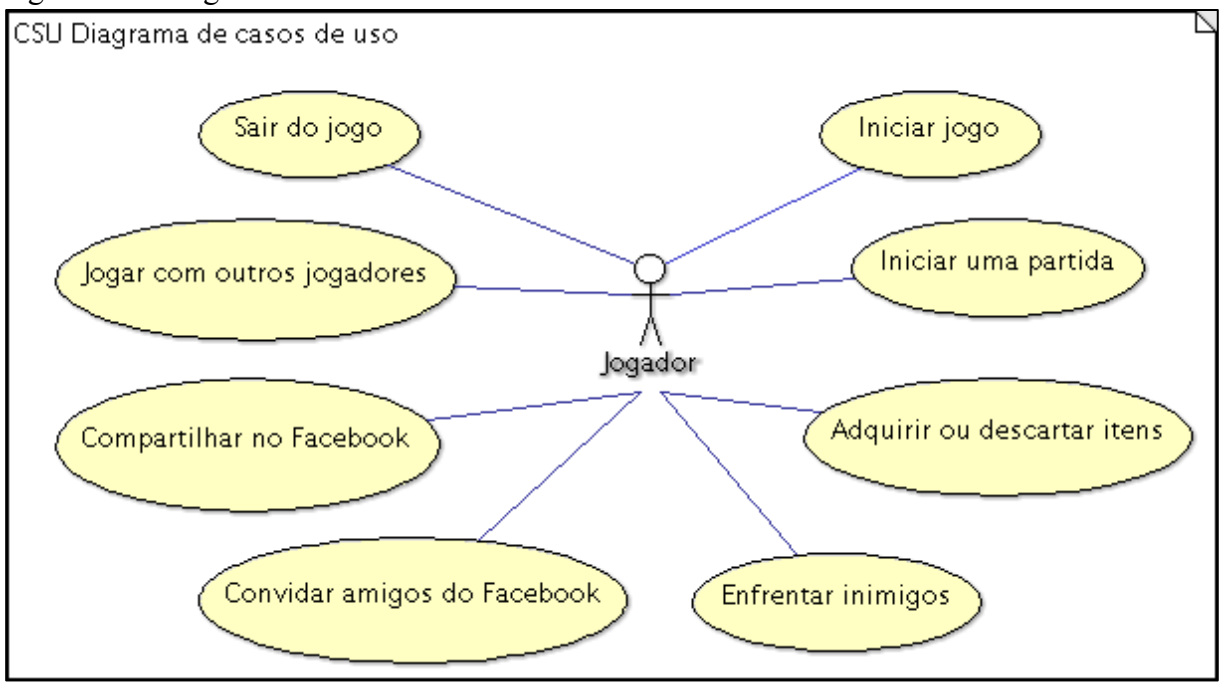
Figura 16 – Atores do sistema



Fonte: Elaboração do autor (2013).

A Figura 17 mostra o diagrama que contém a representação gráfica dos casos de uso mais importantes, descritos no Quadro 5.

Figura 17 – Diagrama de Casos de Uso



Fonte: Elaboração do autor (2013).

Quadro 5 – Casos de uso

Referência: CSU01 **Nome do Caso de Uso:** Iniciar jogo **Ator:** Jogador **Descrição:** O jogador entra no jogo e, se não possuir um personagem, seleciona um novo e então se inicia o jogo. **Finalidade:** Iniciar um jogo para o usuário.
Referência: CSU02 **Nome do Caso de Uso:** Iniciar uma partida **Ator:** Jogador **Descrição:** O jogador aparece no mapa, onde pode executar ações diversas da sua escolha. **Finalidade:** Dar liberdade para o jogador interagir com o jogo.
Referência: CSU03 **Nome do Caso de Uso:** Adquirir ou descartar itens **Ator:** Jogador **Descrição:** O jogador pode adquirir itens que estão no cenário ou descartar aqueles que estão no seu inventário. Qualquer uma das ações fazem com que os itens se atualizem no seu

inventário. **Finalidade:** Permitir ao jogador a possiblidade de obter itens novos e se desfazer de itens que não lhe interessam.
Referência: CSU04 **Nome do Caso de Uso:** Enfrentar inimigos **Ator:** Jogador **Descrição:** O jogador pode se deparar com um ou mais inimigos no seu caminho, gerando um combate. Ele pode fugir ou atacar esse inimigo, que também tentará atacá-lo. **Finalidade:** Combate entre os personagens do jogo.
Referência: CSU05 **Nome do Caso de Uso:** Convidar amigos do Facebook **Ator:** Jogador **Descrição:** O jogador, através do botão de convidar amigos, pode escolher, em uma lista, quem ele quer convidar para instalar o jogo no Facebook. **Finalidade:** Permitir ao jogador a possiblidade de convidar seus amigos no Facebook.
Referência: CSU06 **Nome do Caso de Uso:** Compartilhar no Facebook **Ator:** Jogador **Descrição:** Em determinados momentos do jogo, o jogador pode aceitar compartilhar algum conteúdo do jogo no Facebook, ainda podendo escrever algo sobre esse momento para aparecer na sua lista de notícias do seu perfil. **Finalidade:** Compartilhar experiências vividas pelo usuário no jogo.
Referência: CSU07 **Nome do Caso de Uso:** Jogar com outros jogadores **Ator:** Jogador **Descrição:** O jogador pode convidar um ou mais amigos para se conectarem e jogarem juntos. Uma vez estabelecida a conexão, os dois interagem juntos no mapa. **Finalidade:** Tornar a partida um *multiplayer*.
Referência: CSU08 **Nome do Caso de Uso:** Sair do jogo

Ator: Jogador
Descrição: O jogador deseja sair do jogo e simplesmente fecha a janela do navegador ou vai para outro endereço da *web*.
Finalidade: Permitir o usuário sair do jogo.

Fonte: Elaboração do autor (2013).

4.5 MODELAGEM DO BANCO DE DADOS

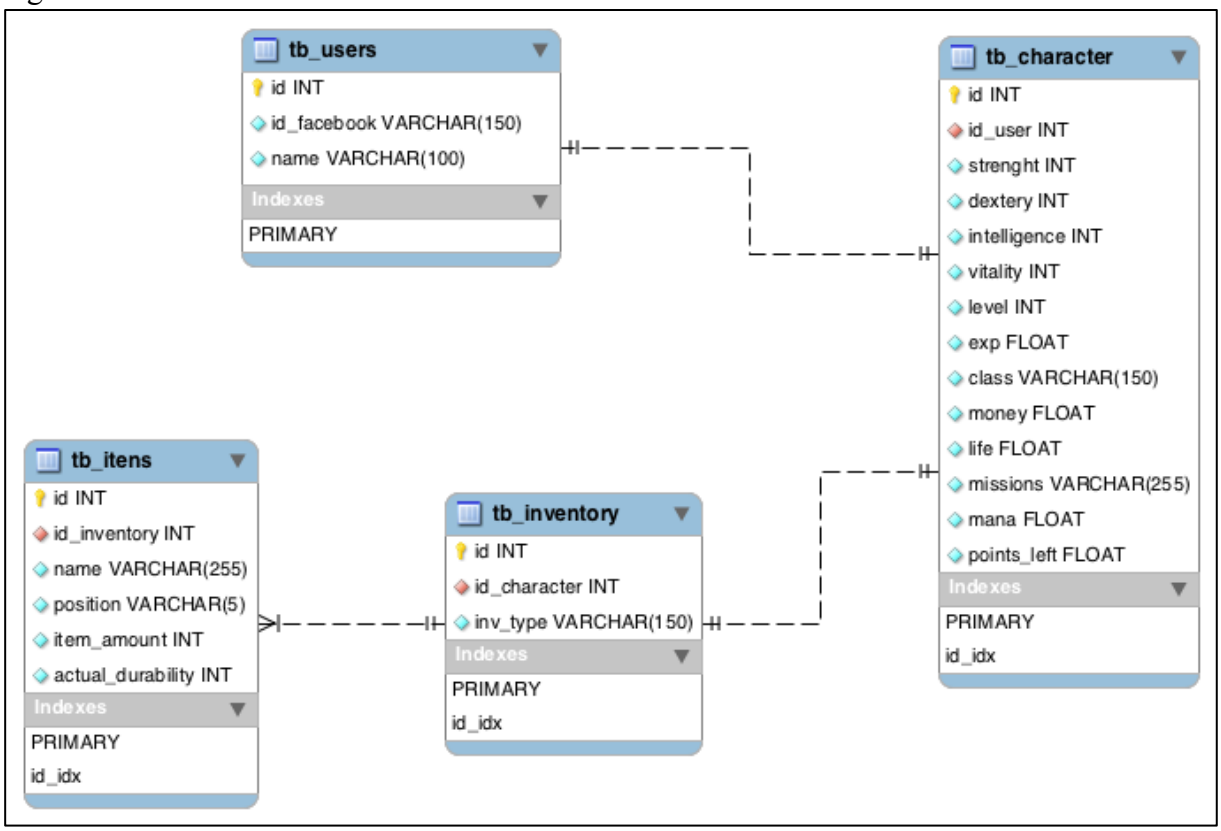
De acordo com Elmasri e Navathe (2011), um banco de dados é um conjunto de dados relacionados que podem ser registrados e possuem um significado implícito. Esse dado é derivado de alguma fonte, sendo de qualquer tamanho e complexidade.

Elmasri e Navathe (2011), afirmam que um modelo relacional de banco de dados representa uma coleção de relações que se baseiam no princípio de que todos os dados estão guardados em tabelas.

O banco de dados do jogo possui 4 tabelas que se relacionam entre si. Uma tabela que é utilizada para guardar as informações do Facebook do jogador ("*tb_users*"), uma tabela que possui as informações dos personagens que o jogador possui ("*tb_character*"), uma tabela com o inventário de itens do personagem escolhido pelo jogador ("*tb_inventory*") e uma tabela de itens contidos no inventário do jogador ("*tb_itens*").

A Figura 18 apresenta a modelagem do banco de dados relacional.

Figura 18 – Modelo relacional do banco de dados



Fonte: Elaboração do autor (2013).

A tabela do personagem contém uma chave estrangeira (*foreign key*[16]) para o id do usuário, com uma ligação 1:1, da qual significa que um usuário pode ter um personagem salvo na base de dados. Essa tabela contém todas as informações do personagem, como a força, destreza, inteligência, vitalidade, o nível, a experiência, a que classe o personagem pertence, o quanto de dinheiro ele possui, quantidade de vida, as missões concluídas, quantidade de mana disponível e os pontos que restam para poder passar de nível.

A tabela do personagem, se liga com a tabela do inventário, em uma ligação 1:1, o qual significa que cada personagem possui apenas um inventário para guardar seus itens. O inventário possui uma ligação de 1:N, onde cada inventário pode ter vários itens.

A tabela de itens é responsável apenas por guardar informações de nome do item, a posição que ele ocupa no inventário, se esse item possui mais de um na mesma posição (*item_amount*), por exemplo, um item como a poção de vida pode estar agrupada ocupando apenas uma posição no inventário, e por final a durabilidade do item.

[16] É um atributo de uma tabela do banco de dados, cujo valor é necessário para equivaler à chave primária de uma relação entre tabelas.

As características dos itens ficam separadas em um arquivo de texto, no formato JSON[17], contendo o nome do item, sua descrição, preço para venda ou compra, o mínimo de força, destreza, inteligência e vitalidade que o personagem precisa ter para poder usá-lo, o quanto de ganho em força, destreza, inteligência e vitalidade que o personagem obtém ao utilizar esse item, a sua durabilidade, se ele é um item que pode ser colocado vários em uma mesma posição do inventário e a que parte do corpo do personagem ele se relaciona.

[17] É um arquivo em formato texto e completamente independente de linguagem, pois usa convenções que são familiares à varias linguagens de programação, sendo um formato ideal para troca de dados.

5 DESENVOLVIMENTO

Neste capítulo, serão apresentados detalhes sobre o desenvolvimento do projeto: um relato histórico, as tecnologias e ferramentas utilizadas, a criação do *game design* e da arte, o desenvolvimento dos componentes do projeto, testes, a apresentação e validação do sistema.

5.1 RELATO HISTÓRICO

A ideia da criação dos componentes para o desenvolvimento de jogos utilizando a *Unity 3D* como *engine* e a sua conexão com a rede social do Facebook, surge de experiências vividas profissionalmente e em discussões com profissionais da área sobre o reaproveitamento de códigos e estruturas/arquiteturas na criação desses jogos.

Uma grande parte dos desenvolvedores *Indie*[18] e empresas de pequeno porte, focam o projeto do jogo em chegar ao resultado de forma menos custosa e em menos tempo, devido a prazos de entrega e alocação dos profissionais nos projetos. Por isso, para que se possa ganhar tempo e ter o objetivo alcançado, esses componentes especializados em determinadas funções do jogo, são desenvolvidos voltados para o jogo em questão, e não de forma genérica, do qual podem ser utilizados em vários outros projetos, ocasionando uma falta de otimização do desenvolvimento.

Uma vez criados esses componentes genéricos e a constante utilização deles nos projetos seguintes, irá fazer com que em pouco tempo de desenvolvimento eles se tornem cada vez mais completos e especializados, permitindo um maior tempo gasto com outros elementos mais específicos de cada projeto.

Foram listadas algumas funcionalidades que normalmente são utilizadas em jogos eletrônicos, das quais os desenvolvedores, geralmente, perdem bastante tempo codificando, como, por exemplo, mecânicas de movimentação do personagem controlado pelo jogador e dos outros personagens envolvidos no jogo, geração dos níveis, experiência e atributos do jogador, criação de itens e seus atributos, sistema de inventário dos itens, atributos e

[18] Jogos criados por uma pessoa ou equipe sem apoio financeiro, ou seja, um jogo independente.

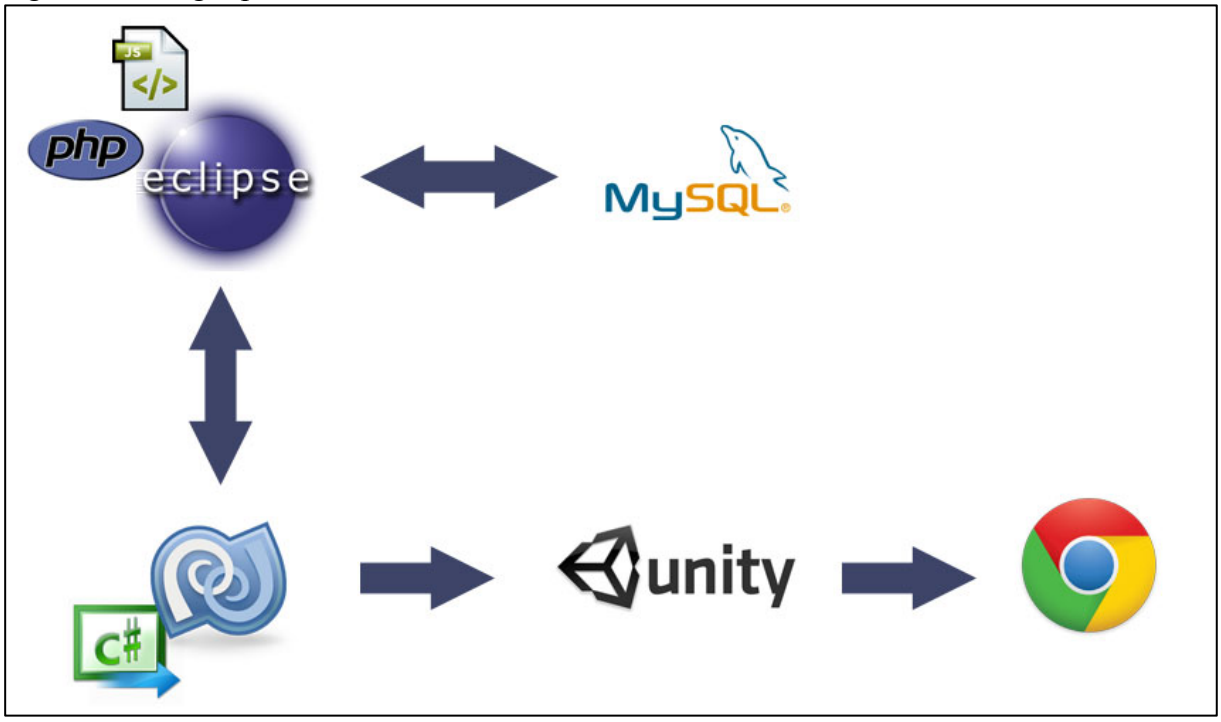
inteligência artificial dos inimigos, conexão *multiplayer* entre dois ou mais jogadores e uma base de dados básica para suportar essas funcionalidades.

5.2 TECNOLOGIAS E FERRAMENTAS

Esta seção apresenta conceitos básicos sobre as principais tecnologias utilizadas no desenvolvimento dos componentes, são elas: a *engine* de jogos *Unity 3D*, MonoDevelop para desenvolvimento na linguagem C#, o Eclipse para desenvolver em PHP e Javascript os scripts que ficam no servidor e o PHPMyAdmin para gerenciamento do banco de dados MySQL. O navegador de internet Google Chrome foi utilizado para interação do usuário com o jogo gerado pela *engine*.

A Figura 19 ilustra a utilização de todas as ferramentas.

Figura 19 – Logotipos das ferramentas utilizadas



Fonte: Elaboração do autor (2013).

5.2.1 *Engine* de jogos *Unity 3D*

A *Unity 3D* é uma *engine* com muitas funcionalidades para a criação de jogos, tanto 3D como em 2D. Ela foi adotada para ser utilizada nesse projeto devido a grande quantidade de desenvolvedores que a utilizam, além de uma vasta documentação e suporte.

A *engine* possui um grande número de características importantes que permitem a criação dos jogos, como a integração com modelos 3D de diversos tipos de *softwares*, como Maya, 3DS e Blender, edição de terrenos, criação de *shaders*[19], suporte a física do jogo, animação de objetos, criação de *prefabs* (objeto com vários componentes dentro, prontos para serem colocados no cenário, sem precisar de qualquer alteração) e uma interface bem planejada, que facilita o desenvolvimento e entendimento de todas as suas funções.

Por ser uma *engine* que interage com modelos 3D, ela exige que o computador tenha capacidade de processar tarefas que exigem mais de um computador que o comum, e também é interessante que o mesmo possua uma placa de vídeo 3D. Esse projeto foi desenvolvido na versão da *Unity 3D* para o sistema operacional Mac OS, porém, o projeto pode ser utilizado no Windows sem necessidade de qualquer alteração em código ou estrutura.

5.2.2 Monodevelop

O Monodevelop é um ambiente de desenvolvimento integrado (IDE) de código aberto, criado em 2003, que suporta as linguagens de programação C#, Javascript, Python, C/C++, Boo, entre outros. Possui versão para a plataforma Linux e os sistemas operacionais Windows e Mac OS. Devido ao projeto ter sido desenvolvido no sistema operacional Mac OS e o código ser na linguagem de programação C#, essa ferramenta foi adotada por sua integração com a *engine*, o C# e o Mac OS.

---

[19] Conjunto de instruções que definem o comportamento da superfície dos objetos na tela, ou seja, aplicam efeitos como reflexos do ambiente na lataria de um carro ou a movimentação da água enquanto um personagem está nadando.

O programa possui características de *autocomplete* (completa uma palavra digitada no código de acordo com a linguagem que está sendo usada), estrutura do projeto, depuração de código, além de ser integrado com a *Unity 3D*.

5.2.3 Eclipse

O Eclipse é uma IDE de código aberto para desenvolvimento de software na linguagem de programação Java e possui uma comunidade de desenvolvedores bastante ativa. Foi adotada nesse projeto devido ao suporte dentro de um mesmo projeto de diferentes linguagens de programação. Ela possui vários *plugins* que possibilitam se desenvolver em outras linguagens de programação como PHP, Javascript, C/C++, Python, entre outras.

Além de suporte para Windows, Mac OS e Linux, possui algumas características importantes como estrutura de projeto, depuração de código, refatoração do código, que é uma forma organizada de reestruturar o código para ser melhorado, *WorkBench* (que funciona como suporte e recursos de interface gráfica) e o *Update* que gerencia as atualizações do *software* e dos *plugins*.

5.2.4 PHPMyAdmin

O PHPMyAdmin é um aplicativo *web* desenvolvido em PHP para administrar o banco de dados MySQL na internet. Por ser um dos administradores de banco de dados mais utilizados em servidores, ele foi adotado nesse projeto para facilitar a integração do banco com o servidor quando a aplicação for colocada em um servidor *web*.

Possui como principais características a importação e exportação de dados, administrar vários servidores, gerenciar os usuários do banco e seus privilégios, interface *web* e a gestão de dados do MySQL.

5.3 *GAME DESIGN* E ARTE

Esta sessão apresenta a definição do game design e a linha de arte que será criada para a utilização no jogo.

5.3.1 Definição do *Game Design*

O estilo do jogo proposto foi do tipo 3ª pessoa, onde o jogador visualiza o personagem inteiro na tela, com a câmera a uma altura que permita uma visão ampla do cenário e um ângulo de 45º, seguindo o personagem por onde ele for.

Os comandos que o jogador pode realizar dentro do jogo foram definidos como *point and click*, onde o jogador clica em algum lugar do cenário e recebe uma resposta para essa ação, como, por exemplo, mover o personagem ou coletar um item. Além disso, o jogador pode utilizar o teclado para outras ações: utilizando a tecla "*shift*" do teclado pressionada, o personagem aumenta sua velocidade de movimentação, a tecla "i" mostra o seu inventário de itens, a tecla "s" mostra a tela de estatísticas e a tecla "d" evita qualquer movimentação do personagem no cenário.

O surgimento dos inimigos no cenário acontece em pontos estratégicos do cenário, onde a quantidade de inimigos deve ser de forma aleatória, assim como os itens que são gerados por eles quando mortos. Suas estatísticas de força e quantidade de vida também devem ser aleatórias, respeitando os valores mínimos e máximos definidos para eles.

No cenário foram definidos pontos estratégicos que possuem os baús de itens. Esses itens são gerados aleatoriamente respeitando uma ordem de prioridade, definidas nos requisitos funcionais. O inventário de itens do jogador pode conter até 20 itens, sendo que os itens de vida e mana devem ser agrupados ocupando apenas um espaço no inventário.

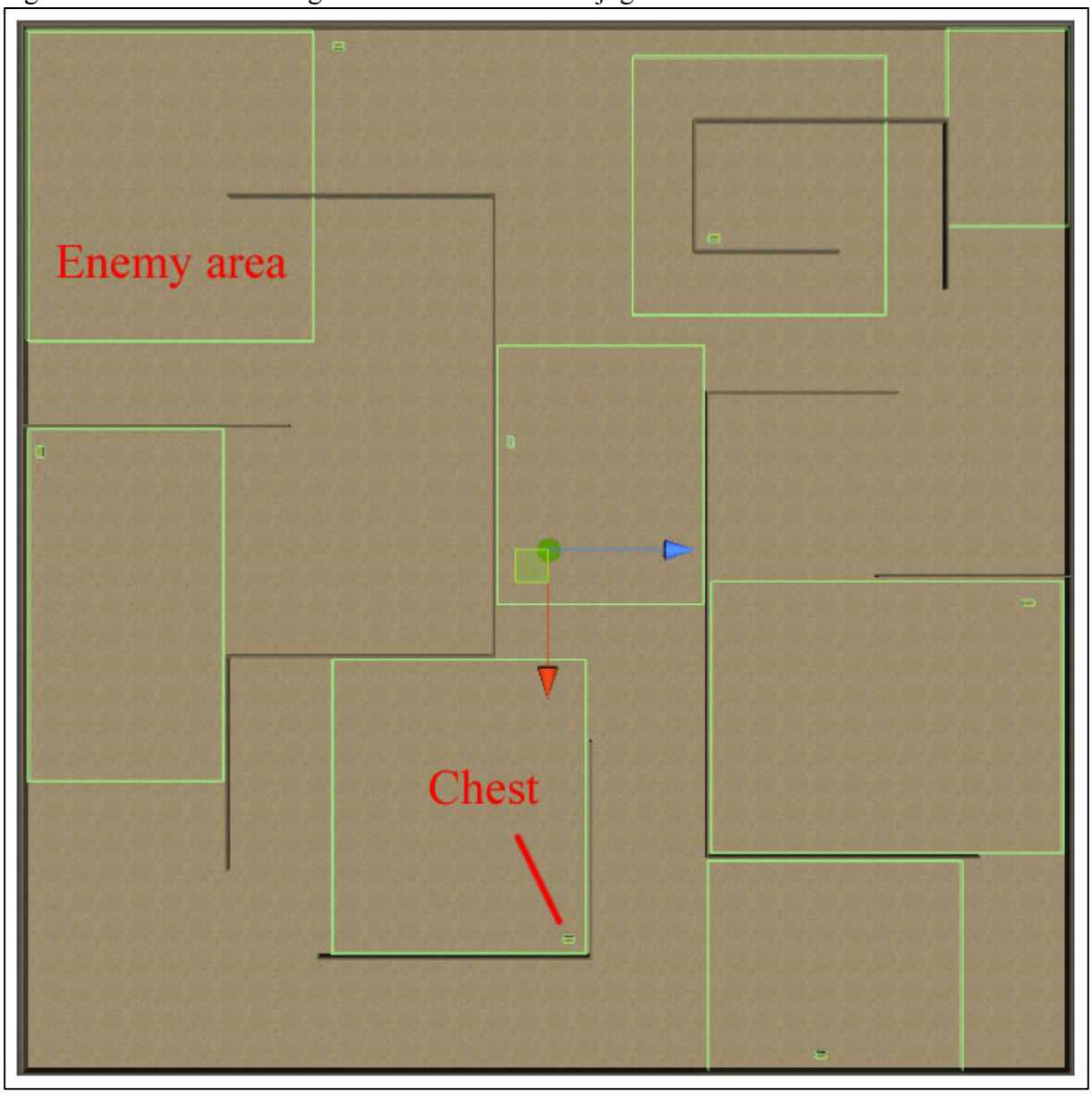
Em um jogo *multiplayer*, os jogadores que entrarem no jogo devem encontrar o cenário exatamente como está para o jogador que iniciou a partida e também pode conversar através de um *chat*. Além disso, o ganho de experiência deve ser compartilhado entre os jogadores, assim, o jogador que deu o golpe final em um inimigo ganha a experiência completa gerada e os outros participantes ganham 70% dessa experiência.

A opção do jogador de convidar amigos do Facebook para conhecerem o jogo fica visível durante todo o jogo, podendo ser acessada a qualquer momento. Para compartilhar suas experiências durante a partida, foi definido que o jogador pode realizar essa ação toda vez que alcançar um nono nível na partida.

Todos os dados do jogador ficam salvos no banco de dados para que, ao sair do jogo e voltar em um outro momento, seu personagem, estatísticas e itens permaneçam os mesmos desde a última vez que jogou.

A Figura 20 mostra como ficou estruturado o cenário, sendo vista de cima, com os pontos onde os inimigos podem surgir e os baús de itens.

Figura 20 – Áreas de inimigos e baús no cenário do jogo



Fonte: Elaboração do autor (2013).

5.3.2 Linha de arte

A linha de arte do jogo foi definida utilizando formas e cores que os jogadores já estão acostumados a distinguir conforme as suas funcionalidades. A arte pode ser dividida em elementos de GUI e elementos dentro de jogo.

As artes usadas para os componentes da GUI foram feitas em 2D. Elas aparecem na visualização da barra de *status* (estado) do jogador, com a experiência, vida, mana, nível e dinheiro, e no inventário e estatísticas do jogador.

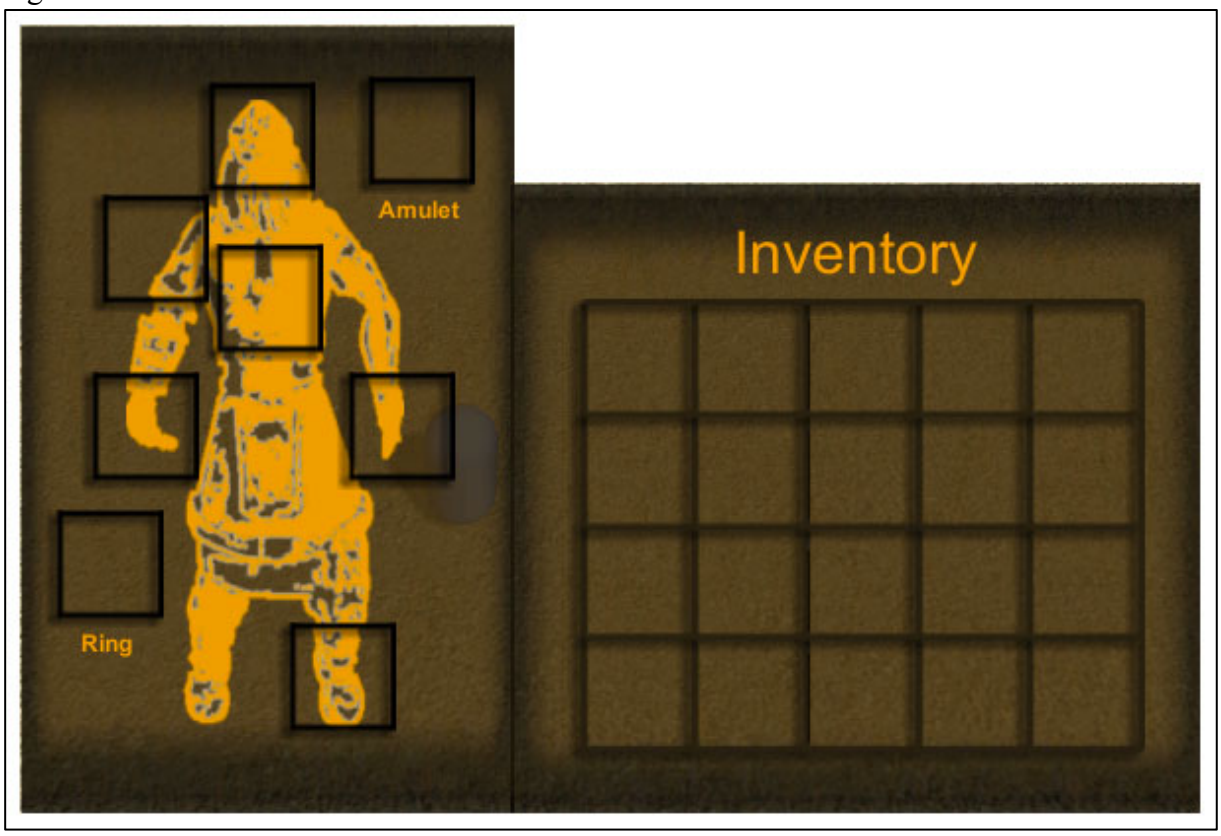
Na barra de *status* a vida e a mana foram feitas com um círculo transparente, com um preenchimento em vermelho para a vida e azul para a mana. Esse preenchimento aumenta ou diminui em tempo real conforme a quantidade que o jogador possui. A barra de experiência também aumenta e diminui em tempo real conforme o cálculo de experiência para atingir um próximo nível. Nela foi usada uma imagem do tamanho real da barra, que é redimensionada em relação aos cálculos. Ainda na barra de experiência, com o mouse colocado sobre a barra, aparece para o jogador uma imagem da mesma cor de fundo da barra de *status*, para manter a referência da GUI, com os valores atuais de experiência e quanto falta para atingir um novo nível. O dinheiro e o nível foram colocados apenas como texto. A Figura 21 mostra a barra de *status*.

Figura 21 – Arte da barra de *status*

Fonte: Elaboração do autor (2013).

As imagens de fundo da GUI mantiveram o mesmo padrão de cores. Assim, nas imagens das estatísticas e do inventário, essa relação de cores torna fácil a percepção do jogador para elementos da GUI. No inventário, foi feita uma silhueta de um corpo humano, para facilitar o jogador identificar onde cada item deve ser colocado, e na parte dos itens, possui uma grade para orientar o jogador de onde ficam os itens. A Figura 22 mostra o inventário do jogador.

Figura 22 – Arte do inventário

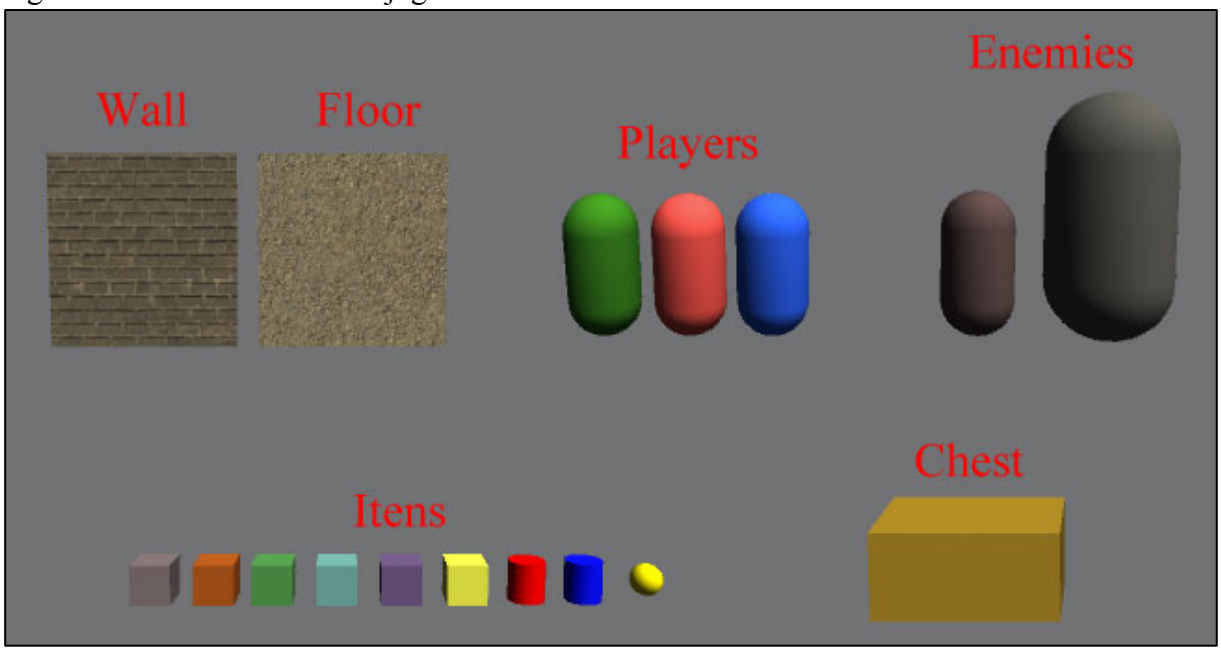


Fonte: Elaboração do autor (2013).

Os botões de interação para conectar e desconectar de um jogo *multiplayer* e de convidar amigos, assim como a tela de *chat*, foram usados componentes padrões da *Unity 3D*. Todas as telas da GUI possuem uma quantidade de transparência em 80%, uma vez que, algumas dessas telas cobrem uma grande parte da tela, fazendo com que o jogador não consiga visualizar o que está acontecendo no jogo.

Os elementos dentro do jogo, que utilizam arte 3D, foram criados utilizando objetos primitivos da *engine*. Assim, os personagens são cápsulas diferenciadas por cores, onde o jogador poder ser azul, verde ou vermelho, conforme a escolha da sua classe. Os itens são cubos, diferenciados os seus tipos por cores. A vida e a mana são cilindros, respeitando as suas cores na GUI, e o dinheiro é um círculo amarelo. Os inimigos são marrons, diferenciando os chefes deles por tamanho e a cor cinza. Os baús são cubos beges. Os muros são cubos e o chão é um plano, os dois possuem imagens inseridas a eles. A Figura 23 mostra todos os elementos de jogo.

Figura 23 – Elementos 3D do jogo



Fonte: Elaboração do autor (2013).

5.4 COMPONENTES

Esta sessão apresenta os componentes criados para serem utilizados no desenvolvimento de jogos dentro da *engine Unity 3D*, voltados para *web* e integrados com a rede social do Facebook.

Os componentes foram divididos em 5 áreas de atuação dentro do jogo: *player* (jogador), *game design* (contém as regras do jogo), conexões externas à *engine* (banco de dados e Facebook), *multiplayer* e *server-side* (*scripts* com a lógica e a persistência dos dados no servidor). Os componentes serão descritos no decorrer desta sessão.

5.4.1 As camadas do projeto

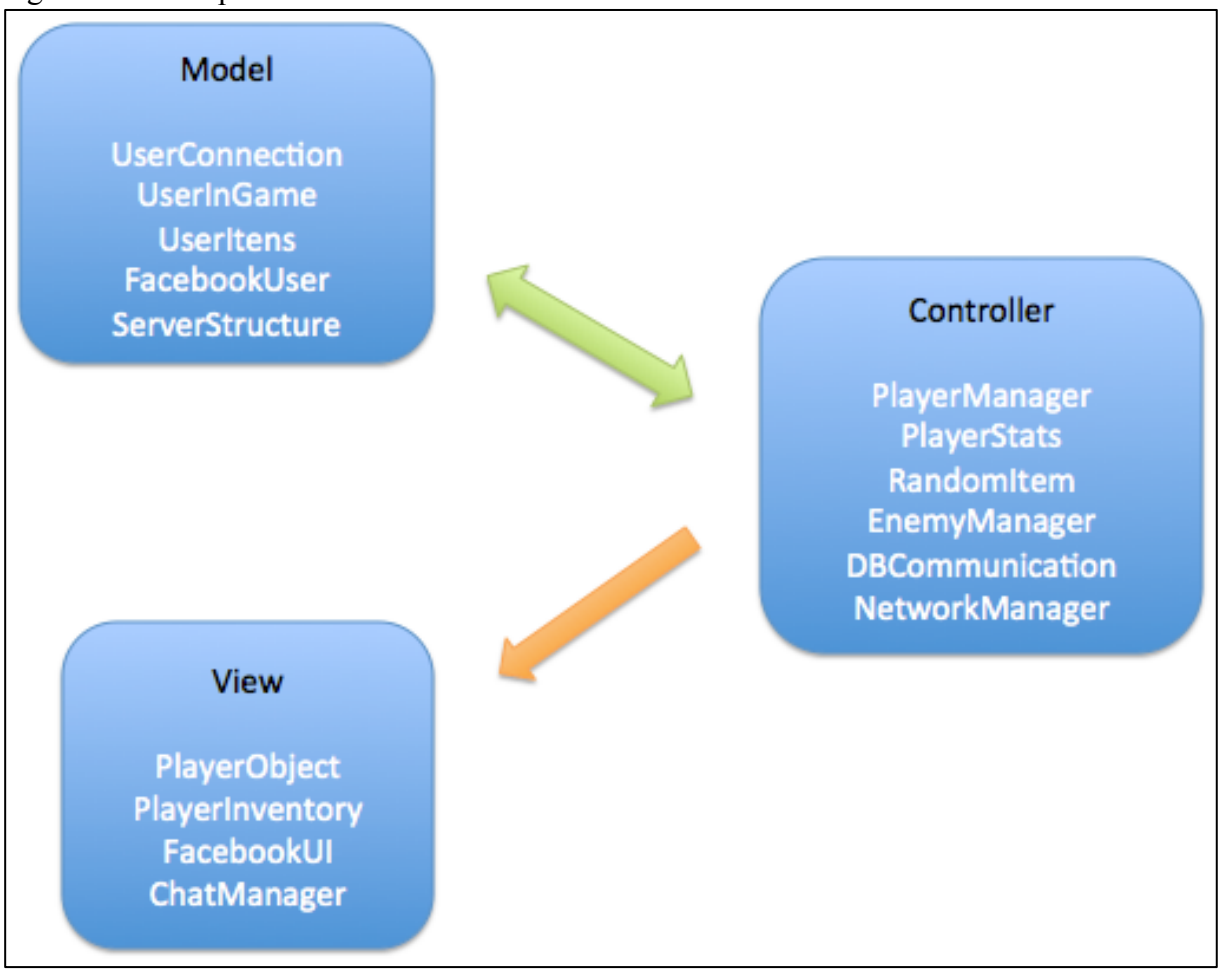
O projeto consiste em três camadas de aplicação, segundo o modelo de arquitetura de *software* MVC. Segundo Gamma, Helm, Johnson *et al*. (1998), o modelo MVC é

constituído por três tipos de objetos. O Modelo (*Model*) é o objeto de aplicação, a Visão (*View*) é a sua apresentação na tela, e o Controlador (*Controller*) define a maneira como a interface do usuário reage a entrada do usuário. As três camadas no projeto estão definidas como:

a) a camada *View* possui o código em C# que se comunica diretamente com o jogo dentro da *engine*, que possui a interface implementada.
b) a camada de *Controller* é composta pelo código em C# dentro da *engine* com a lógica do jogo e pelos códigos em PHP e Javascript que fazem o controle dos dados do Facebook e as buscas no banco de dados com as necessidades que o jogo precisa.
c) a camada *Model* com o banco de dados da aplicação.

A Figura 24 mostra os componentes dentro do modelo MVC.

Figura 24 – Componentes no modelo MVC



Fonte: Elaboração do autor (2013).

O projeto dentro da *Unity 3D* possui uma hierarquia de pastas onde os recursos ficam separados por tipos:

a) a pasta "*Scripts*" que contém os arquivos em C# de toda a aplicação.
b) a pasta "*ServerSide*" que contém os *scripts* que ficam no servidor da aplicação.
c) a pasta "*WebPlayerTemplates*" (que é um nome usado por padrão na *Unity 3D)* que contém a pasta com o SDK do Facebook, a página *web* principal do jogo (index.php), e o *script* que se comunica com o Facebook.
d) a pasta "*Resources*" (também padrão da *Unity 3D*) que possui arquivos de imagem e os *Prefabs*[20] que são carregados em *runtime* (no tempo em que o jogo está executando).
e) as pastas com imagens pré-definidas para serem utilizadas na interface do jogo e que não precisam ser carregadas em *runtime*.

Além das pastas e seus recursos, o projeto possui uma cena de exemplo com o cenário ajustado para a utilização dos componentes.

5.4.2 Componente do jogador

Este componente é caracterizado por gerenciar todas as interações do jogador com o jogo. Ele é dividido em 3 macro objetos, *PlayerManager, PlayerObject* e *PlayerInventory*, cada um com características particulares que serão descritas a seguir.

5.4.2.1 Gerenciador do jogador

Nome: *PlayerManager*

[20] *Prefab* é um tipo de ativo (*asset*) que pode envolver um ou mais objetos dentro dele, com as suas configurações definidas, e ficam armazenados dentro do projeto da *Unity 3D*.

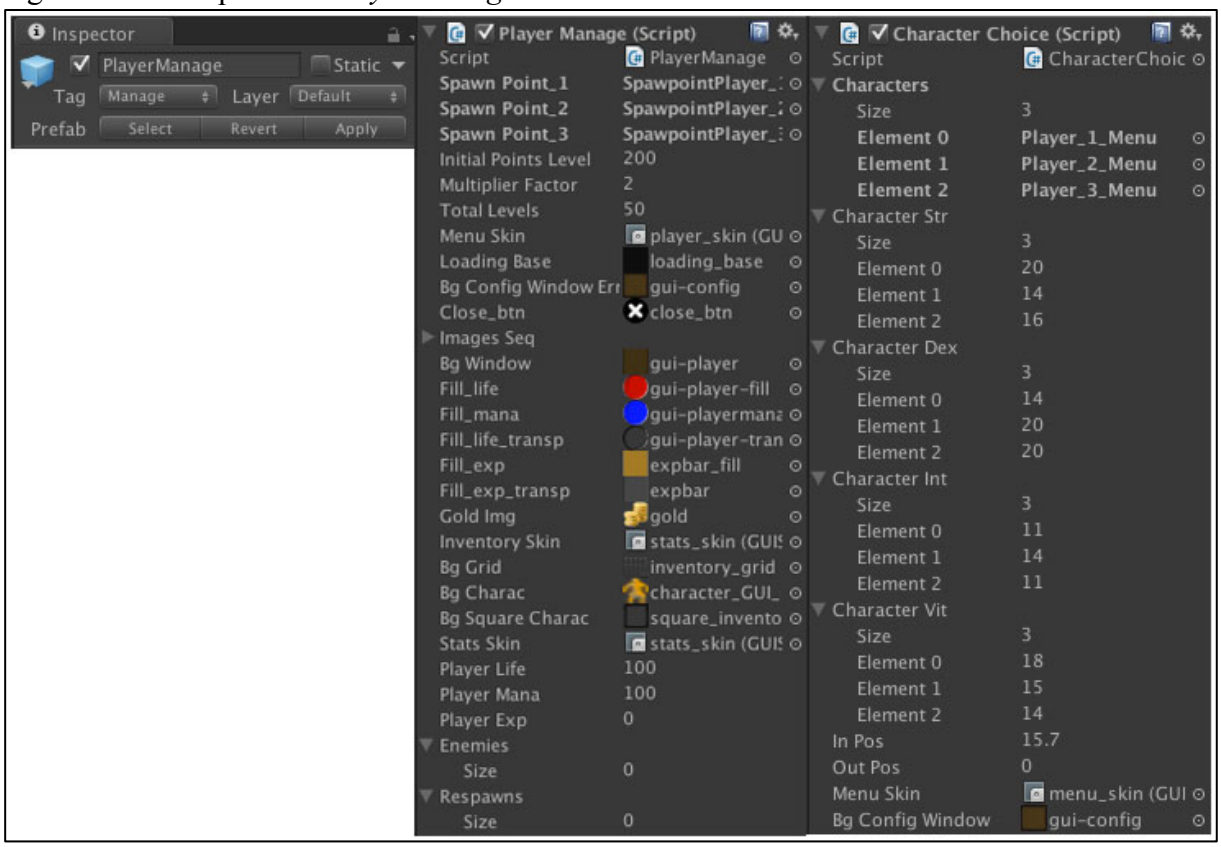
Descrição: está presente na camada de *Controller* da aplicação. É um *Singleton* que, segundo Gamma, Helm, Johnson *et al*. (1998), certifica que a classe tenha apenas uma instância e fornece um ponto global de acesso a ela.

Função: gerenciar do início ao fim do jogo tudo o que o jogador faz durante a partida, como o recebimento e envio dos dados do jogador para o banco de dados através do componente de comunicação externa à *engine*. Suas principais funções são:

a) instanciar os objetos de *database*, estatísticas, inventário, gerador de níveis, o objeto de *network* e o Facebook do jogador.
b) enviar os dados recebidos do banco de dados para os componentes de *game design* preencherem seus atributos.
c) gerenciar a GUI que permanece visível durante todo o jogo, que possui o nível, a vida, a mana, os pontos de experiência e o dinheiro que o jogador possui.
d) gerenciar o inicio do jogo para garantir que, se o jogador não possuir um personagem criado, dar a ele essa possibilidade de escolha.

Aplicabilidade: é um *Prefab* com dois *scripts* anexados a ele: "*PlayerManager.cs*" e "*CharacterChoice.cs*". Ele precisa estar presente na cena e possui variáveis públicas que o tornam configurável no editor da *Unity 3D*. A Figura 25 mostra como ele fica dentro da *engine*.

Figura 25 – Componente *PlayerManager*



Fonte: Elaboração do autor (2013).

Dependências: ele necessita dos componentes que se comunicam com a camada *Model* da aplicação, ou seja, precisa que os dados venham do *database* para que possa gerenciar e enviá-los para os outros componentes.

Considerações sobre a codificação: o *script* "*PlayerManager.cs*" tem como uma de suas funções instanciar outros componentes no início do jogo, além de fazer a comunicação inicial com o Facebook para resgatar o identificador único (id) do jogador. Com isso, todos os objetos que precisam estar no inicio da cena, devem ser instanciados dentro desse método. A Figura 26, demonstra como deve ser o método de *Start* desse componente.

Figura 26 – Trecho do *script PlayerManager.cs*

```
void Start()
{
    _loading = true;

    GameObject DbComponent = new GameObject("DBConfig");
    DbComponent.gameObject.tag = "Database";
    DbComponent.AddComponent<DbCommunication>();
    DbComponent.GetComponent<DbCommunication>().setPlayerManageComponent(this);
    _dbCall = DbComponent.GetComponent<DbCommunication>();

    GameObject StatsComponent = new GameObject("PlayerStats");
    StatsComponent.AddComponent<PlayerStats>();
    StatsComponent.gameObject.tag = "Stats";
    StatsComponent.GetComponent<PlayerStats>().setPlayerManageComponent(this);
    StatsComponent.GetComponent<PlayerStats>().setStatsTextures(statsSkin, bgWindow);
    StatsComponent.GetComponent<PlayerStats>().configGUI(350, 340, 0, 0);
    _plStats = StatsComponent.GetComponent<PlayerStats>();

    GameObject InventoryComponent = new GameObject("PlayerInventory");
    InventoryComponent.gameObject.tag = "Inventory";
    InventoryComponent.AddComponent<PlayerInventory>();
    InventoryComponent.GetComponent<PlayerInventory>().setPlayerManageComponent(this);
    InventoryComponent.GetComponent<PlayerInventory>().setInventoryTextures(inventorySkin, bgWindow,
        bgGrid, loadingBase, bgCharac, bgSquareCharac);
    _plInv = InventoryComponent.GetComponent<PlayerInventory>();

    _facebookObj = (GameObject)Instantiate(Resources.Load("Prefabs/FacebookUI"),
        this.transform.localPosition, Quaternion.identity);
    _facebookObj.name = "FacebookUI";
    _facebookObj.gameObject.tag = "Facebook";
    _facebookAction = _facebookObj.GetComponent<FacebookUI>();

    _levels = new Levels(initialPointsLevel, multiplierFactor, totalLevels);

    _network = GameObject.FindGameObjectWithTag("Network").GetComponent<NetworkManager>();

    //Call javascript web to get user facebook id
    Application.ExternalCall("requestUid");
}
```

Fonte: Elaboração do autor (2013).

5.4.2.2 Gerenciador das ações do jogador

Nome: *PlayerObject*

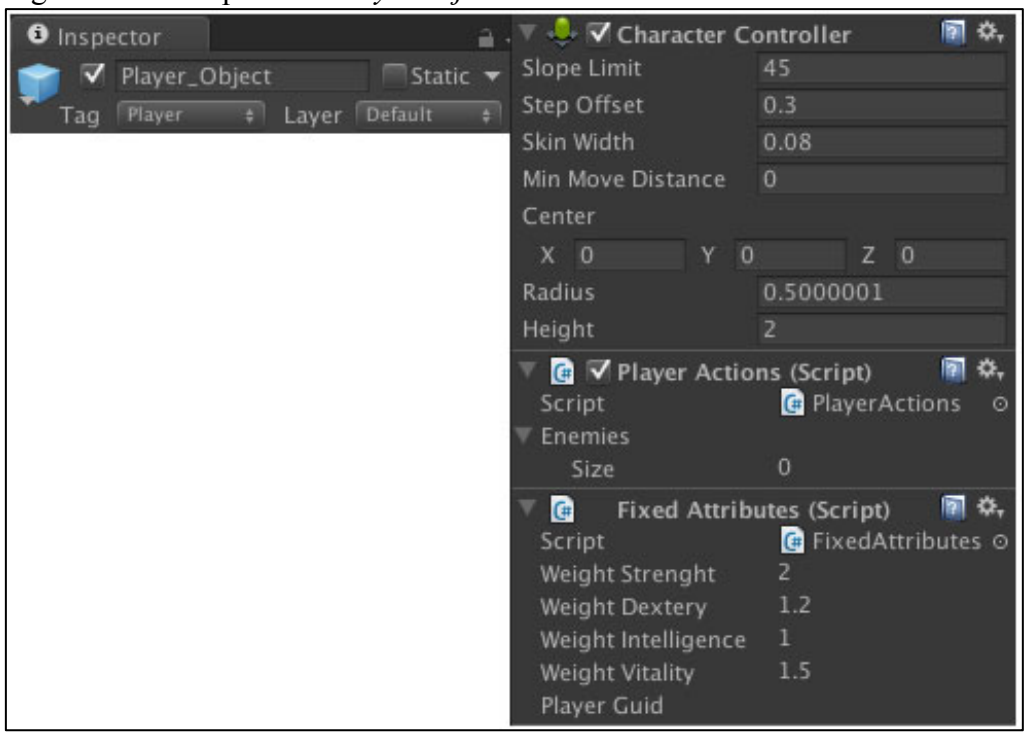
Descrição: está presente na camada de *View* da aplicação por ser a interação entre o jogador e as suas ações dentro do jogo.

Função: gerenciar do início ao fim do jogo as ações tomadas pelo jogador, como movimentar o avatar, clicar em objetos e atacar inimigos. Ainda possui a responsabilidade de guardar o id do jogador quando estiver em modo *multiplayer* e possui os valores dos pesos

para a multiplicação dos atributos do avatar conforme o seu tipo, por exemplo, no cálculo do dano que o jogador gera, esse peso pode ser maior ou menor para determinado tipo de avatar, dependendo de quais são as suas especialidades.

Aplicabilidade: é um *Prefab* com dois *scripts* anexados a ele: "*PlayerActions.cs*" e "*FixedAttributes.cs*". Ele precisa estar dentro do objeto que é o avatar, configurado com a *tag* (identificador global dentro do projeto) "*Player*". O objeto do avatar precisa ter um componente de *CharacterController* nativo da própria *Unity 3D* para que a movimentação ocorra. A Figura 27 mostra como ele fica dentro da *engine*.

Figura 27 – Componente *PlayerObject*



Fonte: Elaboração do autor (2013).

Dependências: esse componente é independente dos outros. Porém, se algum componente tiver alguma GUI, ele necessita saber se o jogador está ou não interagindo com esse elemento de jogo para poder ativar ou não os cliques responsáveis por movimentação e interação com objetos e inimigos.

Considerações sobre a codificação: o *script* "*PlayerActions.cs*" possui o código de movimentação do avatar. Esse tipo de código pode ser feito de várias maneiras, dependendo de como é decidido o tipo de visão que o jogador vai ter. Porém, esse

componente precisa ter um método muito importante, o qual evita que mais de um jogador controle o mesmo avatar na cena durante um jogo *multiplayer*. A Figura 28 mostra o método que tem a informação de que o jogo está no modo *multiplayer*, desativando a câmera que segue o avatar e a movimentação dele na cena.

Figura 28 – Trecho do *script PlayerActions.cs*

```
void OnNetworkInstantiate(NetworkMessageInfo info)
{
    if(!networkView.isMine)
    {
        transform.parent.FindChild("Main Camera").gameObject.SetActive(false);
        enabled = false;
    }
}
```

Fonte: Elaboração do autor (2013).

5.4.2.3 Gerenciador de inventário do jogador

Nome: *PlayerInventory*

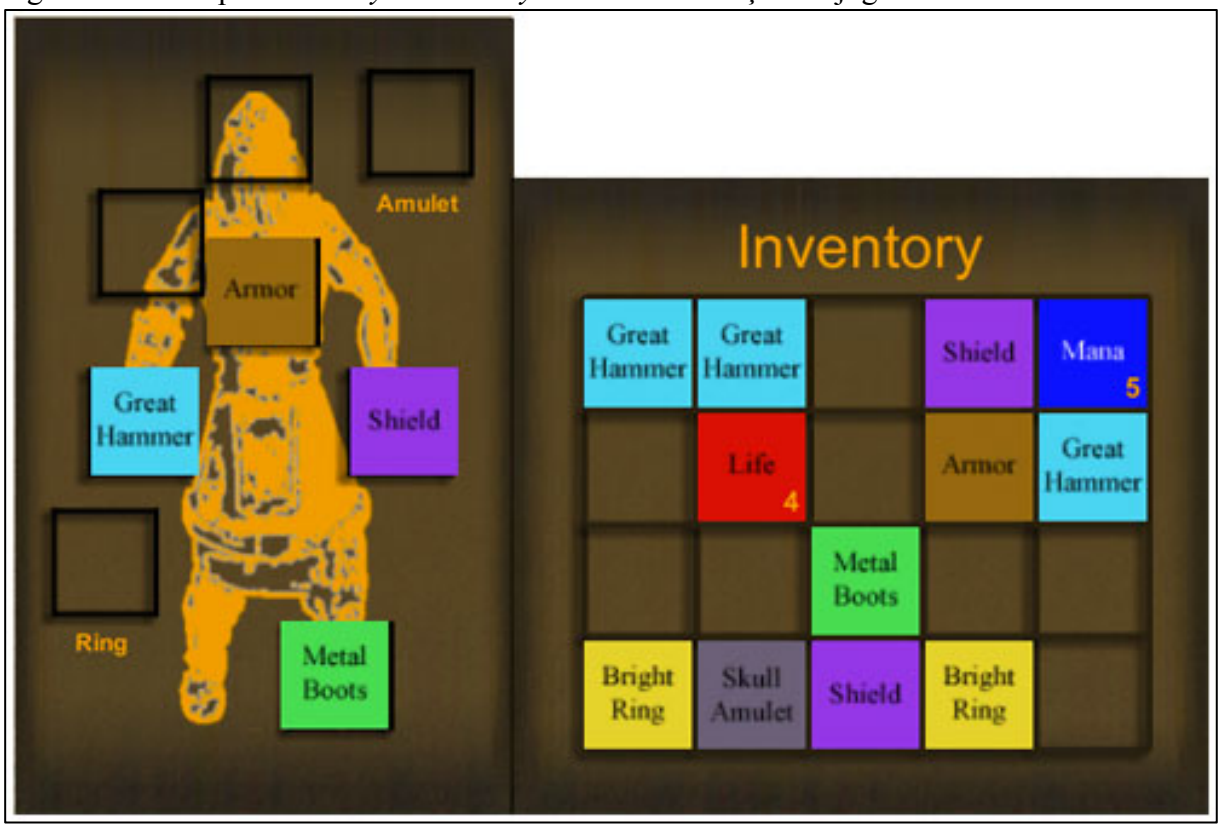
Descrição: é um *Singleton* que está presente na camada de *View* da aplicação por ser a interação entre o jogador e as suas ações dentro do seu inventário de itens.

Função: gerenciar do início ao fim do jogo as ações tomadas pelo jogador para inserir, alterar e excluir os itens do seu inventário. Também especifica a GUI do inventário do jogador e precisa estar disponível para mostrá-la toda vez que o jogador requisitar.

Aplicabilidade: é um *GameObject*[21] criado no inicio do jogo pelo componente *PlayerManager*. Quando é instanciado, precisa ser anexado a ele o *script* "*PlayerInventory.cs*" e receber alguns parâmetros como a *tag Inventory* e as texturas que serão usadas na GUI. A Figura 29 mostra como fica o inventário do jogador durante o jogo.

[21] *GameObjects* são containers para todos os outros componentes dentro da *Unity 3D*. Todos os objetos dentro do jogo são inerentemente *GameObjects*.

Figura 29 – Componente *PlayerInventory* durante a execução do jogo



Fonte: Elaboração do autor (2013).

Dependências: ele necessita do componente *PlayerManager* que recebe as requisições dele para buscar todos os itens do jogador no banco de dados e todas as ações de inserir, equipar e excluir os itens. Também necessita do componente do item para poder instanciá-lo e utilizar os seus recursos, como o modelo 3D, o seu ícone e as suas características.

Considerações sobre a codificação: toda vez que o jogador desejar ver o seu inventário de itens, o *script* "*PlayerInventory.cs*" verifica se já foi feita alguma requisição ao banco de dados dos itens, ou se possui algum item novo para preencher a lista de itens. Esse *script* possui um método que recebe todos os itens e seus atributos, através do componente *PlayerManager*, e preenche a lista com os itens a serem mostrados na GUI do componente e um dicionário de dados desse item com seus atributos. A Figura 30 mostra esse método.

Figura 30 – Trecho do *script PlayerInventory.cs*

```
public void fillInventory(string itens)
{
    itensCollection = new List<Item>();

    string[] tempElementData = itens.Split("|"[0]);

    for(int i = 0; i < tempElementData.Length - 1; i++)
    {
        string[] tempKeyValue = tempElementData[i].Split("#"[0]);
        _itemData = new Dictionary<string, string>();

        for(int ii = 0; ii < tempKeyValue.Length; ii++)
        {
            string[] tempValue = tempKeyValue[ii].Split("="[0]);
            _itemData.Add(tempValue[0], tempValue[1]);
        }

        bool stack =  _itemData["stackable"] == "1" ? true : false;
        bool isTwoHands = _itemData["two_hands"] == "1" ? true : false;

        Item item = new Item(_itemData["name"], _itemData["full_name"], _itemData["description"],
                    _itemData["position"], int.Parse(_itemData["price"]), stack,
                    int.Parse(_itemData["item_amount"]), int.Parse(_itemData["min_level"]),
                    int.Parse(_itemData["min_strenght"]), int.Parse(_itemData["min_dextery"]),
                    int.Parse(_itemData["min_intelligence"]), int.Parse(_itemData["durability"]),
                    int.Parse(_itemData["actual_durability"]), int.Parse(_itemData["gain_strenght"]),
                    int.Parse(_itemData["gain_dextery"]), int.Parse(_itemData["gain_intelligence"]),
                    int.Parse(_itemData["gain_vitality"]), int.Parse(_itemData["gain_life"]),
                    int.Parse(_itemData["gain_mana"]), isTwoHands, _itemData["relates"]);

        itensCollection.Add(item);
    }
}
```

Fonte: Elaboração do autor (2013).

5.4.3 Componente de *Game Design*

Este componente é caracterizado por gerenciar as regras do jogo, como a evolução do jogador, a quantidade de inimigos e suas habilidades e o surgimento de itens. Ele é dividido em 3 macro objetos, *PlayerStats*, *RandomItem* e *EnemyManager*, cada um com características particulares que serão descritas a seguir.

5.4.3.1 Gerenciador dos atributos do jogador

Nome: *PlayerStats*

Descrição: este componente está presente na camada de *Controller* da aplicação por ter responsabilidade de controlar os níveis e as estatísticas do jogador.

Função: gerenciar do início ao fim do jogo os atributos do jogador para alterar a quantidade de força, destreza, inteligência e vitalidade do avatar. Também especifica a GUI das estatísticas do jogador e precisa estar disponível para mostrá-la toda vez que for requisitada. Ainda possui 4 responsabilidades muito importantes para o jogador:

a) baseado na força, destreza e inteligência que o avatar possui, ele faz o cálculo de dano que o jogador irá gerar ao atacar um inimigo.
b) baseado na força, destreza e vitalidade que o avatar possui, ele faz o cálculo da defesa que o jogador tem quando um inimigo o ataca.
c) gerencia os pontos que o jogador pode alterar dos seus atributos ao passar de nível.
d) cria no jogo o sistema de *levels* (níveis) que podem ser alcançados pelo jogador.

Aplicabilidade: é um *GameObject* criado no inicio do jogo pelo componente *PlayerManager*. Quando é instanciado, precisa ser anexado a ele dois *scripts*: "*PlayerStats.cs*" e "*Levels.cs*". O primeiro precisa receber alguns parâmetros como a *tag Stats* e as texturas que serão usadas na GUI. O segundo precisa receber 3 parâmetros os quais irão definir o cálculo dos níveis do jogo: os pontos necessários para passar o primeiro nível, o fator de multiplicação desses níveis e a quantidade de níveis que o jogo vai possuir. A Figura 31 mostra como fica a GUI das estatísticas do jogador durante o jogo.

Figura 31 – Componente *PlayerStats* durante a execução do jogo

Fonte: Elaboração do autor (2013).

Dependências: ele necessita do componente *PlayerManager* para ser instanciado, para requisitar os atributos ao banco de dados e para alterá-los.

Considerações sobre a codificação: os cálculos de dano e defesa são feitos através de duas funções públicas no *script* "*PlayerStats.cs*" que retornam o valor exato desses atributos para que possam ser usados por outros componentes. Qualquer alteração nessa função, seja aumentar ou diminuir o número de variáveis que entram nesse cálculo, é bem simples. A Figura 32 demonstra as duas funções.

Figura 32 – Cálculo de dano e defesa do jogador

```
public float calculateDamage()
{
    float weightSt = _playerAtt.weightStrenght;
    float weightDe = _playerAtt.weightDextery;
    float weightIn = _playerAtt.weightIntelligence;

    if(_levelUp)
        getAttributes();

    return ((plManage.getStrenght() * weightSt) + (plManage.getDextery() * weightDe) +
        (plManage.getIntelligence() * weightIn)) / 3;
}

public float calculateDefense()
{
    float weightSt = _playerAtt.weightStrenght;
    float weightDe = _playerAtt.weightDextery;
    float weightVi = _playerAtt.weightVitality;

    if(_levelUp)
        getAttributes();

    return ((plManage.getStrenght() * weightSt) + (plManage.getDextery() * weightDe) +
        (plManage.getVitality() * weightVi)) / 3;
}
```

Fonte: Elaboração do autor (2013).

5.4.3.2 Gerenciador dos itens

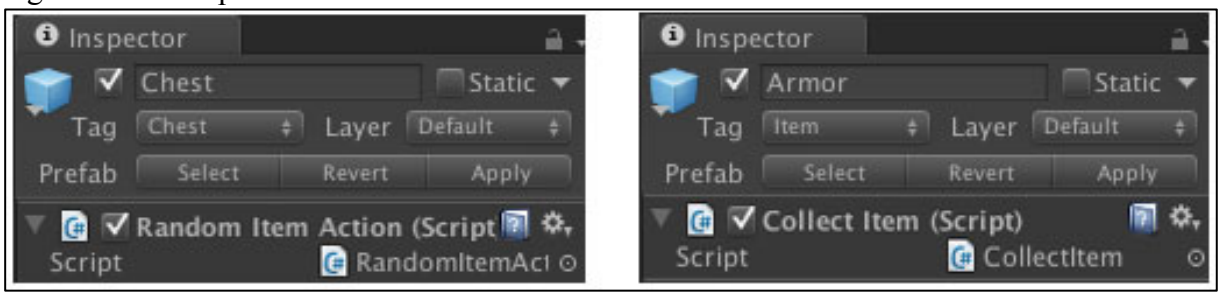
Nome: *RandomItem*

Descrição: este componente está presente na camada de *Controller* da aplicação por ter responsabilidade de controlar os itens gerados durante a partida pelos *scripts* que estão no servidor.

Função: gerenciar do início ao fim do jogo os itens que serão gerados e coletados pelo jogador, sejam por baús (*chests*) ou pelos inimigos. Por ser um componente genérico, ele pode ser anexado a qualquer elemento do jogo que se deseje obter um item de forma randômica. Esse componente também possui a classe do item, que encapsula todos os seus atributos para serem usados dentro dele.

Aplicabilidade: possui 3 *scripts*: "*Item.cs*" que é a classe principal do item e possui todos os seus atributos, o "*RandomItemAction.cs*" que deve ser anexado nos elementos que podem gerar itens (no caso de baú, ele precisa ter a *tag* "*Chest*") e o "*CollectItem.cs*" que deve estar anexado ao *GameObject* do item para receber os parâmetros de inicialização dele. A Figura 33 mostra como ficam esses *scripts* anexados aos seus dois tipos correspondentes.

Figura 33 – Componente *RandomItem*



Fonte: Elaboração do autor (2013).

Dependências: ele necessita dos componentes da camada *Model* da aplicação, ou seja, precisa que os dados venham do *database* para que ele possa instanciar os itens.

Considerações sobre a codificação: o *script* "*CollectItem.cs*" se comunica diretamente com o componente *PlayerObject* para saber quando o item foi clicado para ser coletado pelo jogador. Também se comunica com o componente *PlayerManager* para enviar o item clicado, que se responsabilizará por salvar esse item. A Figura 34 demonstra como deve ser verificado e enviado esse item para o componente responsável.

Figura 34 – Trecho do *script CollectItem.cs*

```
if(PlayerActions.itemClicked && !_getItem)
{
    if(PlayerActions.nameClickedObj == this.transform.gameObject.name)
    {
        _getItem = true;

        string[] objName = new string[] {_processData["name"], this.transform.gameObject.name};

        //Try to save item in database
        _plManage.inventoryHandle(objName);
    }
}
```

Fonte: Elaboração do autor (2013).

5.4.3.3 Gerenciador dos inimigos

Nome: *EnemyManager*

Descrição: este componente está presente na camada de *Controller* da aplicação por ter responsabilidade de controlar os inimigos, preencher os seus atributos e definir onde, quando e como eles aparecerão no cenário.

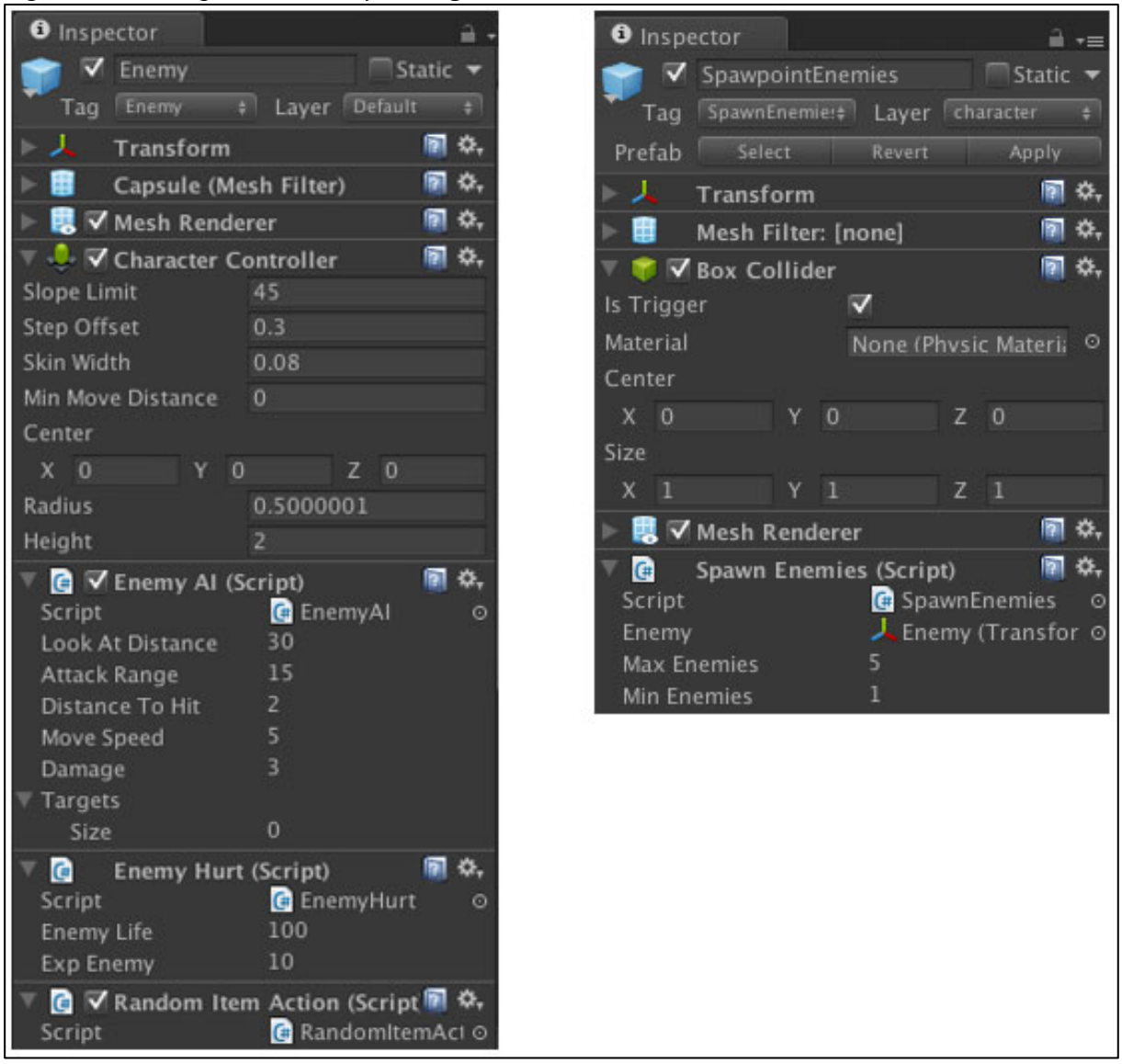
Função: gerenciar do início ao fim do jogo os inimigos. Esse componente possui algumas características muito importantes para o balanceamento do jogo. Suas principais funcionalidades são:

a) movimentação no cenário: possui uma inteligência artificial (AI) para se movimentar de forma aleatória, a uma determinada velocidade e decidir qual oponente atacar.
b) ataque aos oponentes: possui um *script* configurável para cada tipo de inimigo que determina qual a distância mínima para partir para o ataque e o dano aleatório, entre o mínimo e o máximo, que é gerado a cada ataque.
c) atributo de defesa: determina, baseado na quantidade de dano do oponente, quanto ele consegue se defender e diminuir o valor desse dano.
d) vida: dentro de um valor pré-definido para a quantidade de vida, é gerado aleatoriamente em 30%, a mais ou a menos, o valor da sua vida.
e) experiência gerada: dentro de um valor pré-definido para a quantidade de experiência, é gerado aleatoriamente em 30%, a mais ou a menos, o valor da experiência que ele irá dar ao jogador quando for morto.
f) aparecer no cenário: gera aleatoriamente os inimigos em uma área definida. A quantidade é escolhida dentro de um mínimo e um máximo definidos para aquela área, assim como o tipo de inimigo e as posições que eles irão ocupar.

Aplicabilidade: possui 3 *scripts*: "*EnemyAI.cs*" que é anexado ao *GameObject* do inimigo e onde são definidos, pelo desenvolvedor, os valores de: distância para atacar, velocidade de movimentação e dano, o "*EnemyHurt.cs*" que também é anexado ao *GameObject* do inimigo e onde são definidos, pelo desenvolvedor, os valores de vida e

experiência gerados e o "*SpawnEnemies.cs*" que deve estar anexado a um *GameObject* na cena, que contenha um componente nativo da *Unity 3D* de colisão (precisa ser configurado como *trigger*, ou seja, não afeta a física dos elementos e é invisível), e nele são definidos pelo desenvolvedor, o tipo, a quantidade máxima e a mínima de inimigos que aparecerão dentro desse espaço. O objeto do inimigo precisa ter o componente de *CharacterController* nativo da própria *Unity 3D* para que a movimentação ocorra e também a *tag* "*Enemy*". O objeto gerador dos inimigos precisa ter a *tag* "*SpawnEnemies*". A Figura 35 mostra como ficam esses *scripts* anexados aos seus dois tipos correspondentes.

Figura 35 – Componente *EnemyManager*



Fonte: Elaboração do autor (2013).

Dependências: esse componente é independente dos outros. Porém, quando o inimigo sofre um dano, precisa saber os valores de ataque do oponente para realizar os seus cálculos de defesa.

Considerações sobre a codificação: as ações do inimigo são definidas por achar o oponente mais próximo a ele e verificar se está no seu raio de ação ou não. As variáveis do raio de ação são valores constantes que podem variar para cada tipo de inimigo. A Figura 36 mostra como o inimigo decide qual oponente atacar.

Figura 36 – Trecho do *script EnemyAI.cs*

```
void FixedUpdate()
{
    _targets = GameObject.FindGameObjectsWithTag("Player");
    _distances = new float[_targets.Length];

    for(int i = 0; i < _targets.Length; i++)
    {
        //Calculate the _distance to get action
        _distances[i] = Vector3.Distance(_targets[i].transform.position, transform.position);
    }

    float minor = _distances[0];
    int index = 0;

    for(int d = 0; d < _targets.Length; d++)
    {
        if(_distances[d] <= minor)
        {
            minor = _distances[d];
            index = d;
        }
    }

    if(_distances[index] < lookAtDistance)
    {
        LookAtTarget(_targets[index]);
    }

    if(_distances[index] < attackRange)
    {
        _distance = _distances[index];
        AttackTarget(_targets[index]);
    }
}
```

Fonte: Elaboração do autor (2013).

5.4.4 Componente de comunicação externa à *Unity 3D*

Este componente é caracterizado por gerenciar a comunicação da *engine* com o banco de dados e os dados do Facebook. Ele é dividido em 2 macro objetos, *DBCommunication* e *FacebookUI*, cada um com características particulares que serão descritas a seguir.

5.4.4.1 Gerenciador da comunicação com o servidor

Nome: *DBCommunication*

Descrição: este componente é um *Singleton* que está presente na camada de *Controller* da aplicação por ter responsabilidade de controlar todas as chamadas ao servidor e distribuir os dados recebidos aos respectivos componentes que fizeram a requisição.

Função: gerenciar do início ao fim do jogo todas as chamadas ao servidor. Tem a responsabilidade de guardar o objeto que fez a requisição para entregar a resposta que ele necessita, assim dando continuidade as suas funções. Também possui o endereço do servidor e a lista de todos os arquivos que lá estão hospedados. Este componente não altera nem toma decisões dentro do jogo, apenas faz o controle dos dados que entram e que saem na comunicação entre a *engine* e o servidor.

Aplicabilidade: possui 2 *scripts*: "*DBCommunication.cs*" que contém todos os métodos de acesso ao servidor (o componente *PlayerManager* instancia um *GameObject* no inicio do jogo, e este recebe o *script* automaticamente) e o "*ServerAddresses.cs*" que é instanciado pelo *DBCommunication*, recebendo o nome dos *scripts* que ficam no servidor. Dentro do projeto da *Unity 3D*, eles não são manipulados para serem colocados na cena, o componente *PlayerManager* se encarrega disso.

Dependências: esse componente depende de todos os outros componentes listados nessa seção, pois ele não faz sentido no projeto sem que receba requisições e tenha para onde enviar as respostas.

Considerações sobre a codificação: o *script* "*DbCommunication.cs*" possui todas as chamadas para o servidor. O ponto importante sobre esse código é como devem ser feitas essas chamadas. A Figura 37 mostra como devem ser as chamadas para o servidor e receber as suas respostas.

Figura 37 – Chamadas ao servidor no *script DbCommunication.cs*

```
public void loginUser(string uid, string name)
{
    _idUser = uid;
    _nameUser = name;

    StartCoroutine(connectUserDb());
}

IEnumerator connectUserDb()
{
    _webServer = _address.getconnectUserDb();

    _param = new WWWForm();
    _param.AddField("uid", _idUser);
    _param.AddField("name", _nameUser);

    WWW webRequest = new WWW(_webServer, _param);
    yield return webRequest;

    if(webRequest.error != null)
    {
        _playerCall.dbResponse("error", "login");
    }
    else
    {
        resp = webRequest.text;
    }
}
```

Fonte: Elaboração do autor (2013).

5.4.4.2 Gerenciador da comunicação com o Facebook

Nome: *FacebookUI*

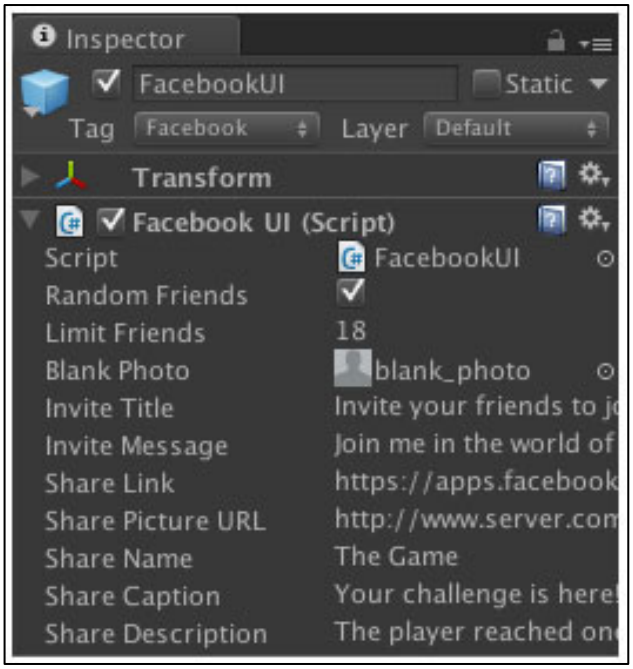
Descrição: este componente está presente na camada de *View* da aplicação por ter a responsabilidade de receber e exibir os dados que foram requisitados ao Facebook e interagir com o jogador.

Função: exibir a interação do jogador com a rede social do Facebook. Possui 4 grandes responsabilidades:

a) receber as configurações do aplicativo no Facebook, inseridas pelo desenvolvedor, e passar para a página principal do jogo.
b) exibir os dados do Facebook do jogador, como id, nome, foto e lista de amigos.
c) enviar para a página principal do jogo as requisições dos outros componentes feitas pelo jogador para convidar amigos para participar do jogo dentro do Facebook.
d) enviar para a página principal do jogo as requisições dos outros componentes para publicar ações no mural do usuário no Facebook.

Aplicabilidade: o *script* "*FacebookUI.cs*" deve estar anexado a um *GameObject* com a *tag* "*Facebook*". Nele devem ser preenchidos os dados para serem compartilhados dentro do Facebook. A Figura 38 mostra como o componente fica dentro da *Unity 3D*.

Figura 38 – Componente *FacebookUI*



Fonte: Elaboração do autor (2013).

Dependências: esse componente é independente dos outros.

Considerações sobre a codificação: toda a comunicação entre a *Unity 3D* e o *browser* deve ser utilizando o Javascript que está presente na página. Ele possui as funções definidas para receber e retornar dados. A Figura 39 mostra como a *engine* faz esse tipo de chamada para requisitar dados do Facebook.

Figura 39 – Chamadas externas da *engine* para o Facebook

```
public void getFacebookUser()
{
    //Call javascript web to get user data
    Application.ExternalCall("getFacebookUser");
}
```

Fonte: Elaboração do autor (2013).

5.4.5 Componente de *multiplayer*

Este componente é caracterizado por gerenciar todas as conexões para o jogo *multiplayer*, como a criação do jogador que será o servidor, os outros jogadores que serão clientes, o *chat* (bate-papo) entre os participantes da partida e tornar todos os elementos do cenário visíveis para a rede. Ele é dividido em 2 macro objetos, *NetworkManager* e *ChatManager*, cada um com características particulares que serão descritas a seguir.

5.4.5.1 Gerenciador da rede

Nome: *NetworkManager*

Descrição: este componente está presente na camada de *Controller* da aplicação por ter a responsabilidade de iniciar, gerenciar e finalizar uma partida *multiplayer*.

Função: gerenciar toda a partida *multiplayer*. Possui a grande responsabilidade de manter o estado do jogo igual para todos os participantes, independente do momento em que se conectam ao servidor. Sua estrutura é definida por eventos que ocorrem no jogador servidor e envia para todos os outros (*broadcast*). Seus principais eventos são:

a) o evento *OnServerInitialized*, que define quando um jogador decide iniciar uma partida *multiplayer*. Esse evento só acontece no servidor, pois transforma o jogador em um elemento *Network* (visto pela rede) e busca todos os elementos dinâmicos na cena atual e os instancia novamente para serem visíveis pela rede, mantendo seus estados atuais.

b) o evento *OnConnectedToServer*, que define quando um jogador se conectou ao servidor. Esse evento só acontece no cliente e é responsável por instanciar esse jogador na rede e receber toda a sincronia de elementos e seus estados.

c) o evento *OnPlayerDisconnected*, que define quando um jogador de desconecta do servidor. Esse evento só acontece no servidor, que se responsabiliza por remover esse jogador da cena e da rede, avisando a todos os outros clientes desse acontecimento para serem sincronizados.

d) o evento *OnDisconnectedFromServer*, que acontece tanto no servidor como nos clientes. Esse evento ocorre quando o jogador servidor fecha a partida para *multiplayer*. No cliente a cena é reiniciada, pois a partida não foi criada por ele, e no servidor a cena continua da mesma forma, porém os jogadores clientes são removidos.

Esse componente possui em seu *script* uma variável booleana pública e estática (*public static bool*) chamada "*isNetworking*", que é acessada por todos os outros componentes do jogo. Esses outros componentes precisam saber se o jogo está sendo *multiplayer* ou não, porque dentro de seus códigos, em alguns pontos, o fluxo de informações e ações é diferente de *single player* para *multiplayer*.

Além disso, esse componente também gerencia a GUI onde o usuário pode iniciar o jogo na rede, inserir o *ip* e se conectar ao servidor, desconectar ou finalizar a partida na rede e também mostra, no cliente, o tempo de *ping*[22] para o servidor e, no servidor, a quantidade de jogadores conectados a ele.

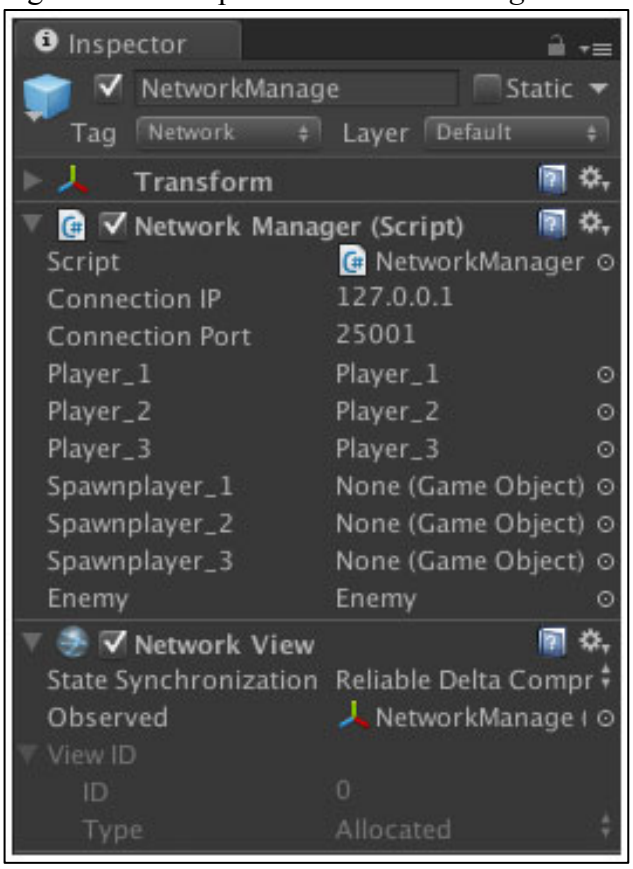
[22] Pode ser chamado também de latência, e serve para testar a conectividade entre equipamentos.

Aplicabilidade: possui o *script*: "*NetworkManager.cs*" que é anexado a um *GameObject* com a *tag* "*Network*" e precisa estar presente na cena durante todo o jogo. Além do *script*, ele precisa ter um componente nativo da *engine* chamado "*Network View*", que é responsável por guardar um identificador para esse objeto na rede e definir o tipo de sincronização que esse objeto tem em relação à rede. Essa sincronização pode ser de 3 tipos:

a) *Unreliable*: a *Unity 3D* envia os pacotes sem checar se eles foram recebidos. Isso significa que ele não sabe que a informação foi recebida e por isso não é seguro enviar apenas os dados alterados.
b) *Reliable Delta Compressed*: esse modo, automaticamente, irá comparar os dados que foram recebidos e passados para o cliente com o estado atual. Se nenhum dado foi alterado desde a última atualização, em seguida, os dados não serão enviados.
c) *Off*: nenhuma sincronização é feita, sendo necessário o desenvolvedor cuidar de todo o envio e recebimento dos pacotes de dados.

Esse componente precisa ser configurado como "*Reliable Delta Compressed*", já que ele contém os eventos de criar a conexão, conectar e desconectar jogadores e finalizar o jogo. Por exemplo, é preciso garantir que quando um jogador desconecta do jogo, todos os clientes recebam essa atualização e deixem de ter aquele *player* dentro da cena. A Figura 40 mostra como o componente fica dentro da *Unity 3D*.

Figura 40 – Componente *NetworkManager*



Fonte: Elaboração do autor (2013).

Dependências: esse componente é independente dos outros, porém, pode receber alguns parâmetros dos objetos que serão instanciados.

Considerações sobre a codificação: algumas funções devem ser chamadas via RPC (permite chamar funções em máquinas remotas) para que todos os jogadores executem uma determinada ação. Os modos de envio desses RPCs podem ser para:

a) *server*: apenas o jogador servidor vai executar a ação.

b) *others*: todos irão executar a ação, exceto o remetente.

c) *othersbuffered*: todos irão executar a ação, exceto o remetente. Porém, essa chamada fica armazenada no *buffer*[23] da rede, para quando um novo jogador entrar na partida, também receber essa chamada e executar a ação.

d) *all*: todos irão executar a ação, inclusive o remetente.

e) *allbuffered*: todos irão executar a ação, inclusive o remetente. Essa chamada fica armazenada no *buffer* da rede, para quando um novo jogador entrar na partida, também receber essa chamada e executar a ação.

[23] Armazenamento temporário de dados para serem usados em um determinado momento.

A Figura 41 mostra como é feita uma chamada RPC para todos os participantes, inclusive adicionando ao *buffer*, que um dos jogadores saiu da partida.

Figura 41 – Chamada RPC para um jogador que saiu da partida

```
GameObject[] goPlayers = GameObject.FindGameObjectsWithTag("Player");
foreach(GameObject disconnectedPlayer in goPlayers)
{
    if(disconnectedPlayer.GetComponent<FixedAttributes>().getGUID() == player.guid)
    {
        networkView.RPC("killPlayer", RPCMode.AllBuffered, disconnectedPlayer.networkView.viewID);
    }
}
```

Fonte: Elaboração do autor (2013).

5.4.5.2 Gerenciador de conversa entre jogadores

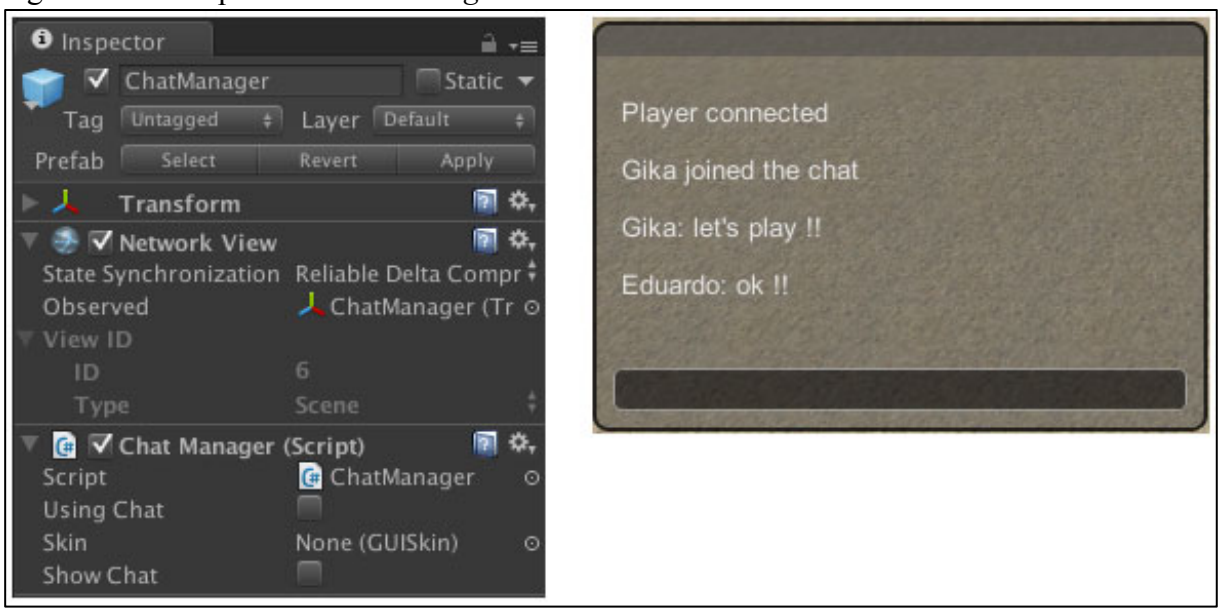
Nome: *ChatManager*

Descrição: este componente está presente na camada de *View* da aplicação por ter a responsabilidade de mostrar a conversa entre os jogadores no modo *multiplayer*.

Função: gerenciar toda a GUI da conversa. Ele recebe os eventos do componente *NetworkManager* para saber quando um jogador entrou ou saiu de uma partida, mostrando para todos os envolvidos essas notificações. Além disso, recebe os comandos digitados no *chat* e reproduz pra todos jogadores.

Aplicabilidade: possui o *script*: "*ChatManager.cs*" que é anexado a um *GameObject* e precisa estar presente na cena durante todo o jogo. Além do *script*, ele precisa ter o componente nativo da *engine* "*Network View*", configurado como "*Reliable Delta Compressed*" para garantir que as mensagens cheguem a todos os participantes. A Figura 42 mostra como o componente fica dentro da *Unity 3D* e a tela de *chat*.

Figura 42 – Componente *ChatManager*



Fonte: Elaboração do autor (2013).

Dependências: ele necessita do componente "*NetworkManager*", para receber os eventos de jogadores conectando e desconectando da partida.

Considerações sobre a codificação: o *script* "*ChatManager.cs*" possui dentro dele duas classes responsáveis por guardar o nome e o componente *NetworkPlayer* de cada jogador conectado, e a cada frase escrita e enviada para o *chat*, instanciar uma nova entrada de dados com o nome do jogador e a sua frase. A Figura 43 mostra essas duas classes.

Figura 43 – Classes dentro do *script ChatManager.cs*

```
public class PlayerNode
{
    public string _playerName;
    public NetworkPlayer networkPlayer;
}

public class ChatEntry
{
    public string name = "";
    public string text = "";
}
```

Fonte: Elaboração do autor (2013).

5.4.6 Componente externo à *Unity 3D*

Este componente é caracterizado por gerenciar todos dados usados pelo servidor. É um grupo de *scripts* da camada *Model* da aplicação, ou seja, modelam os dados e o comportamento por atrás do processo, e se comunicam com componentes específicos dentro da *engine*. Ele é dividido em 5 grupos: *UserConnection*, *UserInGame*, *UserItens*, *FacebookUser* e *ServerStructure*, cada um com características particulares que serão descritas a seguir.

5.4.6.1 Gerenciador de conexão do jogador

Nome: *UserConnection*

Descrição: este componente é utilizado no início do jogo para avisar à *engine* se o jogador já está cadastrado ou não no banco de dados.

Função: gerenciar o início do jogo se comunicando com o banco de dados. Possui duas respostas que afetam o fluxo do jogo:

a) se o jogador é novo no jogo, o sistema cadastra esse usuário com o identificador e o nome utilizado no Facebook.
b) se o jogador é cadastrado, porém não possui um avatar, envia uma resposta para a *Unity 3D* abrir a tela de escolha de personagem. Ao ser escolhido, salva no banco de dados os valores vindos da *Unity 3D* e cria toda a estrutura no banco de dados para esse novo personagem, como seus atributos e seu inventário de itens.
c) se o jogador possui um personagem, simplesmente retorna para a *Unity 3D* os atributos dele para que seja instanciado no jogo.

Aplicabilidade: possui os *scripts*: "*connectUser.php*", "*searchCharacter.php*" e "*saveCharacter.php*" que precisam estar no servidor. Na *Unity 3D*, eles são referenciados

através do componente *PlayerManager*, porém, é o componente *DBConnection* que se comunica diretamente com esses *scripts*.

Dependências: ele necessita do componente em PHP que se conecta com o banco de dados e dos dados do jogador vindo da *Unity 3D*.

5.4.6.2 Gerenciador de atributos do jogador

Nome: *UserInGame*

Descrição: este componente é utilizado durante todo o jogo para fazer a gerência dos dados do jogador com o banco de dados.

Função: gerenciar do inicio ao fim do jogo a comunicação e persistência dos valores de atributos do jogador no banco de dados. Possui as seguinte responsabilidades:

a) salvar as estatísticas e/ou atributos do jogador.
b) salvar a quantidade de vida e mana do jogador.
c) salvar a experiência e o nível do jogador.
d) salvar o dinheiro que o jogador possui.
e) salvar os novos valores de experiência e dinheiro quando o jogador morre na partida, pois eles são decrementados segundo as regras de *Game Design* do projeto. E salvar com o valor total a vida e a mana.

Aplicabilidade: possui os *scripts*: "*saveStats.php*", "*saveLife.php*", "*saveMana.php*", "*saveExp.php*", "*saveLevel.php*", "*saveGold.php*" e "*saveDead.php*" que precisam estar no servidor. O *script* "*saveStats.php*" está relacionado ao componente *PlayerStats* e os outros estão relacionados com o componente *PlayerManager*. Na *Unity 3D*, eles são referenciados através do componente *PlayerManager*, porém, é o componente *DBConnection* que se comunica diretamente com esses *scripts*.

Dependências: ele necessita do componente em PHP que se conecta com o banco de dados e dos dados do jogador vindo da *Unity 3D*.

5.4.6.3 Gerenciador dos itens do jogador

Nome: *UserItens*

Descrição: este componente é utilizado durante todo o jogo para fazer a gerência dos itens do jogador com o banco de dados.

Função: gerenciar do inicio ao fim do jogo a comunicação e persistência dos valores de itens do jogador no banco de dados. Possui as seguintes responsabilidades:

a) possui um arquivo no formato JSON com a especificação de cada item. Todo item que for adicionado ao jogo, precisa estar nesse arquivo para que o jogo saiba seus valores, por exemplo, de preço, ganhos em atributos etc.
b) gerar os itens aleatoriamente e enviar para a *engine* qual foi o escolhido.
c) gerenciar e salvar no inventário do jogador os itens coletados.
d) remover do banco de dados os itens que forem descartados pelo jogador no seu inventário.
e) salvar no banco de dados os itens que estão equipados no avatar do jogador.
f) retornar para a *engine* a lista de todos os itens que o jogador possui.

Aplicabilidade: possui os *scripts* que se relacionam com o inventário do jogador: "*saveItemInventory.php*", "*searchItens.php*", "*swapItens.php*" e "*dropItens.php*". Possui também o *script* que se relaciona com o componente *RandomItem*: "*randomItem.php*". E por fim o arquivo JSON "*itens.json*". Todos esses *scripts* precisam estar no servidor. Na *Unity 3D*, eles são referenciados através do componente *PlayerManager*, porém, é o componente *DBConnection* que se comunica diretamente com esses *scripts*.

Dependências: ele necessita do componente em PHP que se conecta com o banco de dados, dos dados do jogador vindo da *Unity 3D* e dos *scripts* que transformam os dados do

arquivo JSON para dados que o PHP consiga interpretar (presente nos componentes estruturais).

5.4.6.4 Gerenciador da comunicação com o Facebook

Nome: *FacebookConnection*

Descrição: este componente é utilizado durante todo o jogo para fazer a gerência das ações do jogador com o Facebook.

Função: gerenciar do inicio ao fim do jogo a comunicação do jogador com o Facebook. Este componente possui a responsabilidade da comunicação entre todas as chamadas externas da *Unity 3D* com o *browser* (navegador que interage com os documentos virtuais da internet), através de um comando dentro da *engine* (*Application.ExternalCall*) que é recebido por esse componente e assim fazer as chamadas de Facebook.

O primeiro passo que acontece no jogo, é verificar se o usuário está conectado ao Facebook, e é através desse componente que isso é feito. Além disso, esse componente é o único ponto de comunicação do jogo com o Facebook, e dele, são enviados os dados e ações a serem tomadas. Possui as responsabilidades de pegar os dados do usuário e de seus amigos no Facebook, convidar pessoas para jogarem o jogo e compartilhar conteúdos do jogo.

Aplicabilidade: possui apenas o *script* "*user.js*" que se relaciona com o componente *PlayerManager* e precisa estar no servidor.

Dependências: ele necessita do componente estrutural em PHP, que é a página principal do jogo no *browser*, para que a API do Facebook seja inicializada.

5.4.6.5 Gerenciador estrutural

Nome: *ServerStructure*

Descrição: este componente é composto por recursos estruturais da aplicação.

Função: este componente é responsável por apresentar o jogo no *browser*, além de interpretar os arquivos JSON, inicializar a API do Facebook e conectar a aplicação com o banco de dados. Nele também está presente o arquivo de configuração em XML *Crossdomain*, que é um meio de garantir que as informações fornecidas entre dois ou mais domínios de segurança diferentes, tenham a capacidade de serem acessadas ou transferidas na internet.

Aplicabilidade: possui o *script* "*index.php*", que precisa estar no servidor, que apresenta toda a página do jogo na internet, também inicializando a API do Facebook e o *player* da *Unity 3D* no *browser*. O *script* "*crossdomain.xml*" precisa estar no mesmo nível de pastas do "*index.php*" no servidor. O *script* "*readJsonFiles.php*" para interpretar os arquivos JSON e o *script* "*connect.php*", que possui as configurações para acessar o banco de dados, e efetivamente o conecta a ele.

Dependências: ele necessita da pasta com o SDK do Facebook para poder fazer a comunicação entre o jogo e a rede social, além do arquivo gerado pela *Unity 3D*, no formato "*.unity3d*", para que ele possa carregar o jogo no *browser*. Necessita também que exista um banco de dados MySQL para que ele possa fazer a conexão com esse banco.

5.4.7 Resumo dos componentes

O Quadro 6 mostra o resumo de todos os componentes, com seus *scripts* envolvidos e as ligações entre eles.

Quadro 6 – Resumo dos componentes

Nome	*Scripts*	Ligação
PlayerManager	*PlayerManager.cs* *CharacterChoice.cs*	*DbCommunication* *UserConnection* *UserInGame*
PlayerStats	*PlayerStats.cs* *Levels.cs*	*PlayerManager* *DbCommunication*
RandomItem	*Item.cs* *RandomItemActions.cs* *CollectItem.cs*	*DbCommunication* *UserItens*
EnemyManager	*EnemyAI.cs* *EnemyHurt.cs* *SpawnEnemies.cs*	*PlayerObject* *PlayerStats*
DbCommunication	*DbCommunication.cs* *ServerAddresses.cs*	*UserConnection* *UserInGame* *UserItens*
NetworkManager	*NetworkManager.cs*	*PlayerManager*
PlayerObject	*PlayerActions.cs* *FixedAttributes.cs*	*EnemyManager* *PlayerManager*
PlayerInventory	*PlayerInventory.cs*	*DbCommunication* *UserItens*
FacebookUI	*FacebookUI.cs*	*FacebookConnection*
ChatManager	*ChatManager.cs*	*NetworkManager*
UserConnection	*connectUser.php* *searchCharacter.php* *saveCharacter.php*	*DbCommunication* *PlayerManager* *ServerStructure*
UserInGame	*saveStats.php* *saveLife.php* *saveMana.php* *saveExp.php* *saveLevel.php* *saveGold.php* *saveDead.php*	*DbCommunication* *PlayerManager* *ServerStructure*

UserItens	*itens.json* *randomItem.php* *dropItens.php* *swapItens.php* *searchItens.php* *saveItemInventory.php*	*DbCommunication* *PlayerInventory* *RandomItem* *ServerStructure*
FacebookConnection	*user.js*	*FacebookUI* *ServerStructure*
ServerStructure	*FacebookAPI* *index.php* *crossdomain.xml* *readJsonFiles.php* *connect.php*	*UserConnection* *UserInGame* *UserItens* *FacebookConnection*

Fonte: Elaboração do autor (2013).

As ligações desses componentes fazem com que o jogo funcione como um todo. É importante observar que essa dependência ocorre pelo fato da aplicação possuir comunicação externa entre a *engine* com o banco de dados e o Facebook, além de as funcionalidades dentro dela serem divididas por tipo, fazendo com que, por exemplo, o inimigo precise de um jogador para poder ter seus atributos calculados e suas ações acontecerem dentro do jogo.

5.5 CICLOS DE TESTES

Os testes foram realizados conforme cada componente foi sendo desenvolvido, tornando um modelo iterativo de criação dos componentes. Esses testes também ocorreram conforme a ligação entre dois componentes finalizados, testando a sua integração e ajustando as suas dependências.

Na parte de jogabilidade, os testes foram feitos conforme o jogo foi se tornando mais completo e seus ajustes se tornando mais perceptíveis, como por exemplo, a movimentação do jogador, que precisava ser polida conforme as necessidades que o jogo apresentava.

Os testes com o Facebook precisam ser feitos dentro da rede social. Como a aplicação está configurada em modo *sandbox*, foi necessário adicionar usuários de teste do aplicativo. Assim, quando um convite ou um compartilhamento é feito, os resultados dessas ações não aparecem públicas dentro do Facebook, e sim no perfil particular dos usuários de teste.

É extremamente importante manter a integridade dos dados. Com isso, os testes de banco de dados foram feitos conforme os componentes de *server side* terminavam de ser desenvolvidos. Foram feitos também, testes de carga com grande número de requisições ao banco de dados, para determinar como o jogo iria se comportar. No final dos testes, o resultado foi satisfatório.

Em relação ao *multiplayer*, foram instaladas as aplicações em usuários de teste utilizando diferentes conexões com a internet. Conforme uma pior conexão, foi detectado maior latência, gerando um atraso na sincronia do jogo. Porém, com uma conexão normal, o jogo aconteceu normalmente, tanto em sincronia como nas funcionalidades de manter os estados do jogo iguais para todos os jogadores envolvidos.

5.6 APRESENTAÇÃO DO PROTÓTIPO

Esta sessão apresenta o protótipo do jogo utilizando os componentes descritos no item anterior. Serão expostas imagens do jogo em funcionamento, com a descrição das funcionalidades, fazendo uma ligação com os componentes que estão sendo usados para gerar as ações dentro do jogo.

O jogo começa com a escolha do personagem. São 3 tipos de personagens com diferentes tipos de habilidades. Nesse momento, o componente *PlayerManager* faz uma chamada para a API do Facebook requisitando o id do usuário. No retorno dessa chamada, o componente *DbCommunication* é acionado para buscar no banco de dados se o usuário possui ou não um personagem criado. No caso de não possuir, é apresentada a opção de escolha conforme a Figura 44.

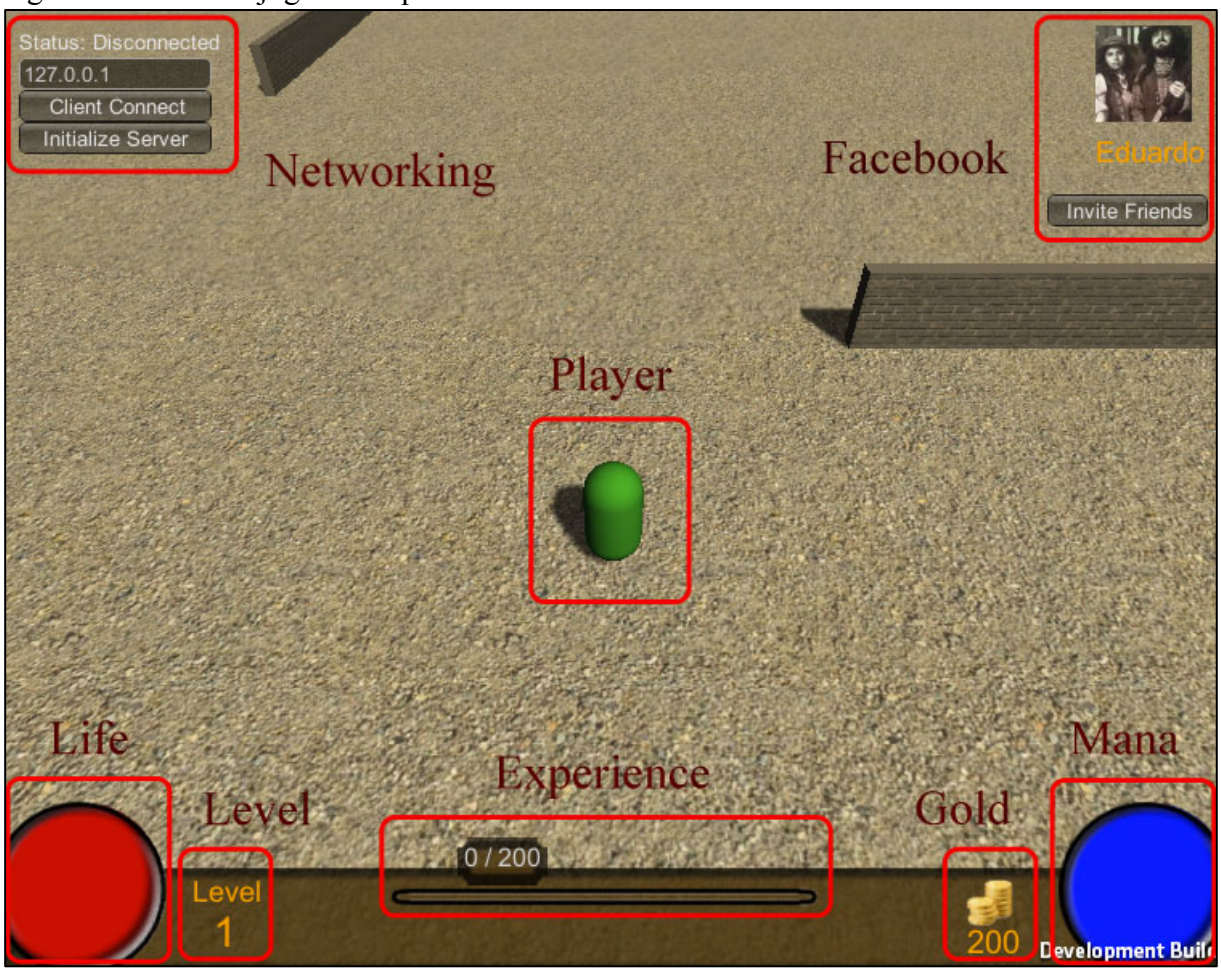
Figura 44 – Escolha do personagem

Fonte: Elaboração do autor (2013).

Uma vez escolhido o personagem, o componente *PlayerManager* faz uma chamada para o componente *DbCommunication*, que envia os dados para o componente *UserConnection* salvar as informações no banco de dados. Depois disso, a cena é criada e o personagem é instanciado para começar a jogar.

As opções que o jogador tem para interagir com a GUI inicial da cena são: convidar amigos do Facebook para conhecerem o jogo e iniciar uma partida *multiplayer*. Além disso, a GUI mostra o quanto de vida e mana o jogador possui, a quantidade de dinheiro, o nível e a barra de experiência. Ao passar o *mouse* sobre a barra de experiência, é possível visualizar quanto falta para alcançar um novo nível no jogo. A Figura 45 mostra como é a tela de jogo e os componentes da GUI.

Figura 45 – Tela de jogo e componentes da GUI



Fonte: Elaboração do autor (2013).

Ao jogador clicar na tecla “S” do teclado, o componente *PlayerStats* é acionado, e as suas estatísticas são mostradas com as informações de força, destreza, inteligência, vitalidade, dano e defesa. Além disso, cada vez que ele passa de nível, cinco pontos são dados para poderem ser somados nessas estatísticas, da forma que o jogador quiser escolher. A Figura 46 mostra a tela de estatísticas do jogador.

Figura 46 – Tela de estatísticas do jogo

Fonte: Elaboração do autor (2013).

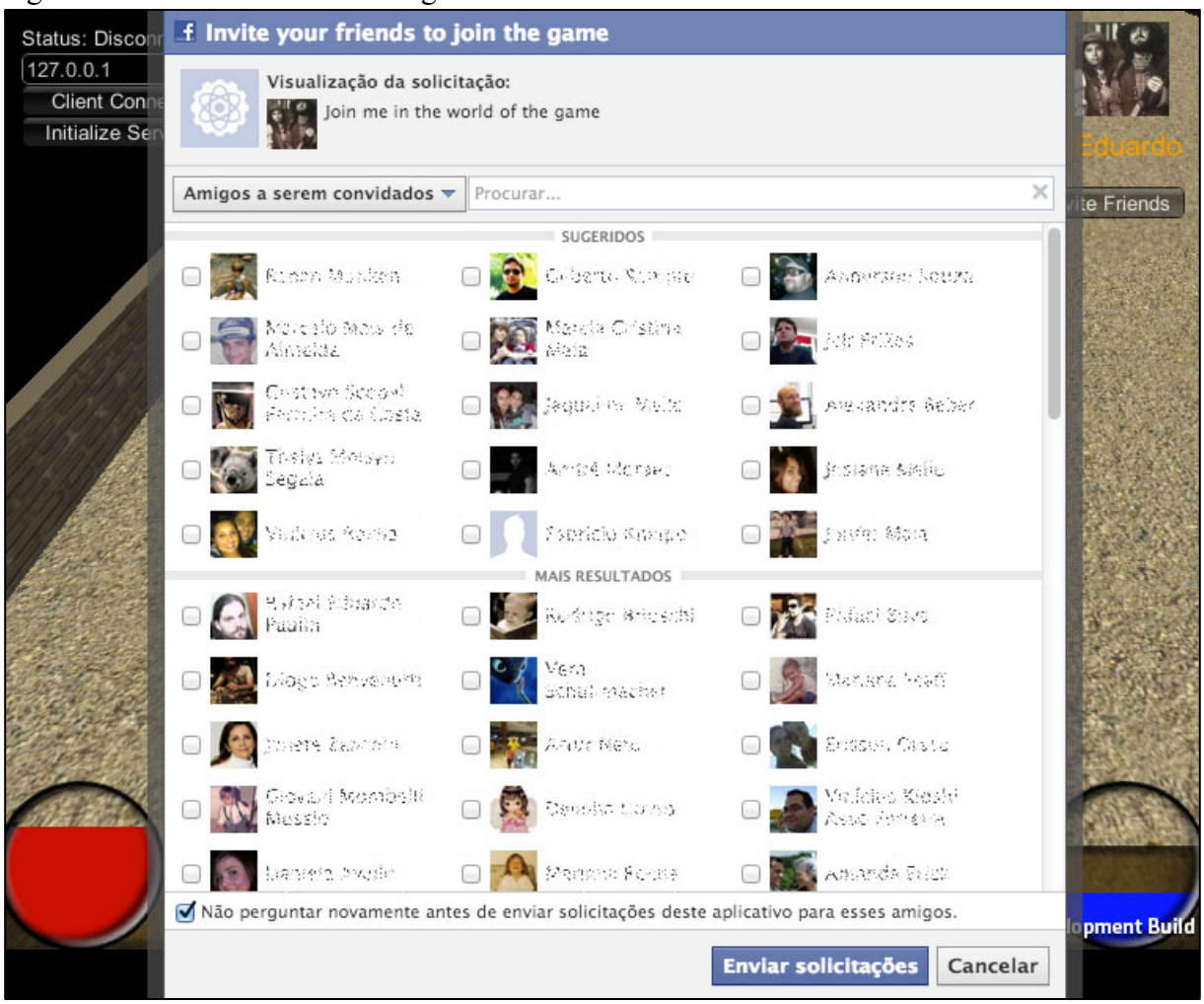
Ao jogador clicar na tecla “I” do teclado, o componente *PlayerInventory* é acionado para chamar, através dos componentes que ele depende, o banco de dados com todos os itens que o jogador possui. Nessa tela, o jogador pode equipar seu personagem com itens ou descartar algum item que não lhe interesse. Além disso, os itens de vida e mana podem ser usados para recuperar o que foi perdido enfrentando algum inimigo. A Figura 47 mostra a tela de inventário do jogador.

Figura 47 – Tela de inventário do jogo

Fonte: Elaboração do autor (2013).

Ao clicar no botão "*Invite Friends*", o componente *FacebookUI* é acionado para fazer uma chamada para a API do Facebook e, assim, mostrar a tela com seus amigos para que possa ser enviado o convite. A Figura 48 mostra essa tela.

Figura 48 – Tela de convidar amigos do Facebook



Fonte: Elaboração do autor (2013).

O objetivo do jogo é vencer todos os inimigos do cenário e conquistar itens, dinheiro e nível. Com isso, os inimigos se tornam mais difíceis e os cenários mais complicados. Para confrontar os inimigos, o jogador precisa apenas clicar com o botão esquerdo do *mouse* em cima do inimigo, para um ataque normal, ou com o botão direito, para um ataque mais forte, que resulta em uma diminuição de mana ao acertá-lo. Nesse momento, os componentes *PlayerObject* e *EnemyManager* estão sendo acionados para que a batalha ocorra.

Quando um inimigo é morto, ele gera de forma randômica um item para o jogador. Se o jogador morrer, ele volta para o início da fase, perdendo dinheiro e experiência, porém, o estado do jogo permanece, ou seja, possui os mesmos inimigos e itens nos mesmo lugares antes da sua morte. A Figura 49 mostra um confronto entre o jogador e os inimigos.

Figura 49 – Combate entre jogador e inimigos

Fonte: Elaboração do autor (2013).

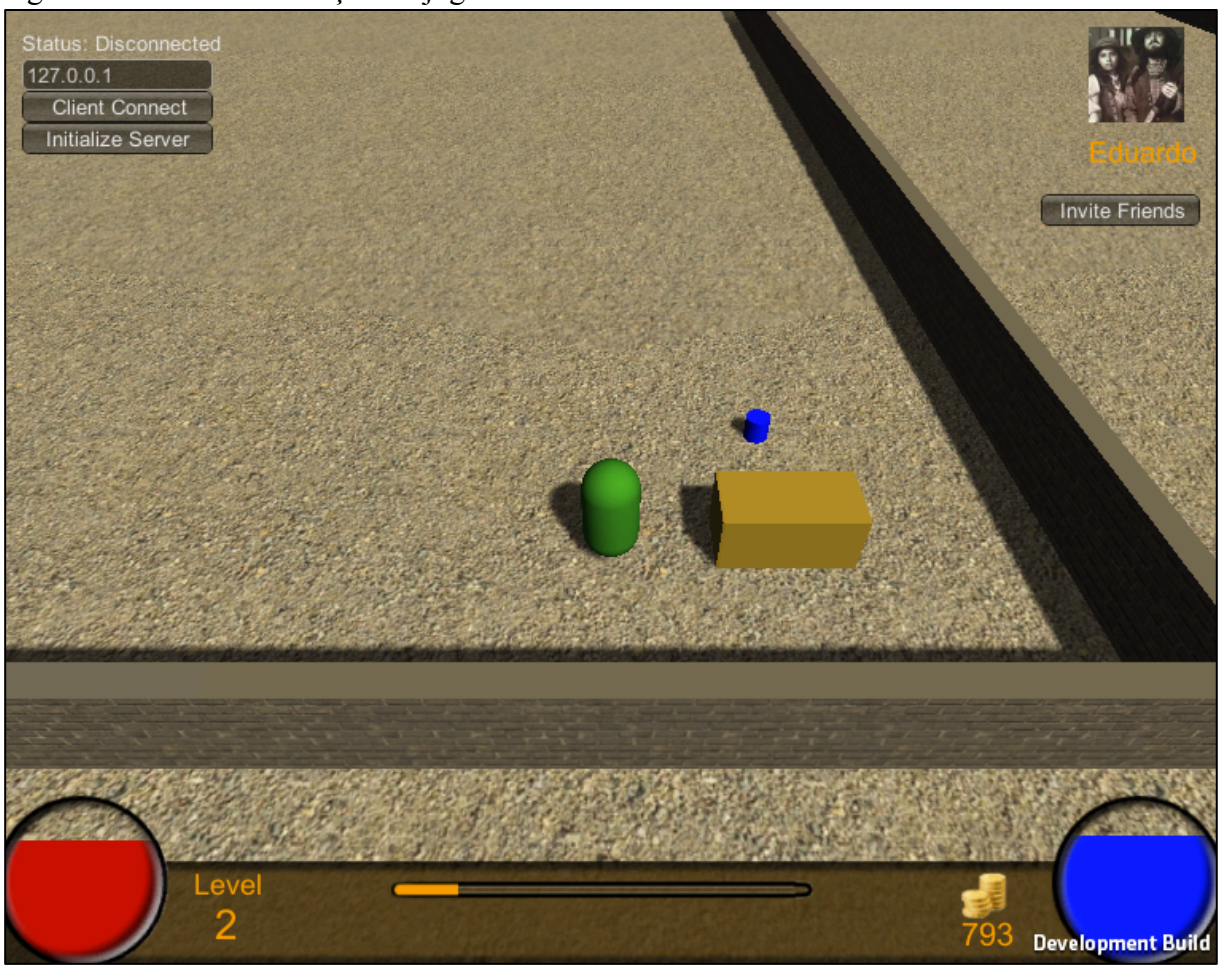
Ao passar de nível, o jogador recebe uma notificação para compartilhar no seu Facebook que atingiu um novo nível. Nesse momento, o componente *FacebookUI* faz uma chamada para a API do Facebook, requisitando a tela de compartilhamento e, em seguida, preenche os dados definidos pelo jogo e permite o jogador escrever alguma coisa sobre esse momento. A Figura 50 mostra como é essa interação entre o jogador e o Facebook.

Figura 50 – Tela de compartilhar no Facebook

Fonte: Elaboração do autor (2013).

Além do inimigo, que quando morre gera itens ou dinheiro para o jogador, existem os baús (*chests*) que estão espalhados pela cena. Ao serem clicados pelo jogador, geram um item aleatório, do qual são acionados pelo componente *RandomItem*. A Figura 51 mostra a interação do jogador com o baú.

Figura 51 – Tela de interação do jogador e um baú na cena



Fonte: Elaboração do autor (2013).

Outra opção que o jogador possui, é iniciar uma partida *multiplayer* com um amigo. Nessa opção, o jogador que se torna o servidor, é quem comanda o jogo. O seu cenário é mantido exatamente como estava. Os jogadores que se conectarem a ele, entrarão para ajudá-lo a conquistar o objetivo do jogo. Nesse momento, o componente *NetworkManager* é acionado para gerenciar toda a partida. Além disso, os participantes desse jogo, tem a opção de conversarem em tempo real, podendo traçar estratégias de ataque ou até falar sobre qualquer outra coisa. Nesse ponto, o componente *ChatManager* gerencia a conversa. A Figura 52 mostra a tela de interação entre os jogadores.

Figura 52 – Tela de interação entre jogadores em uma partida *multiplayer*

Fonte: Elaboração do autor (2013).

5.7 VALIDAÇÃO

Segundo SOMMERVILLE (2011), a validação é um mecanismo utilizado para assegurar que um processo, ao qual se submete um programa, cumpre com as suas especificações. Esse processo costuma ser no final da etapa de desenvolvimento, com a intenção de confirmar que o aplicativo esteja em condições de desenvolver suas tarefas.

Foram convidados três programadores que trabalham diariamente com o desenvolvimentos de jogos utilizando tanto a *Unity 3D* como a API do Facebook. Dois deles são formados em Ciência da Computação, o terceiro está no último semestre do curso de Jogos Digitais. Os três avaliadores são jogadores ativos de vários tipos de plataformas.

A validação teve como objetivo perceber a facilidade de se usar os componentes desenvolvidos, além de verificar se as funcionalidades e as decisões tomadas em relação à elas foram assertivas.

Inicialmente, foi feita uma explanação sobre o projeto como um todo, mostrando seus objetivos. Na sequência, foram explanadas as funcionalidades para que os usuários soubessem qual a finalidade dos componentes.

A partir dessa explanação, os avaliadores foram convidados, individualmente, a fazer uso dos componentes, com uma cena pronta dentro da *Unity 3D*, para que os mesmos pudessem usar e alterar valores de cada uma das funcionalidades dos componentes.

5.7.1 Apresentação dos resultados da validação

Cada usuário testou e avaliou as funcionalidades do sistema. Seguem os quesitos avaliados e os comentários correspondentes a cada um:

a) quanto à necessidade do sistema, os três usuários acharam que a necessidade é pertinente, pois a falta desses componentes, gera um retrabalho a cada projeto, devido a falta de padronização e de se ter uma base sólida para fazer apenas as adaptações dessas funcionalidades para outros projetos.
b) quanto à capacidade de alteração dos códigos para se adaptarem a novos projetos, a opinião do usuário 1 e 2 é que está simples de se entender, porém, o usuário 3, acha que o desenvolvedor precisa ter um conhecimento intermediário na *engine*, por utilizar componentes de *network*, banco de dados e *game design*, onde ele teve dificuldade de entender como utilizar e alterar o componente de *network*.
c) quanto ao *gameplay*, os três usuários comentaram que está de boa qualidade, tornando-se divertido por poder jogar com outros amigos, trocar itens, conversar pelo *chat* e interagir com o Facebook suas conquistas durante a partida.

Após a validação feita, os três usuários foram convidados a fazer um pequeno debate sobre melhorias e novas funcionalidades que pudessem ser adicionadas ao sistema.

Nesse debate, todos concordaram que, apesar de o jogo possuir uma interface gráfica simples, ele é divertido, e acreditam que, com essa base de componentes pronta, o trabalho de um artista tornaria o jogo bastante atrativo em pouco tempo, para que o jogo pudesse ser lançado oficialmente no Facebook.

O avaliador 1 ainda contribuiu com uma sugestão de implementação futura para o sistema: a possibilidade de criar um sistema de leilão de itens, que seria uma forma de jogadores interagirem dentro do jogo, assim, conseguindo ganhar mais dinheiro e itens diferentes. Os avaliadores 2 e 3 sugeriram que, com a evolução dos componentes, podia ser criado um componente para tratar de outros tipos de ataques do jogador, como um grupo de ataques especiais e com atributos de magias.

6 CONCLUSÕES E TRABALHOS FUTUROS

Esta monografia apresenta o desenvolvimento de componentes para facilitar a criação de um jogo 3D *multiplayer* básico e em tempo real para a rede social Facebook. Para isto, foi criado um protótipo, a fim de demonstrar a aplicação desses componentes. A sua principal contribuição encontra-se no suporte que estes componentes podem proporcionar, com a diminuição do tempo gasto com o desenvolvimento dessas funcionalidades, além de poderem ser aproveitados em vários projetos.

O objetivo principal deste projeto é diminuir o tempo gasto pelos profissionais que trabalham com desenvolvimento de jogos utilizando a *Unity 3D* conectada ao Facebook e com a possibilidade de ser *multiplayer*. Foi atingido esse objetivo com o desenvolvimento dos componentes apresentados no Capítulo 5. Através de testes realizados durante e após o desenvolvimento, percebeu-se que se leva um maior tempo para pensar nas soluções desses componentes, junto com o desenvolvimento e os testes. Um tempo considerável é gasto para criar as funcionalidades que um jogo precisa ter, e com os componentes desenvolvidos, parte de um jogo está pronto, precisando apenas pequenos ajustes para cada tipo de jogabilidade e definições de *game design*.

Acredita-se que esses componentes poderão servir de apoio a estudantes e a profissionais que queiram criar projetos de jogos desenvolvidos na *Unity 3D* e conectados ao Facebook. À partir de um sistema pronto é mais fácil compreender as funcionalidades que um jogo precisa ter, pois o código-fonte pode ser estudado, e podem ser realizadas alterações e testes para analisar o seu comportamento. Assim, espera-se que este projeto possa se tornar uma referência de estudos e exemplo para jogos a serem criados.

Com este projeto, procurou-se fazer um protótipo que facilite no entendimento dos desenvolvedores na utilização dos componentes. Essa cena de exemplo possui toda a estrutura que deve ser adotada para a criação do jogo e está pronta para ser usada como um jogo, precisando apenas de melhorias de arte.

Os principais resultados obtidos neste trabalho são a agilidade na criação de jogos, facilidade de alterar as regras do jogo, assim como, adicionar funcionalidades, proporcionando aos leitores uma visão geral das mesmas e a geração de códigos padronizados para se obter um projeto organizado e de fácil reaproveitamento.

Algumas características que se destacam no projeto são: ser um conector entre a *Unity 3D* e o Facebook, a simplicidade de se configurar as características de uma cena e de

seus objetos em um jogo e a facilidade de adicionar a opção de objetos serem configurados como *multiplayer*. Além disso, é importante salientar que o sistema possui um banco de dados próprio e simples, de fácil alteração na sua estrutura, ou seja, uma tarefa a menos para o desenvolvedor, que não precisa fazer configuração alguma de banco de dados se não desejar, podendo apenas configurar a cena e ter o jogo pronto.

Um ponto de dificuldade no desenvolvimento dos componentes foi a criação do sistema de inventário do jogador. Esse sistema precisa ser seguro o bastante para não prejudicar o jogador, perdendo ou obtendo itens sem a sua ação. Para isso, foi feita uma sincronização dos dados, no momento que uma alteração é feita no inventário. O sistema precisa requisitar o banco de dados e retornar a resposta, para depois atualizar o acontecimento para o jogador.

Como possibilidade de continuação deste projeto em trabalhos futuros, seguem algumas sugestões:

a) finalizar o jogo, tendo todos os módulos de missões;
b) tornar o jogo um *multiplayer* massivo, integrando com um servidor externo dedicado;
c) adicionar nos componentes a funcionalidade de compra e venda de itens do jogador;
d) adicionar nos componentes a funcionalidade de mini-mapa, para ajudar o jogador a se localizar dentro do cenário;
e) adicionar nos componentes a funcionalidade de novos atributos mágicos, como ataques e itens;
f) aumentar a possibilidade de teclas de atalho para diferentes tipos de funcionalidades;
g) finalizar a arte do jogo com modelos 3D e a criação de terrenos diversos;
h) elaborar uma documentação e um manual do sistema, para que possa ser uma referência de estudos e exemplo para jogos sociais utilizando a *Unity 3D* e o *multiplayer*.

REFERÊNCIAS

ACIOLI, Sonia. Redes sociais e teoria social: revendo os fundamentos dos conceitos. **Informação & Informação**, Londrina, v. 12, n. esp., 2007.

ADAMS, Ernest. **Fundamentals of Game Design** (2nd Edition). California: New Riders, 2009.

BATTAIOLA, André L. Jogos por Computador – Histórico, Relevância Tecnológica e Mercadológica, Tendências e Técnicas de Implementação In: **XIX Jornada de Atualização em Informática**. Curitiba: SBC, Julho/2000, v. 2. pp. 83-122.

BENEVENUTO, Fabrício; ALMEIDA, Jussara M.; SILVA, Altigran S.. **Explorando Redes Sociais Online: Da Coleta e Análise de Grandes Bases de Dados às Aplicações**. 2011. Departamento de Ciência da Computação Universidade Federal de Ouro Preto, Ouro Preto, 2011.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Campus, 2002.

BLACKMAN, Sue. **Beginning 3D Game Development with Unity**. New York: Apress, 2011.

CASTELLS, Manuel. **A Sociedade em rede**. 3ª ed.V.1. São Paulo: Paz e Terra, 1999.

CASTRO, Gisela G. S. Entretenimento, sociabilidade e consumo nas redes sociais: cativando o consumidor-fã. Artigo da revista **Fronteiras**, Vol. 14, nº 2, maio/agosto, 2012.

CLUA, Esteban Walter Gonzalez; BITTENCOURT, João Ricardo. **Desenvolvimento de Jogos 3D: Concepção, Design e Programação**. 2005. Centro de Ciências Exatas e Tecnológicas Universidade do Vale do Rio dos Sinos (UNISINOS).

CRAWFORD, Chris. **The Art Of Computer Game Design: Reflections Of A Master Game Designer**. Naperville: Osborne/McGraw-Hill, 1984.

DÉDA, Talita; ZAGALO, Nelson. **Funcionalidades dos Jogos Sociais**. 2010. Universidade do Minho, Braga, Portugal. Disponível em: <http://repositorium.sdum.uminho.pt/bitstream/1822/13430/1/968-3398-1-PB%281%29.pdf>. Acesso em 23 de abril de 2013.

DEMASI, Pedro. **Desenvolvimento de Jogos**. 2005. Disponível em: <http://www.nce.ufrj.br/conceito/artigos/2005/08-2.htm>. Acesso em: 19 abril de 2013.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de Banco de Dados**. 6ª ed. São Paulo: Pearson Education do Brasil, 2011.

FIGUEIREDO, Giovanna Santos. **As redes sociais na era da comunicação interativa**. 2009. 66 f. Monografia (Graduação em Comunicação Social) - Departamento de Universidade Católica de Pernambuco, Recife, 2009.

FOWLER, Martin. **UML Essencial: um breve guia para a linguagem padrão de modelagem de objetos**. 3ª ed. Porto Alegre: Bookman, 2005.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software**. Indianapolis: Pearson, 1998.

GAMEREPORTER. Disponível em: <http://gamereporter.uol.com.br/paypal-diz-que-12-milhoes-de-clientes-pagam-para-jogar-no-facebook/>, acesso em 17 de abril de 2013.

GIL, Antonio Carlos. **Métodos e Técnicas de Pesquisa Social**. 6ª ed. São Paulo: Atlas, 2008.

GRAHAM, Wayne. **Facebook API Developers Guide**. New York: First Press, 2008.

JUUL, Jesper. **A Casual Revolution: Reinventing Video Games and Their Players**. London: The Mit Press, 2012.

KIRKPATRICK, David. **The Facebook Effect: The Inside Story of the Company That Is Connecting the World**. Australia: Simon & Schuster, 2011.

MACHADO, Joicemegue Ribeiro; TIJIBOY, Ana Vilma. **Redes Sociais Virtuais: um espaço para efetivação da aprendizagem cooperativa**. 2005. CINTED – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2005.

MARTELETO, Regina Maria. **Análise de redes sociais – aplicação nos estudos de transferência da informação**. Ciência da Informação, Brasília, v. 30, n. 1, p. 71-81, jan./abr. 2001

MORIMOTO, Carlos E. **UDP**. 2005. Disponível em: <http://www.hardware.com.br/termos/udp>. Acesso em: 21 de abril de 2013.

PINGDOM. Disponível em: <http://royal.pingdom.com/2013/01/16/internet-2012-in-numbers/>, acesso em 17 de abril de 2013.

RECUERO, Raquel. **Redes Sociais na Internet**. Porto Alegre: Sulina, 2009.

ROLLINGS, Andrew; MORRIS, Dave. **Game Architecture and Design: A New Edition**. Indiana: New Riders Publishing, 2004.

SILVA, A.; FERREIRA, M. **Gestão do conhecimento e capital social: as redes e sua importância para as empresas**. Informação & Informação, Londrina, v. 12, n. esp., 2007.

SILVA, E. L. da; MENEZES, E. M. **Metodologia da pesquisa e elaboração de dissertação**. 4ª ed. Florianópolis: Laboratório de Ensino à Distância da UFSC, 2005.

SOMMERVILLE, Ian. **Engenharia de Software**. 9ª ed. São Paulo: Pearson Education, 2011.

STATISTICBRAIN. Disponível em: <http://www.statisticbrain.com/facebook-statistics/>, acesso em 15 de agosto de 2013.

TANENBAUM, Andrew S. **Computer networks**. Boston: Prentice Hall PTR, 2003.

WASSERMAN, S.; FAUST, K. **Social Network Analysis: Methods and Applications**. Cambridge, UK: Cambridge University Press, 1994.