



CENTRO UNIVERSITÁRIO
INSTITUTO DE EDUCAÇÃO SUPERIOR DE BRASÍLIA – IESB
ESPECIALIZAÇÃO EM DISPOSITIVOS MOVEIS

RAFAEL BRASILEIRO DE ARAUJO

**IMPLEMENTAÇÃO E OTIMIZAÇÃO DE JOGOS MULTIJOGADORES EM
PLATAFORMAS ANDROID**

Um Estudo de Caso do Jogo Tales of Unreal

Brasília-DF
2014

RAFAEL BRASILEIRO DE ARAUJO

**IMPLEMENTAÇÃO E OTIMIZAÇÃO DE JOGOS MULTIJOGADORES EM
PLATAFORMAS ANDROID**

Um Estudo de Caso do Jogo Tales of Unreal

Trabalho de Conclusão de Curso apresentado
ao curso de Especialização em Dispositivos
Moveis do Instituto de Educação Superior de
Brasília, como requisito parcial para obtenção
do título de Pós-Graduação.

Orientador: Prof. Kenniston Arraes Bonfim

Brasília – DF

2014

RAFAEL BRASILEIRO DE ARAUJO

**IMPLEMENTAÇÃO E OTIMIZAÇÃO DE JOGOS MULTIJOGADORES EM
PLATAFORMAS ANDROID**

Um Estudo de Caso do Jogo Tales of Unreal

Trabalho de Conclusão de Curso aprovado
pela Banca Examinadora com vistas à
obtenção do título de Especialização em
Dispositivos Móveis, do Instituto de Educação
Superior de Brasília.

Brasília, DF, 21 de novembro de 2014

DEDICATÓRIA

Dedico esse trabalho a todos aqueles que sonham em desenvolver jogos e para aqueles que, assim como eu, já estão nessa longa jornada de aprendizado e experiência.

AGRADECIMENTOS

Agradeço a todos que me apoiaram nessa jornada, principalmente minha família, amigos e professores. Agradeço à Deus, por ter a oportunidade de ter cursado essa especialização e ter tido embasamento para concluir este trabalho. Obrigado!

RESUMO

Jogos de multijogadores, os chamados MMORPGs, são bastante famosos nos dias de hoje. Com o advento das plataformas móveis, é inevitável que jogos desse gênero sejam levados para essas novas plataformas. O seguinte trabalho tem por objetivo mostrar formas simples e eficientes para uma plataforma de jogo MMO compatível com dispositivos Android e PC, mantendo a jogabilidade semelhante entre as plataformas, utilizando o motor de jogo Unreal Engine 4 e o servidor de aplicação Photon Server

Palavras-Chave: MMORPG, Android, Unreal Engine 4, Photon Server

ABSTRACT

Multiplayer games, specially MMORPGs, are famous nowadays. With the advent of mobile platforms, it is clear that those games will reach them soon. The present work will show simple and efficient forms for a MMO game platform, compatible with Android devices and PC, with the almost same gameplay and experience. The present platform use the power of Unreal Engine 4 and Photon Server to bring this experience.

Keywords: MMORPG, Android, Unreal Engine 4, Photon Server

LISTA DE ABREVIATURAS E SIGLAS

ADSL – Asymmetric Digital Subscriber Line
DOCSIS – Data Over Cable Service Interface Specification
MMOG – Massive Multiplayer Online Game
MMORPG – Massive Multiplayer Online Role Playing Game
MVC – Model View Controller
PNJ – Personagem Não-Jogador
UMTS – Universal Mobile Telecommunications System
HSPA – High Speed Packet Access
UE4 – Unreal Engine 4
TCP – Transmission Control Protocol
UDP – User Datagram Protocol
SDK – Software Development Kit
NDK – Native Development Kit
API – Application Programming Interface
FPS – Frames per second
CPU – Central Processor Unit
GPU – Graphics Processor Unit

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS.....	10
1 INTRODUÇÃO	13
1.1 Justificativa	13
1.2 Problemática.....	14
1.3 Objetivos.....	14
1.4 Fundamentação Teórica.....	15
2 Resultados Esperados	19
3 Projeto	20
3.1 Proposta	20
3.2 Tecnologias utilizadas	20
3.2.1 Tecnologias de desenvolvimento	20
3.2.2 Plataformas de testes e desenvolvimento	21
3.2.3 Estrutura do servidor	21
3.2.4 Estrutura do jogo	25
4 Resultados Obtidos	37
4.1 Resultado geral de desempenho.....	37
4.2 Resultado de desempenho na plataforma Android	37
5 Conclusões e trabalhos futuros.....	39
REFERÊNCIAS BIBLIOGRÁFICAS	40

1 INTRODUÇÃO

Um gênero de jogo bastante comum nos dias de hoje é o MMORPG (Massive Multiplayer Online Role Playing Game), onde diversos jogadores se conectam e podem ter a experiência de jogar juntos um mesmo jogo. O gênero é predominante em plataformas de computador, seja PC ou Mac. Com o advento do smartphone, muitos desenvolvedores estão dando preferência para plataformas móveis ao criar seus jogos, e nesse gênero, esse movimento não é diferente. Quando se fala de jogos online, a característica técnica principal é a rede que os comunica. Hoje em dia, existem redes de alta velocidade mas a história muda quando falamos de dispositivos móveis, onde a capacidade de rede atual é aquém à existente para plataformas maiores como os computadores e consoles de videogame. Ao criar um jogo do gênero, a disponibilidade e velocidade da rede é fator crucial para seu sucesso, fator que é limitante quando se fala de plataformas móveis. Mesmo em meio a dificuldades, muitos jogos do gênero são lançados para essas plataformas. Será estudado nesse trabalho, formas eficazes de se construir uma estrutura de servidor e cliente para esse tipo de jogo, de forma que a mesma experiência de jogo exista tanto nas plataformas PC e Mac quanto nas plataformas Android, que será o foco do projeto. Será feito um estudo de caso do jogo “Tales of Unreal”, jogo experimental que está sendo desenvolvido ao longo desse projeto.

1.1 Justificativa

O uso de redes móveis para jogos de múltiplos jogadores ainda requer estudo para determinar a melhor forma de utilização delas de modo eficiente. Além disso, a integração entre plataformas de computadores e plataformas móveis não é devidamente explorada devido às restrições existentes nas redes móveis e na própria plataforma móvel. Assim, existe uma grande oportunidade em desenvolver um meio de integrar essas plataformas em jogos desse tipo, o que será um grande benefício para a indústria de jogos, abrindo novas possibilidades de mercado. Este projeto pretende aproveitar dessa oportunidade, já que não existem pesquisas suficientes para resolver o problema proposto.

Atualmente já existem soluções para jogos multijogadores tanto para plataforma de computadores quanto para plataformas móveis, porém ainda não se tem de forma efetiva, soluções que integrem ambas as plataformas. Ao utilizar os métodos já existentes entre as plataformas, se permite desenvolver uma alternativa mais viável para essa integração, se baseando nas restrições e exigências de cada plataforma.

Desenvolver uma arquitetura de servidor que integre as plataformas móveis às plataformas de computadores auxiliará o gênero de jogos MMORPG, entre outros, a terem uma maior visibilidade no mercado, principalmente ao abranger o mercado de dispositivos móveis.

1.2 Problemática

Um jogo totalmente baseado em comunicação em rede precisa utilizar de forma eficiente os recursos oferecidos por esta. Essa eficiência deve ser maior quando o foco é alguma plataforma móvel como o sistema Android. Ainda não se tem uma forma eficiente para lidar tanto com plataforma móvel quanto com plataformas maiores. Uma correta e eficiente utilização das estruturas e capacidades da rede em dispositivos móveis para jogos desse tipo, ainda é área que permite amplo estudo.

Ao desenvolver um jogo do gênero MMORPG para o sistema Android, como é possível utilizar de forma eficiente a rede disponível em dispositivos desse sistema, sem comprometer a experiência de usuário nas plataformas PC e Mac?

1.3 Objetivos

O objetivo geral do projeto é explorar eficientes formas de utilizar a rede entre dispositivos Android e computadores, em um ambiente de jogo de múltiplos jogadores. Para isso, será desenvolvida uma forma de comunicação que se adeque tanto para computadores quanto para dispositivos móveis.

1.4 Fundamentação Teórica

Com o advento da internet e a necessidade de gerenciamento centralizado de informações, sistemas baseados em uma arquitetura cliente/servidor se tornaram um requisito para a tecnologia da informação. Em uma estrutura simplista, temos a separação da aplicação em camadas distintas. Vamos considerar as camadas mais clássicas, isto é, a arquitetura MVC. Segundo Reenskaug (1979), o propósito da arquitetura MVC é criar uma ponte entre o modelo mental do usuário e o modelo digital existente no computador. Para a arquitetura, temos três camadas essenciais, a de modelo, onde os dados são gerenciados pelo computador, i.e., armazenamento e persistência de dados, tratamento e conversão de dados para o sistema, entre outras atribuições. O acesso a essa camada deve ser feito prioritariamente pela próxima camada, a de controle. Nessa camada temos a gerência de regras de negócio do sistema que possibilitam ao usuário sua utilização. Após isso temos a camada de visualização, onde os dados são apresentados e fornecidos pelo usuário. Quando dividimos a aplicação entre cliente e servidor, temos que a camada de modelo reside no próprio servidor, onde todos os dados dos clientes serão armazenados. A camada de controle pode estar tanto no servidor quanto no cliente, mas a camada de visualização, na maioria das vezes, reside no cliente.

Com o advento dos jogos multijogadores, tais como Doom, Quake, Counter Strike, Starcraft, entre outros, a utilização de um centralizador de ações, onde todos os jogadores pudessem trocar informações, se tornou essencial. No princípio, tínhamos uma estrutura arcaica de cliente/servidor, como a que existia nos jogos citados acima. Nesses jogos, o próprio jogo tinha capacidade de ser cliente e servidor. Assim, os jogadores elegiam um deles como sendo o servidor, normalmente o responsável por criar e gerenciar a partida. Após criar a partida e configurar suas informações, os demais jogadores podiam se conectar a esse jogador anfitrião, isto é, o servidor, para participarem da partida. O anfitrião também era um cliente, se conectando a ele mesmo. A dificuldade desse tipo de arquitetura era a limitação de partidas em rede local. Claro que, teoricamente, seria possível a mesma arquitetura ser expandida para a internet, porém deixar que um jogador seja servidor pode

acabar com a fluidez da partida, pois não só a latência e velocidade da conexão de rede dos jogadores seriam responsáveis por problemas de atraso, mas também a latência e velocidade da conexão do anfitrião, principalmente. Além disso, toda a conexão entre jogadores depende do anfitrião, sendo perdida caso este se desconecte.

Jogos mais modernos, como Quake 3 e o Unreal Tournament 3 utilizam uma autêntica arquitetura cliente/servidor, onde um servidor autoritativo é o responsável por executar a lógica do jogo. A esse servidor, são conectados clientes “burros”, onde não fazem nada a não ser enviar e receber dados desse servidor. O servidor é responsável por executar comandos de entrada, mover objetos e mandar essas informações de volta para o cliente. (Bernier, 2001).

Com o advento dos jogos de múltiplos jogadores massivos (MMOG), a arquitetura se tornou mais que necessária, um requisito prioritário ao desenvolver um jogo desse tipo. A princípio, jogos pioneiros desse tipo, como Tibia e Ragnarok Online, utilizavam a mesma arquitetura acima, onde o cliente de jogo era parcialmente “burro”, deixando boa parte da lógica de jogo a cargo do servidor. Para exemplificar esse tipo de abordagem, usarei como exemplo o jogo Ragnarok Online, da empresa Gravity. O jogo possui um cliente simples, onde o usuário conectava-se ao servidor com sua informação de login, então era redirecionado para a seleção de personagens. Após selecionado o personagem, o jogador podia entrar no jogo. O servidor do jogo roda a plataforma Aegis, também criada pela Gravity. Após alguns vazamentos sobre essa plataforma e análise do próprio cliente e dos pacotes de comunicação, foi criado o Athena, e sua evolução para o mercado americano, o eAthena. O eAthena é um emulador do servidor Aegis, feito pela comunidade e possui código aberto. Para analisar como o servidor Aegis se comporta, será analisado o eAthena. De forma semelhante ao Aegis, o eAthena possui três servidores, o Login Server, o Char Server e o Map Server. Cada servidor possui endereços distintos, podendo ser separados em diferentes hardwares. Eles se auto conectam entre os outros, e o cliente do jogo se conecta em cada um deles. Assim, o processo de entrada no jogo pelo jogador segue a ordem: Login Server, Char Server, Map Server. O primeiro servidor gerencia a autenticação e credenciais do usuário. O segundo gerencia estado dos

personagens, incluindo posição no mundo, nível, e outros dados. O último servidor gerencia mapa, personagens não-jogadores e monstros. É nesse servidor que parte da lógica do jogo é executada.

No Ragnarok Online, o cliente é simplista. Devido à direção de arte, o jogo é 2D, possuindo jogabilidade baseada em mouse. O jogador pode se movimentar por um mapa, clicando em uma determinada posição do mapa ou em um monstro para iniciar um ataque. Cada mapa se conecta a outro através de portais, assim, o jogador sempre está limitado à área do mapa que está atualmente. O cliente do jogo basicamente envia requisições ao servidor de mapa (Map Server) e este responde a todos os jogadores conectados e presentes no mapa em questão. Desse modo, a ação do jogador só é executada no cliente caso o servidor processe devidamente a requisição e o cliente receba a mensagem de que o personagem teve seu estado alterado. Para exemplificar, considere um personagem em um mapa no jogo. O jogador clica em uma posição no mapa, o jogo então requisita ao servidor a mudança de estado do personagem, nesse caso, sua posição no mapa. O servidor então processa e devolve uma mensagem a todos os personagens no mapa, avisando que esse personagem possui um novo estado. O próprio cliente do jogador também recebe a mensagem e só executa a ação de movimentação quando essa mensagem é recebida do servidor. Para o jogador, essa forma de comunicação acaba sendo prejudicial, pois o mesmo só percebe movimento após o servidor responder sobre a atualização do personagem.

Com a evolução dos jogos, uma nova abordagem foi utilizada, a qual segue a nova geração de jogos de múltiplos jogadores. Com a utilização da jogabilidade 3D, e o uso de movimentação por comandos de teclado, a forma utilizada até o momento se tornou impraticável. Assim, a atualização de informações do jogador passou a ser feita periodicamente pelo cliente, e este não esperava a resposta do servidor para efetuar a mudança. Desse modo, o jogo passa a ser mais fluido para o jogador e a alta latência da conexão não afeta drasticamente a jogabilidade.

Com o advento das plataformas móveis, o gênero de jogo multijogador passou por melhorias para se adaptar a essa nova plataforma. O problema maior dessas novas plataformas são as redes móveis, que ainda não possuem a mesma qualidade das redes fixas, tais como a ADSL ou a DOCSIS,

conhecida comumente como “Cable Internet”, que são as duas redes fixas predominantes no Brasil. Para atingir esse novo nicho, a quantidade de informações transferidas entre cliente e servidor deve ser revista. Atualmente temos redes com velocidades competitivas com as redes fixas, como por exemplo a tecnologia UMTS (4G) e a HSPA (3G), porém a maioria dos pacotes de dados oferecidos pelas operadoras possuem limite de tráfego de dados, o que limita e muito a experiência online de um jogador dessas plataformas. Para minimizar esse problema, o protocolo de comunicação entre cliente e servidor deve permitir a compressão dos dados, o que diminui o tamanho do pacote transmitido. Além disso, cabe ao desenvolvedor criar pacotes de dados da forma mais otimizada possível.

Em um servidor autoritativo, mesmo que o cliente possua parte da lógica, ainda sim este precisa lidar com diversos problemas, como por exemplo, gerenciar o comportamento de PNJs e monstros. Necessariamente, a inteligência artificial desses tipos de objetos precisa ser gerenciada pelo servidor. Implementar algoritmos de inteligência artificial no servidor não é uma tarefa árdua, porém, um sistema inteligente precisa ter ciência do ambiente que ele existirá, isto é, mesmo que o sistema funcione no servidor, este precisa de informações do ambiente atual de jogo, algo que apenas o cliente possui de forma completa. Alguns trabalhos nessa pesquisa foram feitos, como o modelo proposto por Björnsson e Halldórsson (2006), que utiliza uma heurística aprimorada para o já utilizado algoritmo A*, com base em algoritmos de busca de caminhos hierárquicos. Alguns projetos de jogos multijogadores utilizam servidores baseados no próprio motor de jogo para lidar com essas questões de forma otimizada.

2 Resultados Esperados

Espera-se com esse projeto, criar um jogo multijogador onde pode-se customizar o personagem entre três raças e classes diferentes. O objetivo é permitir a interação entre os jogadores em tempo real de jogo, onde cada jogador visualiza de forma simultânea, os outros jogadores conectados. Para simplificar o desenvolvimento, as seguintes características de um jogo deste formato serão implementadas:

- Replicação de estado de cada jogador para os demais conectados, i.e., novo jogador conectado, movimento de jogador, desconexão, entre outros estados.
- Instanciação de personagens não-jogadores (PNJ), que possuem movimentação automática segundo uma inteligência artificial básica, mas que são considerados, a nível de comunicação no lado cliente, como um jogador.
- Identificação de localização de jogador por área, permitindo que o jogo não carregue dados de todos os jogadores e PNJs conectados, mas somente daqueles que estão em sua área de jogo atual.

Espera-se que os resultados acima se comportem de forma semelhante tanto na plataforma Android quanto na plataforma PC.

3 Projeto

3.1 Proposta

A proposta do projeto é desenvolver o jogo Tales of Unreal, atendendo aos requisitos esperados, discutidos no tópico anterior. Para isso, foi criado um projeto de jogo multijogadores/multiplataforma, permitindo um grau de customização ao jogador e simulando características básicas de jogos desse tipo.

3.2 Tecnologias utilizadas

Para simplificar o processo de desenvolvimento e testes, foram definidas algumas questões técnicas, detalhadas a seguir.

3.2.1 Tecnologias de desenvolvimento

Para a arquitetura de servidor, foi escolhido o sistema Photon Server, da Exit Games. O sistema é codificado em C++ integrado ao .NET Framework da Microsoft. Assim ele age como um servidor de aplicação .NET disponibilizando um protocolo próprio para comunicação entre cliente e servidor, permitindo que toda a regra de negócio seja desenvolvida de acordo com a necessidade do projeto. Para o cliente de jogo, foi escolhido o motor Unreal Engine 4, da Epic Games. O motor de jogo é um dos maiores na área de jogos, conhecido por grandes jogos, como Gears of War, Unreal Tournament, entre outros. O motor possui suporte para jogos multijogadores, podendo ser usado como servidor, mas não possui uma boa forma de gerenciamento quando lidamos com jogos de mundo aberto (open world) e grande quantidade de jogadores (massive multiplayer). Devido a essa limitação, a união das duas ferramentas foi proposta, de modo a prever uma boa qualidade gráfica e de jogabilidade, tanto em computadores quanto dispositivos móveis e prover a compatibilidade e gerência de muitos jogadores simultâneos. Para armazenamento de dados dos jogadores, será utilizado banco de dados Oracle MySQL.

3.2.2 Plataformas de testes e desenvolvimento

Para o desenvolvimento e testes, foram selecionadas algumas plataformas alvo. Para PC, o foco será para plataforma Windows com hardware necessário para executar o cliente do jogo. Para Android, foram selecionados dois aparelhos para testes, de modo a comparar diferentes hardwares. Abaixo a especificação de cada aparelho:

- NVIDIA Shield
 - CPU: NVIDIA Tegra 4 quad-core
 - GPU: NVIDIA Tegra 4 (integrada ao processador)
 - Memória RAM: 2 GB
 - Sistema operacional: Android KitKat 4.4.2
 - Resolução: 1280x720
 - Adicionais: Tela em modo paisagem apenas, possui gamepad embutido
- Google Nexus 4
 - CPU: Qualcomm Snapdragon S4
 - GPU: Adreno 320 (integrada ao processador)
 - Memória RAM: 2 GB
 - Sistema operacional: Android KitKat 4.4.4
 - Resolução: 768 x 1280

3.2.3 Estrutura do servidor

3.2.3.1 Protocolo de comunicação

O Photon Server possui um protocolo próprio para comunicação entre o servidor e seus clientes. Internamente, temos a implementação da pilha TCP/UDP, que é gerenciada pelo framework do servidor, que abstrai as camadas mais baixas e cria uma padronização entre as mensagens. Assim, o conceito utilizado pelo framework é o de operação e evento (Exit Games, 2014).

Toda mensagem enviada do cliente para o servidor é tratada como uma operação. Essa operação é composta por um código e um mapa de parâmetros. O servidor pode responder a essa operação no mesmo padrão em

que a mensagem foi enviado, i.e., uma operação de resposta contendo um código e um mapa de parâmetros. O servidor pode a qualquer momento disparar um evento para qualquer um dos clientes conectados. A mensagem segue o mesmo padrão, sendo composta de um código e de um mapa de parâmetros (Exit Games 2014).

Tal padronização nos permite enviar todos os dados que precisamos entre cliente e servidor. O framework já trata da compactação necessária desses pacotes, de modo a minimizar a carga de dados entre os dois lados. De modo a minimizar o tamanho do mapa de parâmetros, nem todos os parâmetros definidos para uma determinada entidade no jogo são mapeados, pois como existe a serialização, não é possível dizer se a serialização/compactação dos dados é otimizada. Segundo a Exit Games (2014), em sua documentação da SDK do servidor, o protocolo é otimizado para dados em formato texto, assim, foi escolhido o padrão JSON para trafegar grandes objetos de dados, tais como os dados do jogador, por exemplo. Essa abordagem visa também a otimização para a plataforma Android e para redes móveis.

3.2.3.2 Comunicação com o banco de dados

Como um sistema acima do .NET Framework, uma aplicação rodando sobre o Photon Server tem compatibilidade com todo framework .NET e complementos. Assim, a comunicação com banco de dados segue o mesmo formato já utilizado na plataforma. Para facilitar o projeto, e manter a arquitetura MVC, utilizamos a biblioteca nHibernate, um subprojeto do JBoss Hibernate, compatível com C# e .NET. O banco de dados utilizado nos testes foi o Oracle MySQL Community Edition. Devido a arquitetura simples do modelo de dados, a plataforma é suficiente para lidar com os dados. Mesmo devido a simplicidade, a aplicação sobre o Photon Server faz todo o gerenciamento de dados antes que este seja gravado no banco de dados, minimizando assim o impacto de gravações. Devido a simplicidade do projeto, não foram feitas nenhuma otimização e configuração preventiva na base de dados e no código, de modo a evitar problemas como bloqueio de dados na base de dados (data lock ou deadlock). O servidor, para esse projeto, gerencia de modo eficiente a conexão com os dados, através do lock otimista do

nHibernate, evitando esse tipo de problema.

3.2.3.3 Gerenciamento de usuários e personagens

Cada jogador possui dados de credencial para acesso ao jogo. Deste modo, é garantida a segurança entre jogadores, os quais possuem acesso privado aos seus personagens e recursos. Deste modo, cada usuário pode ter diversos personagens, porém o acesso simultâneo entre personagens do mesmo jogador não é permitido por questões de segurança e gerência. Os motivos de segurança são implícitos, já que impede que um mesmo jogador use sua conta em dois computadores diferentes. O segundo motivo existe para facilitar a localização de dados referentes ao jogador a partir do personagem, evitando que exista dados fora de sincronia devido ao acesso concorrente do mesmo registro de dados. O jogo segue o seguinte fluxo entre o usuário iniciar o jogo e entrar no mundo com seu personagem: primeiramente o jogador necessita efetuar o login na interface do jogo. O jogo então envia uma operação ao servidor informando o login do usuário. Nesse momento, ao receber a mensagem que o login foi efetuado com sucesso, o jogo envia operação solicitando os personagens do usuário conectado. O servidor retorna com uma lista de objetos de personagens ao jogo. O jogador então seleciona o personagem e entra no jogo. Nesse momento uma operação requisitando a entrada no mundo é feita indicando qual personagem que irá se conectar. Nesse momento, o servidor salva carrega os dados do personagem que entrou no mundo, o coloca em uma lista de personagens em jogo e informa que o personagem entrou no jogo para o jogador. O jogo então solicita a informação de todos os jogadores conectados, segundo a gerência de áreas que será discutida a seguir. O jogo carrega o mapa onde o jogador se encontra, o coloca na última posição em que se encontrava da última vez que se conectou e o instancia no jogo. No mesmo tempo, o jogo lê os dados dos jogadores que o servidor enviou e também os instanciam no jogo. Durante isso, o servidor envia evento para todos os clientes, exceto para o cliente atual, que um novo personagem entrou no jogo. Cada cliente possui uma lista de jogadores visíveis que é atualizada segundo os eventos disparados pelo servidor. A cada 300ms, caso algum personagem tenha se movido, os novos dados são enviados ao servidor que o replica a todos os clientes. Caso um personagem se

desconecte, o servidor o remove de sua lista e informa os outros clientes que este desconectou.

3.2.3.4 Gerenciamento de áreas em um mundo aberto

Ao se pensar em um jogo multijogadores, podemos pensar em duas alternativas para o gerenciamento de mapas e áreas. A primeira é criar mapas de tamanho fixo, indexados no servidor. Dessa forma, o servidor precisa apenas saber a posição do jogador e o mapa que este se encontra. Essa abordagem funciona bem para jogos onde o jogador transita entre uma área e outra através de sistemas de carregamento de mapas, ou comumente conhecido em se tratando de jogos, portais. Dessa forma, ao entrar em um portal, um novo mapa é carregado e o personagem é transferido para esse mapa tanto no jogo quanto no servidor. O problema é que, para um jogo de mundo aberto, onde o jogador tem a impressão de sempre estar no mesmo mapa, a abordagem acima não funciona. Segundo Boulanger, et al. (2006), o sistema de área de interesse pode ser utilizado para resolver essa questão. Nesse sistema, uma área virtual é criada ao redor do personagem. Todo novo personagem que entra nessa área se inscreve para receber atualizações desse personagem. Seguindo essa regra, todo personagem que se inscreve na área de interesse de outro, recebe atualizações desse personagem, assim o enxergando. Todos os personagens fora da área de interesse, isto é, não inscritos nela, não se tornam visíveis pelo jogador, diminuindo a quantidade de mensagens recebidas por ele.

Outra forma de lidar com a situação de atualizações de personagens é utilizar uma modificação da primeira forma. Em motores modernos, como é o caso da Unreal Engine 4, um mapa pode ser carregado sob demanda, ou seja, pode ser carregado enquanto o jogador se encontra em um mapa anterior desde que este tenha ultrapassado uma determinada área próxima ao final do mapa atual. Quando o novo mapa é carregado, ele é adicionado ao mapa atual como se fosse um único mapa. Após a transição, o mapa anterior é descarregado. Usando esse recurso, podemos lidar com a transição da mesma forma que é tratada a transição de áreas através de portais. Devido à complexidade tanto no cliente quanto no servidor, foi optado utilizar essa última forma, de modo a facilitar o desenvolvimento e minimizar a carga do servidor.

3.2.4 Estrutura do jogo

3.2.4.1 Tratamento de mensagens do servidor

Quando o jogo recebe uma mensagem do servidor, seja um evento ou uma resposta de operação, a mensagem é tratada da mesma forma com que chega no servidor. O jogo recebe um pacote com um código e um mapa de parâmetros. De acordo com o código definido, o jogo trata os parâmetros e realiza a conversão de JSON para objeto, caso necessário. Quando a mensagem se trata de uma atualização de personagem, o jogo encontra o personagem em sua lista local de personagens conectados e o atualiza. Mais detalhes de como esse atualização é tratada serão vistos a diante.

3.2.4.2 Personagens e customização

Para simular uma plataforma multijogadores, decidiu-se pela customização de personagens. A customização foi selecionada para ser simples, com um sistema simples de raça e classe. Por simplicidade, foram criadas as raças de raposa, rato e dinossauro, e como classes, mago, guerreiro e clérigo.

Cada personagem é gerado com a combinação de uma raça ou classe e seguido de um nome. Os dados são armazenados no servidor e enviados ao jogo assim que o jogador efetua login com sucesso. No jogo, pensando na otimização para a plataforma Android, cada raça possui um modelo fixo e a classe é representada por uma roupa, isto é, um item vinculado ao modelo da raça.

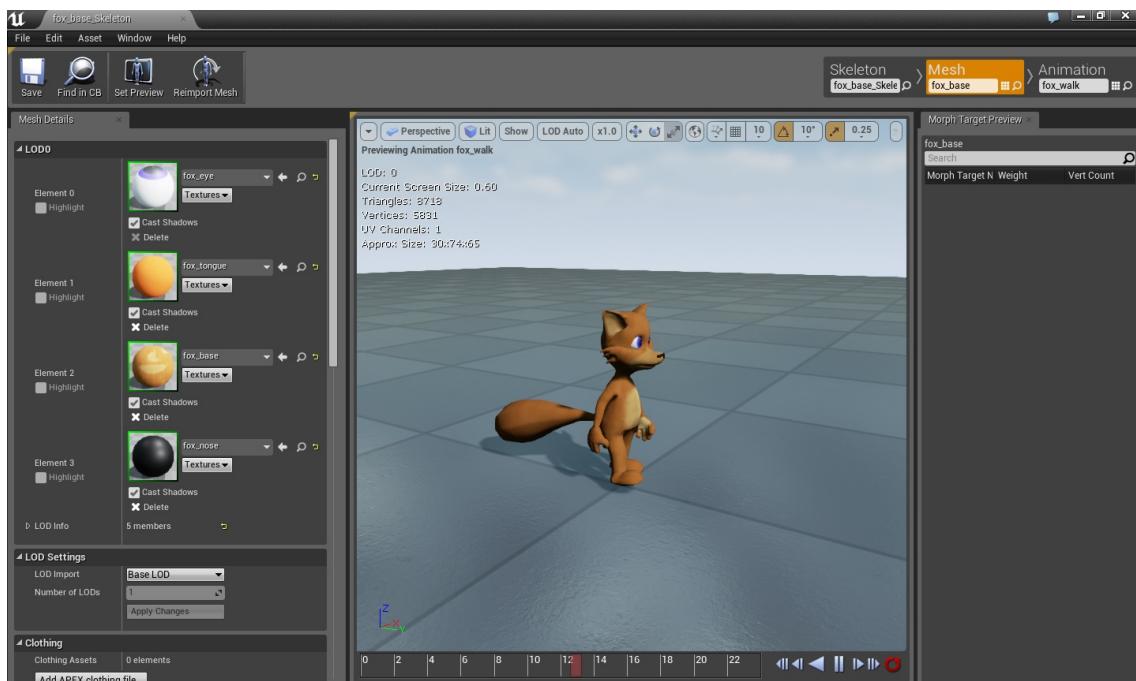


Figura 1 - Modelo da Raposa e animação

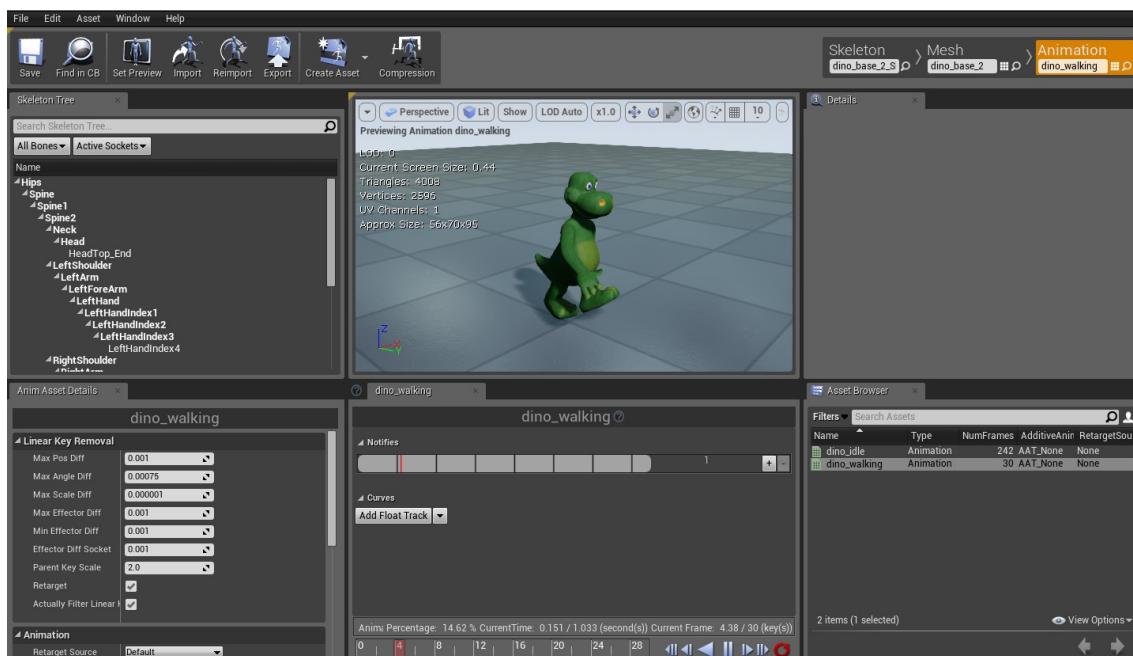


Figura 2 - Modelo do Dinossauro e animação

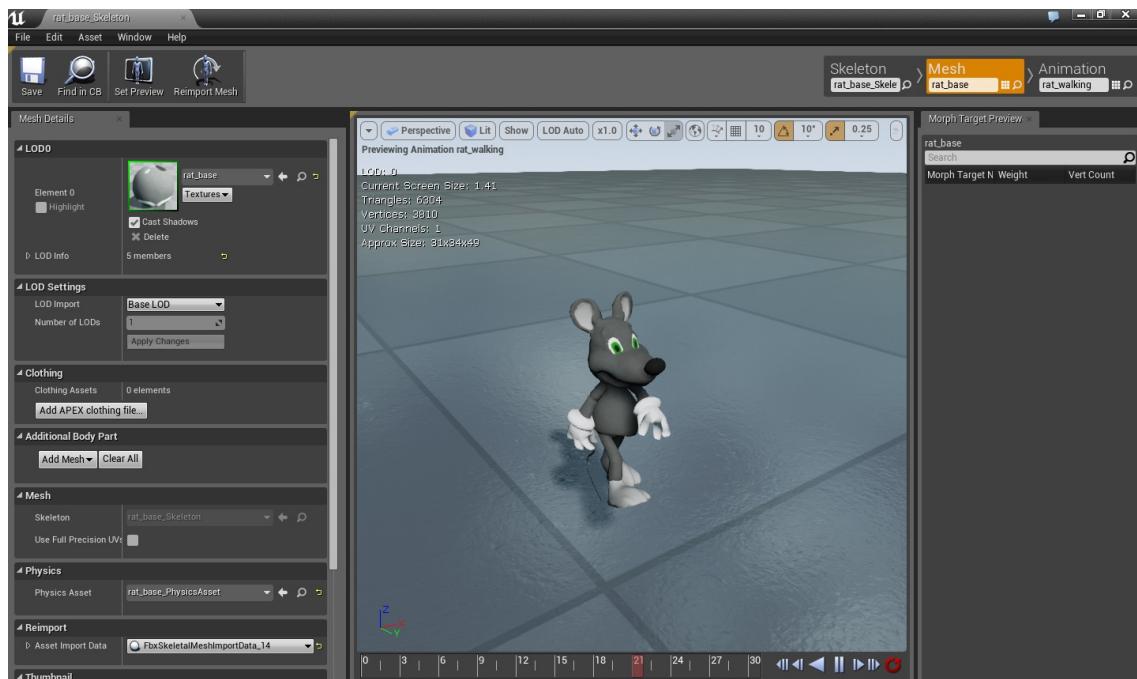


Figura 3 - Modelo do Rato e animação

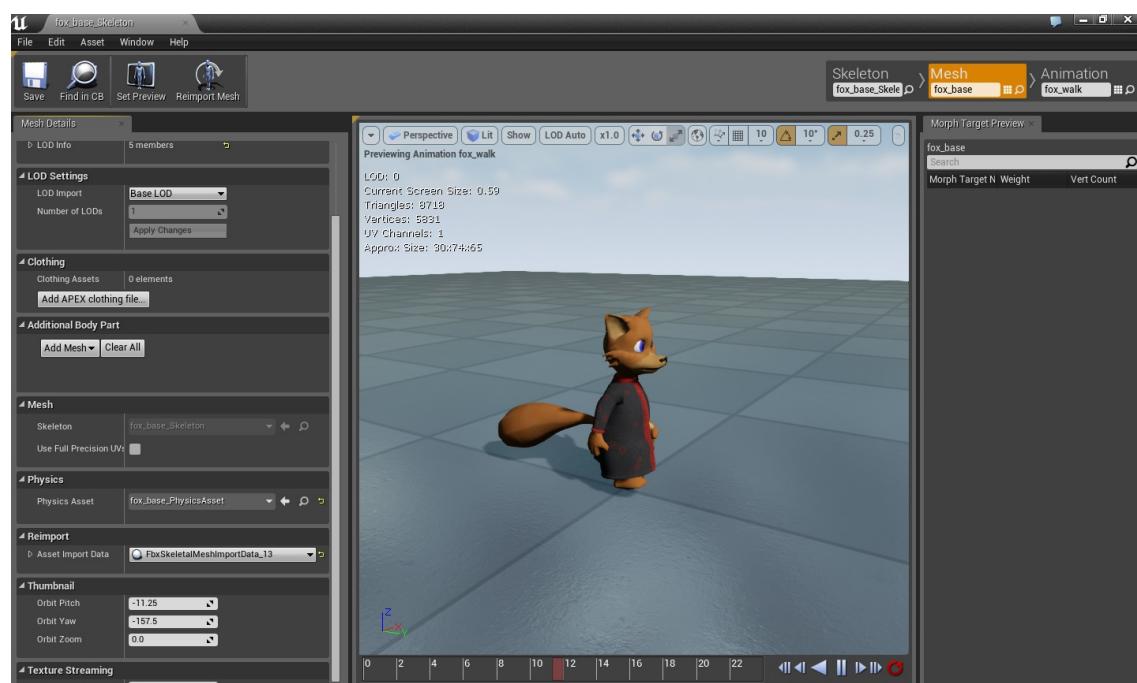


Figura 4 - Roupa de Mago (Raposa)

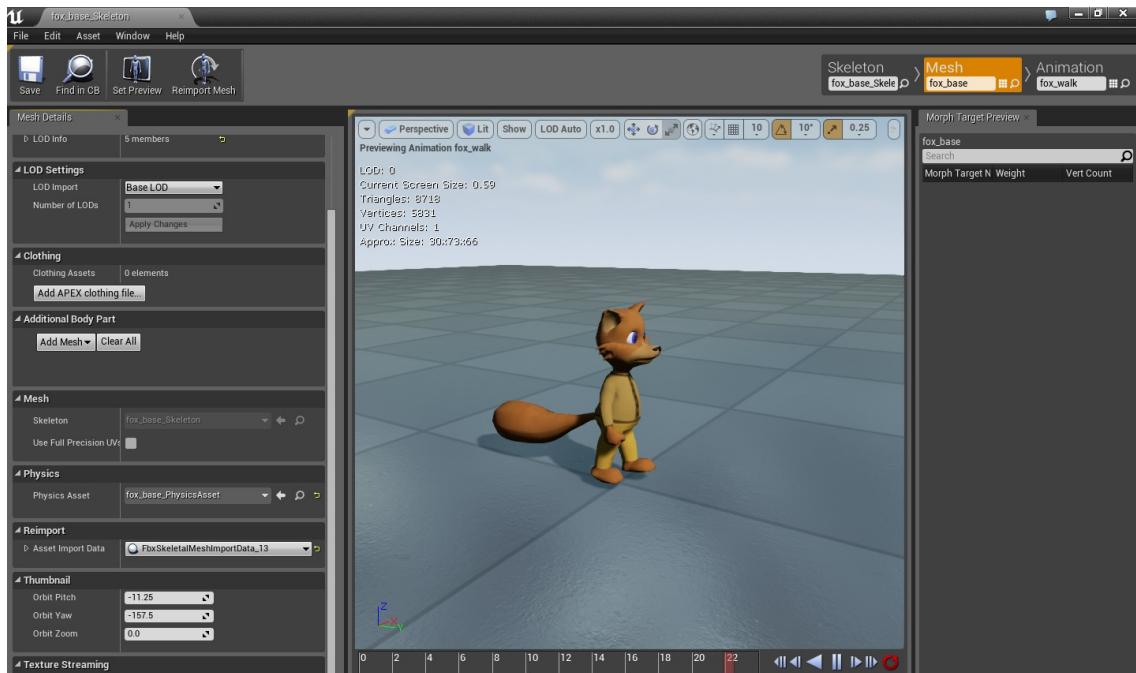


Figura 5 - Roupa de Guerreiro (Raposa)

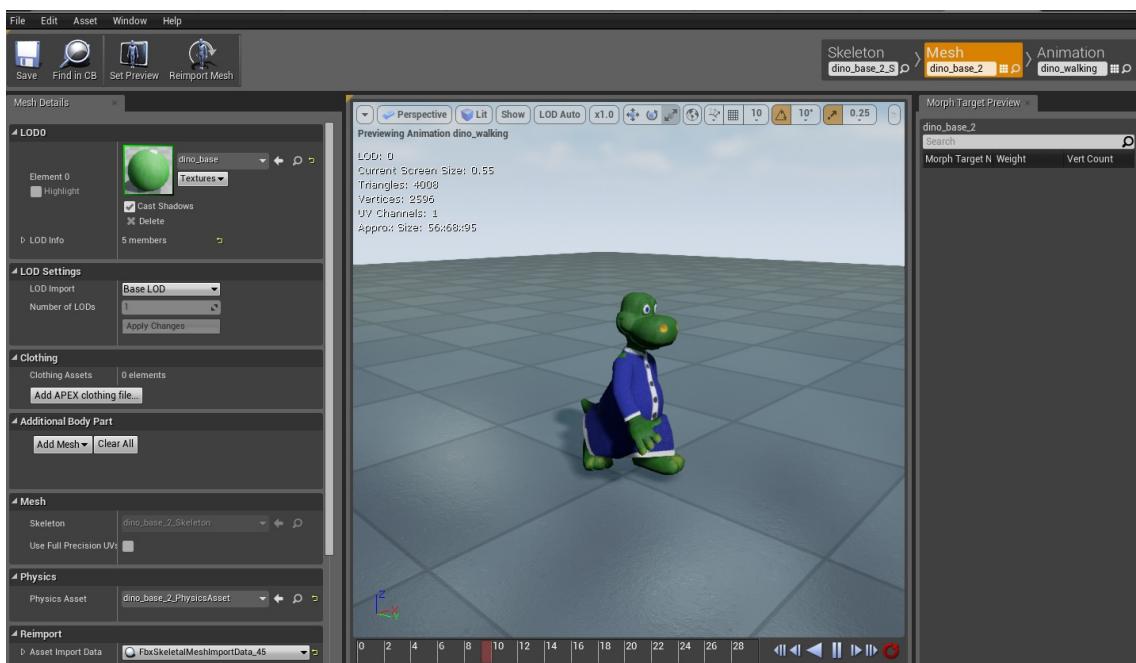


Figura 6 - Roupa de Mago (Dinossauro)



Figura 7 - Roupa de Guerreiro (Dinossauro)

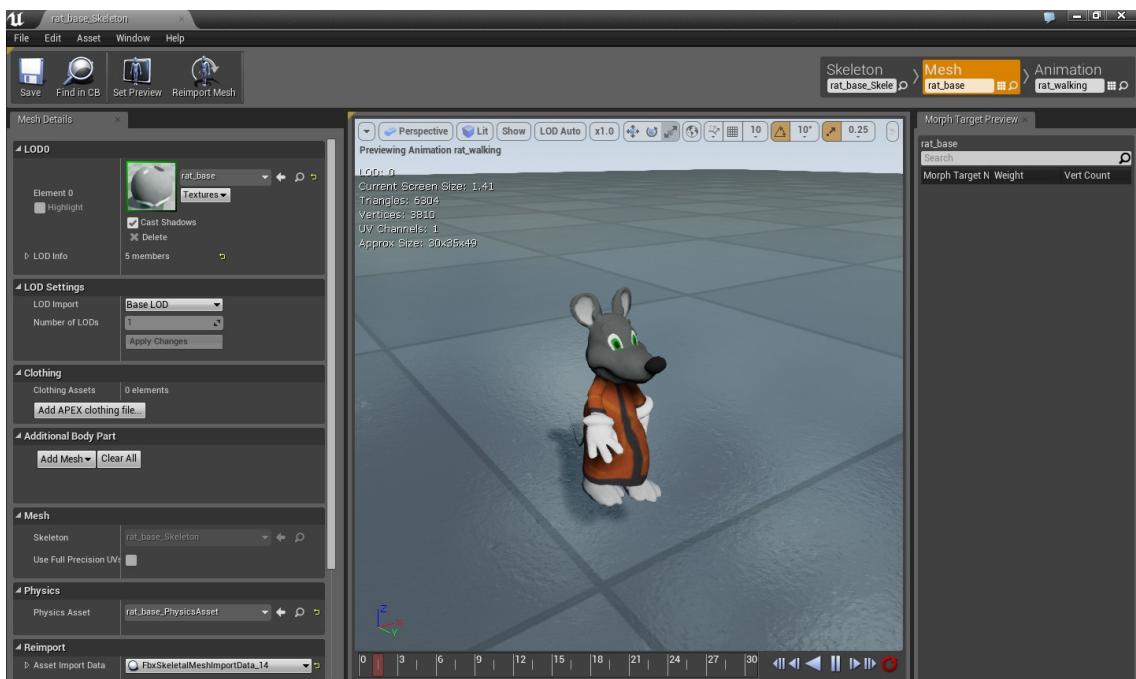


Figura 8 - Roupa de Mago (Rato)

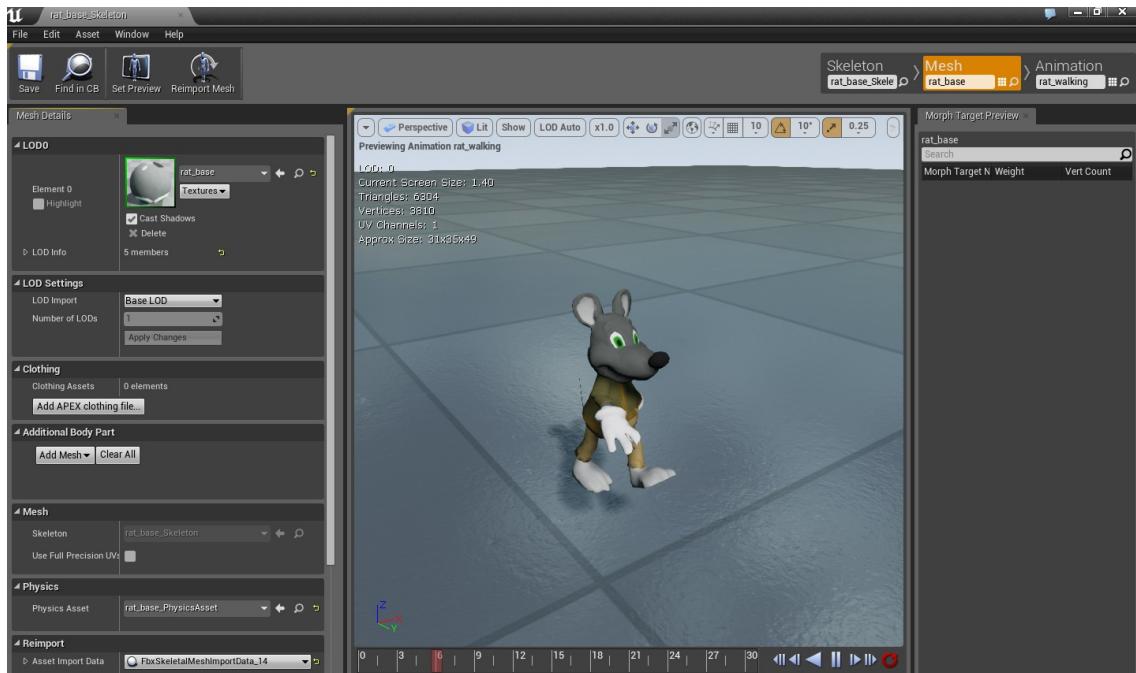


Figura 9 - Roupa de Guerreiro (Rato)

Os modelos tiveram algumas limitações devido à plataforma Android. A diante será especificado as peculiaridades da plataforma quanto ao servidor e ao cliente de jogo.

3.2.4.3 Replicação de informações dos jogadores

Abaixo a representação do mapa onde os jogadores podem se movimentar.

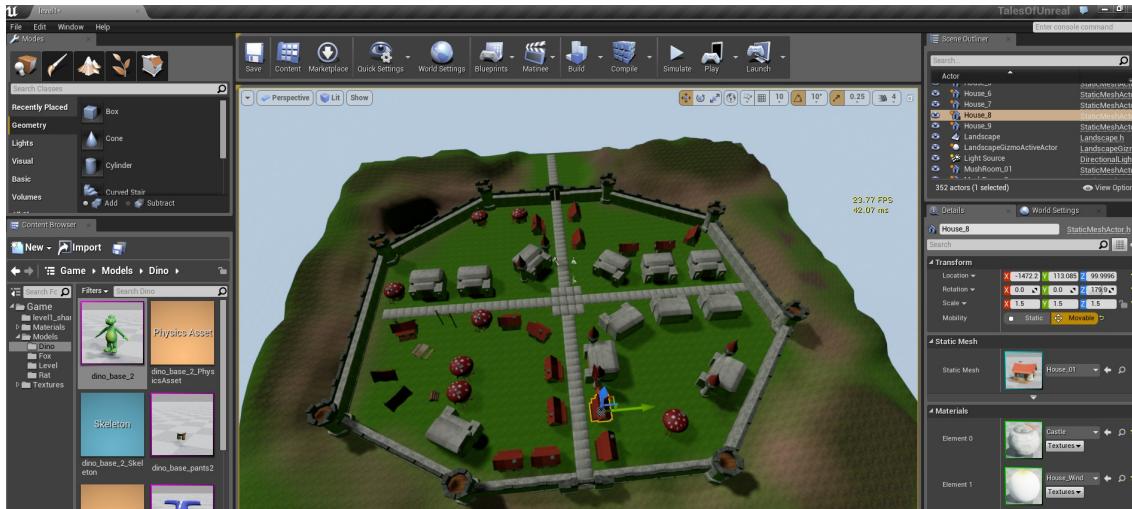


Figura 10 - Mapa



Figura 11 - Mapa aproximado

Para que fosse possível a navegação pelo mapa, alguns pontos foram considerados. O primeiro era em relação ao controle do personagem. Felizmente, para o motor de jogo, as entradas de teclado e mouse são replicadas em controles na tela do dispositivo. Além disso, no NVIDIA Shield,

onde os controles são embutidos no dispositivo, a mesma entrada de usuário é mapeada para esses controles.

Como discutido anteriormente, cada entrada do usuário faz com que o personagem se movimente no jogo. A cada janela de atualização, definida em 300ms, o estado do jogador é verificado. Se o personagem se movimentou ou alterou seu estado desde a janela anterior, o jogo envia uma operação de atualização para o servidor. Para os outros personagens, as mensagens são tratadas de forma igual, alterando os valores do objeto personagem salvo em lista local no cliente. Porém, o jogador precisa ter a impressão de que o personagem se movimentou de forma suave, isto é, utilizando sua animação de andar. Para resolver esse problema, cada personagem conectado, exceto o personagem do jogador que é controlado pela entrada padrão do jogador, é tratado como um objeto de inteligência artificial. Assim, duas coisas precisaram ser feitas. A primeira era definir a malha de navegação no mapa (Navigation Mesh). Essa malha permite que, de qualquer ponto, um objeto autônomo possa se movimentar para outro ponto, escolhendo o melhor caminho entre os obstáculos. Como discutido anteriormente, utilizam-se algorítimos de busca de melhor caminho como o A*. Na Unreal Engine, a malha de navegação é calculada previamente, isto reduz a necessidade de um algoritmo de busca em tempo real, em tempo de jogo. No motor, todos os possíveis caminhos, isto é, grafos de caminhos são salvos e carregados automaticamente assim que necessário. Para a plataforma Android, esse ganho é maior pois algoritmos de busca de caminho são custosos em recursos de memória e CPU.

A segunda configuração feita foi tornar personagens em agentes autônomos. Cada personagem, isto é, a combinação entre raça e classe, é representado como um objeto único dentro do jogo. Esse objeto pode ser possuído por um controle, que assume a responsabilidade de mostrar o objeto para a câmera principal do jogo e para lhe dar comandos de movimentação. Ao entrar no jogo, o jogador escolhe seu personagem. Este é então instanciado no jogo e possuído por um controle de jogador, que o coloca em foco na câmera principal e permite que este seja movimentado pelos comandos do jogador. Todo novo personagem conectado, ou já existente no mundo, são instanciados, segundo sua raça e classe e são possuídos por um controlador diferente, que permite que comandos sejam dados sem que uma entrada física

seja feita. Esse tipo de controle são os controles de inteligência artificial existentes no motor de jogo. No nosso caso, precisamos apenas da funcionalidade de movimento autônomo desse controle, baseando-se na malha de navegação. Assim, quando um personagem remoto é atualizado, a nova posição é então enviada ao controle que o movimenta seguindo o caminho especificado pela malha de navegação. Com a janela de atualização de 300ms, o movimento se mostrou suave o suficiente para passar a impressão de tempo real necessária nesse tipo de jogo.



Figura 12 - Replicação de movimento

O último problema que precisa ser lidado é quanto à conexão e desconexão do jogador. Tanto servidor quanto cliente possuem uma lista de jogadores conectados. Ao conectar um novo jogador, o servidor envia mensagem de novo personagem no mundo para todos os jogadores conectados. Esses clientes então incluem o novo personagem na lista, o instancia no jogo e então o possui com um controle de IA. Ao se desconectar, o servidor remove o jogador de sua lista e avisa os clientes. Os clientes por sua vez, ao receber o evento de desconexão, destrói o personagem desconectado e o remove da lista.

3.2.4.4 Sistema de Inteligência Artificial replicado do servidor

O sistema proposto para lidar com inteligência artificial não foi desenvolvido em sua totalidade. A proposta, a fim de simplificar o desenvolvimento, foi replicar comportamentos simples. No sistema proposto, o comportamento de PNJs e monstros é feito diretamente no servidor. Porém, como visto, não temos a nossa disposição, por parte do servidor, meios de lidar com a física de cada um desses componentes. Esse problema pode ser resolvido, mas será discutido em trabalhos futuros.

No momento, a IA é considerada como um personagem remoto. Dado um PNJ, este tem seus dados salvos no servidor. Dado um comportamento simples, como mover aleatoriamente para alguma posição dentro de um determinado raio de ação, o servidor decide a direção em que o PNJ irá tomar em seguida. Essa posição então é enviada aos jogadores que atualizam o movimento do PNJ usando a malha de navegação. Caso a posição selecionada para destino tenha um obstáculo, a própria malha impedirá o movimento, além do que o sistema de física, isto é, o sistema de colisões, será aplicado a esse PNJ. Como o mapa é igual para todos, o movimento será, com uma certa margem de erro, igual em todos os clientes.

3.2.4.5 Detalhes específicos da plataforma Android

Para que o jogo fosse portado para a plataforma Android, alguns detalhes específicos da plataforma precisaram ser seguidos. Como já falado, o sistema de entrada de usuário é automaticamente exposto ao usuário pela UE4, conforme a figura 13.

Além disso, o próprio motor de jogo possui restrições para a plataforma. A primeira delas é a quantidade de junções (joints) para malhas com esqueleto (Skeletal Mesh). Segundo a documentação (Epic 2014), apenas malhas com 45 ossos (bones) no máximo podem ser renderizadas em plataformas móveis. Caso esse limite seja ultrapassado, o jogo continua funcionando no PC, mas a malha e todo o objeto não é renderizado nem processado pelo motor.



Figura 13 - Controles de entrada Android

Outra limitação, já comentada, é devido ao protocolo de comunicação e suporte para redes móveis. Para o servidor, isso foi transparente, pois tudo trafega sob a pilha TCP/IP pela internet, porém, para o cliente, alguns módulos precisaram ser incluídos na compilação. Graças à uma característica da UE4, todo projeto criado em C++ com destino de compilação Android, utiliza a Android NDK para compilar todo o código em código nativo Android. Esse fator nos permitiu utilizar a API do Photon Server para Android NDK dentro do código do projeto. A mesma coisa foi feita para incluir a API para Windows, também em C++. A UE4 utiliza parte do MonoFramework, uma implementação multiplataforma, de código aberto do .NET Framework, para gerenciar a compilação multiplataforma. Assim, é possível especificar os módulos do motor que precisam ser carregados, e o próprio processo de compilação decide qual módulo carregar de acordo com a plataforma destino. O mesmo sistema permite que bibliotecas estáticas possam ser utilizadas dentro do projeto, e, com algumas verificações de plataforma, foi possível, no mesmo projeto, ter suporte ao Photon Server tanto para PC quanto para Android.

Considerando o hardware existente nas plataformas Android, nem

todos os recursos gráficos são compatíveis com a plataforma. Um exemplo são as sombras dinâmicas, que não são geradas na plataforma Android, na versão do motor que foi utilizada. Em versões futuras, alguns recursos, como sombras dinâmicas serão suportados por plataformas móveis.

Em geral, o motor UE4 tem compatibilidade excelente com a plataforma, mas ainda existem as limitações de hardware que não são simples de serem ultrapassadas, principalmente na questão gráfica. A qualidade gráfica do jogo em PC e Android se mostrou praticamente a mesma devido às otimizações gráficas ou perdas de recursos para a plataforma PC. Essa adaptação é necessária pois caso um recurso não compatível seja utilizado, não será mais possível manter o mesmo cliente, ou seja, o mesmo projeto, para as duas plataformas.

Na questão de processamento lógico, isto é, processo em CPU, não foi necessária grandes modificações no projeto. A própria mecânica escolhida para os sistemas de gerenciamento proporcionou performance boa tanto para Android quanto para PC. Mais sobre a análise de performance será visto no próximo tópico.

4 Resultados Obtidos

A seguir serão detalhados alguns resultados obtidos com os testes da plataforma.

4.1 Resultado geral de desempenho

Para a análise de desempenho, foram consideradas dois parâmetros: a quantidade de quadros por segundo (FPS) e a fluidez dos personagens no jogo. A análise geral foi boa, conseguindo uma média de 60 fps na máquina de desenvolvimento (Windows). A replicação de rede teve sucesso, não demonstrando qualquer perda de fluidez nos personagens remotos. O tempo levado para todo fluxo (login->conexão->entrar no mundo) durou aproximadamente 10s, tendo um tempo médio de resposta entre cada operação de 132ms, em rede local.

A conclusão da análise geral é que a plataforma desenvolvida obteve sucesso no que foi proposto. Em testes PC-PC, todo o fluxo de conexão funcionou de forma fluida, mantendo a média de FPS alcançada em testes sem conexão com o servidor.

4.2 Resultado de desempenho na plataforma Android

Analisando apenas o ambiente Android, foram efetuados três testes. O primeiro foi a replicação PC – Android via rede local. Nesse teste, foi utilizado o NVIDIA Shield. Devido a otimização GPU/CPU existente no chipset Tegra 4 e, pelo fato de já existir uma otimização por parte da NVIDIA na UE4 para suas plataformas, foram obtidos excelentes resultados. Alguns recursos gráficos como HDR que são suportados apenas para GPUs avançadas, rodou com sucesso no dispositivo. A média alcançada de quadros foi de 46fps, o que está perfeitamente dentro da faixa considerada jogável, isto é, sem quedas de performance ou sensação de travamento. A média de quadros é até alta se compararmos com outros dispositivos. A fluidez dos personagens remotos não ficou diferente dos testes entre PC.

O segundo teste foi executado entre PC e Android, mas dessa vez utilizando a rede móvel do dispositivo. Para esse teste foi utilizado o Nexus 4 como teste, conectado à rede 3G (HSPA). Nos testes, devido ao hardware do

aparelho, este obteve a média de 32fps, o que está na faixa jogável. A latência de rede prejudicou um pouco no fluxo de conexão e atualização de personagens remotos, mesmo assim, foi percebida uma leve lentidão na atualização de personagens remotos, o que não afetou a fluidez da movimentação destes.

O terceiro teste foi feito exclusivamente entre dispositivos Android. Nesse caso foi utilizado tanto rede local quanto rede móvel. Tanto a média de FPS dos dois dispositivos se manteve, e também a fluidez e sensação de movimento dos personagens remotos se manteve em relação aos testes anteriores.

5 Conclusões e trabalhos futuros

Conclui-se com esse trabalho que é perfeitamente possível criar uma plataforma multijogador com capacidade para múltiplas conexões simultâneas tanto para plataforma Android quanto para a plataforma PC. Considera-se que a estrutura física para adequar um jogo deste porte ainda é essencial, mas que a complexidade de desenvolver essa plataforma não se torna tão grande se os recursos forem aplicados de forma correta. A utilização de grandes ferramentas, como o motor Unreal Engine 4 e o servidor de aplicação Photon Server, permitiu uma grande customização do ambiente da plataforma, o que permitiu-se a otimização e o resultado esperado, uma jogabilidade semelhante em ambas as plataformas.

Um dos pontos de atenção detectados no trabalho está no gerenciamento de IA e sistemas de física. Futuramente, pensa-se em utilizar o próprio motor em modo servidor, sendo um auxiliar para o Photon Server. A ideia seria utilizar a UE4 como motor remoto de física e inteligência artificial, processando e devolvendo os dados ao servidor Photon. Essa metodologia pode permitir comportamentos mais avançados e do mesmo modo diminuir a carga de processamento do servidor Photon.

A utilização de melhorias gráficas, seguindo a evolução do motor também será implementada futuramente. Espera-se que o motor proporcione praticamente a mesma qualidade gráfica que se vê em plataformas PC nas plataformas móveis.

Algumas otimizações ainda precisam ser feitas. Durante alguns testes, houve travamentos na plataforma Android. O que se suspeita é que o problema foi causado devido a algum recurso mal utilizado do motor de jogo. Mesmo assim, o enfileiramento de mensagens do servidor não pode ser descartado.

Mesmo com os excelentes resultados deste trabalho, muito ainda precisa ser feito. Diversos sistemas precisam ser implementados e novos modelos de gerenciamento precisam ser desenvolvidos.

Concluindo, é possível criar a mesma experiência de jogo entre plataformas móveis e plataformas fixas, como o PC. Para isso, basta que os recursos tecnológicos sejam aplicados de forma criativa.

REFERÊNCIAS BIBLIOGRÁFICAS

REENSKAUG, Trygve. *MVC XEROX PARC 1978-79*. 1979. Disponível em: <<http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>>. Acesso em: 10 nov. 2014.

BERNIER, Yahn. *Latency compensating methods in client/server in-game protocol design and optimization*. Proceeding of 15th Games Developers Conference, San Jose, CA, EUA, 2001.

BJÖRNSSON, Yngvi; HALLDÓRSSON, Kári. Improved heuristics for optimal pathfinding on game maps. Proceedings of American Association for Artificial Intelligence. EUA, 2006.

EPIC GAMES. Official Unreal Engine 4 Documentation. 2014. Disponível em: <<https://docs.unrealengine.com/>>. Acesso em: 11 nov. 2014.

EXIT GAMES. Photon Server Intro. 2014. Disponível em: <<http://doc.exitgames.com/en/onpremise/current/getting-started/photon-server-intro>>. Acesso em: 11 nov. 2014

_____. Exit Games Photon C++ Client Library. 2014. Disponível em: <<http://doc-api.exitgames.com/en/onpremise/current/cpp/doc>>. Acesso em: 11 nov. 2014

_____. Photon Server SDK. 2014. Disponível em: <<http://doc-api.exitgames.com/en/onpremise/current/server/doc>>. Acesso em: 11 nov. 2014

BOULANGER, Jean-S'bastien; KIENZLE, Jörg; VERBRUGGE, Clark. Comparing interest management algorithms for massively multiplayer games. Proceedings on 5th ACM SIGCOMM workshop on Network and system support for games. Nova Iorque, EUA, 2006.