UNIVERSIDADE SÃO FRANCISCO
Curso de Engenharia da Computação

FERNANDO ÉTORE GALLÃO

DESENVOLVIMENTO DE JOGO DISTRIBUÍDO EM REDE UTILIZANDO UNREAL DEVELOPMENT KIT

Itatiba
2011

FERNANDO ÉTORE GALLÃO – R.A. 002200600254

DESENVOLVIMENTO DE JOGO DISTRIBUÍDO EM REDE UTILIZANDO UNREAL DEVELOPMENT KIT

Monografia apresentada ao Curso de Engenharia da Computação da Universidade São Francisco, como requisito parcial para obtenção do título de Engenheiro da Computação.
Orientador: Prof. Fabio Andrijauskas.

Itatiba
2011

AGRADECIMENTOS

Agradeço aos meus pais, meus amigos, colegas de trabalho, professores, e todos que me apoiaram e me permitiram que fosse capaz de trabalhar neste projeto de desenvolvimento de um jogo. Agradeço ao Thomas Cintra Penteado por se comprometer com o desenvolvimento das interfaces para serem utilizadas neste jogo.

Hunt or be Hunted

RESUMO

Grid Computing é um modelo de computação distribuída descentralizada que tem como foco o uso de recursos em uma rede de computadores de forma a melhorar o desempenho ou disponibilidade de recursos compartilhados. Tendo isto em mente o objetivo desse trabalho é desenvolver um jogo de computador capaz de ter os seus recursos distribuídos, melhorando o desempenho e a disponibilidade do jogo, para isso será utilizado a ferramenta de desenvolvimento de jogos *Unreal Development Kit* (*UDK*). Esta ferramenta é gratuita para uso sem fins lucrativos, permite a portabilidade de jogos feitos tanto para a plataforma *Windows* quanto para *iOS*. Unindo a ideia de computação distribuída e a possibilidades de criar jogos em rede que a ferramenta *UDK* possibilita, será criado um jogo que terá em seu código as funcionalidades de cliente e servidor ao mesmo tempo. Agindo como um cliente para se conectar aos outros jogadores e como servidor para hospedar o "mundo" virtual onde o jogo se passa, este mundo possui uma ação de "*Teleporter*" que será responsável por fazer a comunicação com o servidor existente em outro computador da rede. Tal ação fará com que o personagem possa viajar livremente entre um mundo a outro, onde cada mundo estará hospedado no servidor existente no computador de cada jogador pelo tempo em que este estiver conectado ao jogo.

Palavras-chave: *Grid Computing*. desempenho. *Unreal Development Kit*.

ABSTRACT

Grid Computing is a model of decentralised distributed computing that focuses on the use of resources in a network of computers to improve performance or availability of shared resources. With this in mind the goal of this work is to develop a computer game able to have their distributed resources, improving performance and availability of the game, for this will be used to game development tool Unreal Development Kit (UDK). This tool is free for non-profit use, allows portability of both games made for the Windows platform and iOS. Uniting the idea of distributed computing and the possibilities for creating networked games that the tool enables a UDK game that will have in your code the client and server functionalities at the same time. Acting as a client to connect to other players and as a server to host the "virtual" world where the game takes place, this world has an action of "Teleporter" which will be responsible for making communication with the existing server on another computer on the network. Such action will cause the character can travel freely from one world to another, where each world will be hosted on the existing server on the computer of each player by the time that this is connected to the game.

Keywords: Grid Computing. performance. Unreal Development Kit.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGO FONTE

LISTA DE ABREVIATURAS E SIGLAS

UDK – *Unreal Development Kit*

RAM – *Random Acess Memory*

MSDNAA – *Microsoft Development Network Academic Alliance*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, todo computador pessoal independente de *desktop* ou *notebook* possui um ou mais recursos que acabam por ser subutilizados, sejam eles o processador, a memoria *RAM*, ou disco rígido. Estes recursos, criatividade e a ferramenta de desenvolvimento de jogos da *Epic Games*, o *Unreal Development Kit* possibilitam o desenvolvimento de um jogo de computador que pode ser distribuído em computadores em rede.

No decorrer deste trabalho foi possível desenvolver um jogo que possui características de servidor e cliente, e possibilita o aproveitamento de recursos computacionais ociosos através da rede distribuindo processamento e conteúdo entre os computadores. Para que isso fosse possível, além do uso da ferramenta *UDK*, que possibilita um ambiente gráfico para criar aplicações em ambiente tridimensional, também foram usados o *Visual Studio 2010* da Microsoft em conjunto com o *nFringe* da *Pixelmine* para o desenvolvimento do código usado no jogo, e algumas outras ferramentas que fazem parte do *UDK* que facilitaram o desenvolvimento e testes.

Levando em consideração estes recursos computacionais ociosos este trabalho busca criar uma solução de entretenimento que aproveite estes recursos de forma a melhorar o desempenho em geral da aplicação, para isto esta sendo desenvolvido um jogo de computador que será formado por um servidor e um cliente, o servidor utilizara os recursos de espaço em disco para armazenar o mapa que poderá ser criado pelo usuário utilizando a ferramenta *Unreal Development Kit* e a memoria *RAM* e processamento para criar um serviço no qual irá hospedar o mapa na rede permitindo que outros jogadores se conectem utilizando o cliente do jogo onde poderão se organizar em grupos para realizar missões de forma cooperativa. Além dos recursos computacionais o jogo tirara proveito da criatividade de cada usuário permitindo que sejam adicionados conteúdos criados pelos próprios usuários permitindo assim que possa crescer em conteúdo sem ficar limitado a um mesmo desenvolvedor.

2 DESENVOLVIMENTO

Neste trabalho faz-se uma breve introdução a jogos de computador e algumas tecnológicas utilizadas nos mesmos, assim como as ferramentas de desenvolvimento utilizadas durante o desenvolvimento deste trabalho.

2.1 Conceitos Sobre Jogos de Digitais

Um jogo digital (ou videojogo ou jogo eletrônico), termo genérico que se refere a jogos eletrônicos desenhados para serem jogados num computador, num console ou outro dispositivo tecnológico (PIVEC E KEARNEY, 2007), como celulares, pode ser definido como um jogo onde existe interação entre humano e computador, recorrendo ao uso de tecnologia (GEE, 2003).

Os jogos de computador são aplicações que possuem como objetivo a diversão e comodidade do usuário visando o seu entretenimento. Jogos de computador podem receber classificações quanto ao seu tipo e gênero, dentre eles podem ser classificados em jogos 2D, 2.5D e 3D onde jogos 2D possuem apenas imagens em duas dimensões, jogos 2.5D possuem recursos em três dimensões como o cenário, personagens, etc., porem não oferecem todos os recursos de uma aplicação em 3D que oferece recursos de visualizar uma imagem, estrutura, ambiente, personagem, etc., de vários pontos de vista podendo movimentar a “câmera” para vários ângulos diferentes. (Clua, Esteban W.G.; Bittencourt, João R.).

2.1.1 Jogos 3D

Segundo Clua (Clua, Esteban W.G.; Bittencourt, João R.), um jogo 3D é um software especial, pois contém elementos muito variados: módulos de Computação Gráfica, Inteligência Artificial, Redes de Computadores, Multimídia, entre outros. Todos estes módulos devem funcionar em perfeita harmonia, obedecendo a uma característica fundamental de um jogo: deve ser um software em tempo real.

Com o avanço da tecnologia do *hardware* e do *software* hoje em dia é possível produzir imagens gráficas geradas por computador que são capazes de enganar a percepção humana fazendo acreditar que tais imagens são reais, como por exemplo, um cenário criado artificialmente por computação gráfica.

Com o atual poder computacional e a redes de computadores já é possível que jogos sofisticados sejam utilizados em rede.

2.1.2 Jogos em Rede

Segundo o autor do texto JOGOS EM REDE “Jogos por rede são jogos de computador que usam uma rede (Internet ou rede local), geralmente através do protocolo TCP/IP, para permitir uma ligação de utilizadores entre si em jogos multijogadores. No tipo de jogo multijogador podem ser encontrados jogos no estilo de tiro em primeira pessoa, como por exemplo, o *Quake III*, *Counter-Strike* ou *sites* da *web* que usam simplesmente a ligação de rede para promover os usuários com sistemas de pontuação.” (**JOGOS EM REDE).**

Atualmente a muitas das ferramentas e linguagens usadas nos jogos já possui algum método ou função para a transmissão de dados pela rede, sendo assim, possível que o jogador possa jogar com mais pessoas utilizando a rede de computadores, uma das ferramentas que possui estas instruções nativamente é o *Unreal Development Kit.,* com o uso da rede é possível aproveitar recursos disponíveis em outros computadores assim como é feito em *grid computing.*

2.2 *GRID COMPUTING*

O *Grid Computing* é um novo conceito que explora as potencialidades das redes de computadores, com o objetivo específico de disponibilizar camadas virtuais que permitem a um usuário ter acesso a aplicações altamente exigentes, bem como aderir a comunidades virtuais de grande escala, com uma grande diversidade de recursos de computação e de repositórios de informações. [**MORIMOTO, Carlos E**].

Com o *Grid* é possível que a aplicação agregue o recurso de vários computadores de uma rede somando os recursos e tratando isso como se fosse um único recurso de uma quantidade muito maior, como por exemplo, a memoria *RAM* dos computadores ou o espaço em disco. Utilizando essa técnica, pode-se usufruir de alguns benefícios como:

1. Organizações podem agregar recursos
2. Poderosa plataforma de suporte a Organizações Virtuais
3. Acesso distribuído a diversos tipos de recursos
4. Colaboração entre centro de pesquisas
5. Melhor utilização de largura de banda
6. Aproveitamento de recursos ociosos

Isto já é utilizado no *software Unreal Development Kit* durante a renderização do mapa durante o calculo de luz e iluminação do mapa, por este motivo o software foi escolhido para este trabalho.

2.3 *Unreal Development Kit*

O *Unreal Development Kit (UDK),* segundo Jason Busby, Zak Parrish e Jeff Wilson, autores do livro *Mastering Unreal® Technology, Volume I Introduction to Level Design with Unreal® Engine 3,* o *Unreal Engine* usado pela ferramenta *UDK* é um sistema que organiza os seus "*assets*" – personagens, sons, níveis, musicas, vozes, armas, efeitos de som e etc, - em um ambiente gráfico interativo, não "qualquer" ambiente gráfico, mas sim um que se comporta da maneira que você desejar – em outras palavras um *jogo*[**BUSBY, Jason, 2009**]. Todas as informações sobre *Unreal Development Kit,* seus componentes e ferramentas foram retirados do próprio site da ferramenta http://www.udk.com e do livro *Mastering Unreal® Technology, Volume I Introduction to Level Design with Unreal® Engine 3.*[**BUSBY, Jason**].

2.3.1 Componentes do *Unreal Engine*

O *Unreal Engine* é formado por vários componentes que trabalham de forma independente, mas são mantidos em harmonia por uma central *"core engine".* Como todo o sistema é modularizado, licenciados da tecnologia do *Unreal Engine* podem editar ou alterar qualquer componente sem modificar drasticamente o *core engine* (o que aumentaria o custo de desenvolvimento e/ou atrasos na entrega).

- Motor Gráfico

O motor gráfico controla o que você vê enquanto esta jogando um jogo. Sem ele, você veria apenas uma lista enorme de propriedades mudando seus valores, como a localização do jogador, quantidade de vida, armas que esta carregando, munição, animação que estaria sendo executada, etc. Graças ao motor gráfico toda essa informação é traduzida em um ambiente estonteante que você vê enquanto esta jogando como na figura 1.

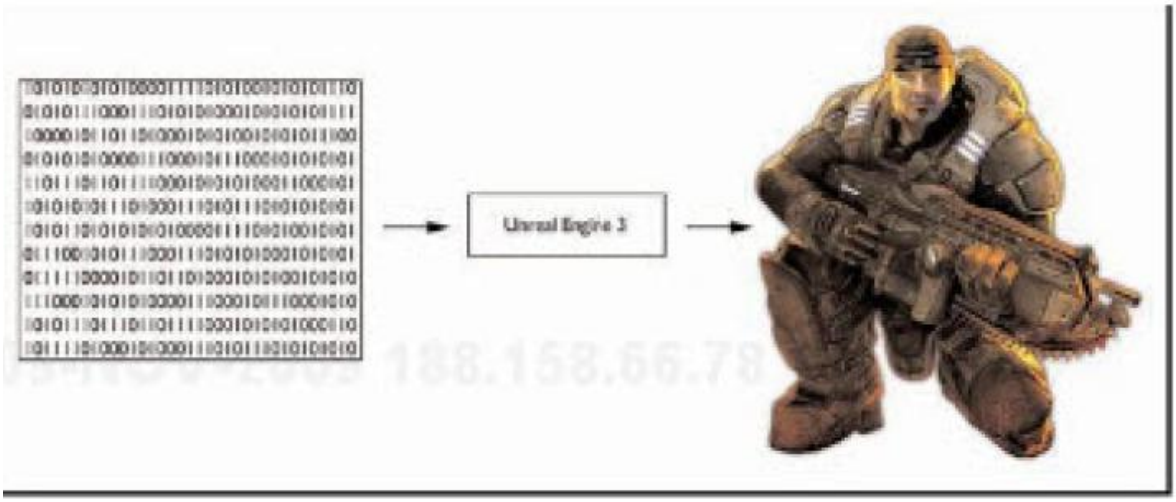

Figura 1 - O motor gráfico é responsável por traduzir uma lista de propriedades em imagens interativas na tela.

O motor gráfico é responsável pelos cálculos “por trás das cenas”, por exemplo, ele determina quais objetos estão na frente de outros, e quais objetos devem “ocultar” outros na visão que aparece na tela.

No *Unreal Engine* 3, quando você oculta objetos que se encontram atrás do ponto de visão, estes objetos não são *renderizados.* Isso libera muitos ciclos de processamento para algo que precise mais, otimizando para um melhor desempenho da aplicação.

➢ Motor Sonoro

É dito que 90% do que você percebe é o que você ouve, até o melhor filme ou jogo perde a sua essência com o som desligado, os jogos atuais possuem muitos sons diferentes na sua programação, que vão desde o som do jogador caminhando ou correndo, vidros quebrando, som das águas de um rio ou chuva, som de ventos, a própria musica ou som ambiente que dão vida ao jogo.

O motor sonoro torna possível todos estes efeitos de som nos jogos. Ele utiliza todos os efeitos sonoros que tenha sido gravado e importado, e utiliza ou ativa eles com base em certos eventos dentro do jogo em si como o som de uma explosão sendo tocado quando uma granada é jogada ou algum veiculo explode.

➢ Motor Físico

O *Unreal Engine 3* utilizado pelo *UDK* não possui um motor de física próprio, mas utiliza o *PhysX* desenvolvido pela *NVIDIA*, este suporta uma grande gama de simulações físicas como, corpos rígidos, *constrained bodies*(organismos restritos), *dynamics skeletal meshes*(malhas de esqueleto dinâmico) e até mesmo simulação de tecidos. Cada uma dessas simulações pode ser interativa no jogo e pode ser manipulada seja pelo jogador diretamente ou por alguma sequencia de *script.* O motor físico pode trabalhar em conjunto com o motor sonoro para gerar sons de impacto por exemplo.

➢ Gerente de Entradas

Sem as entradas do jogador, como as teclas pressionadas pelo usuário, sejam estas em um controle ou usando o teclado e mouse, não existiria um jogo, mas sim apenas uma animação dirigida por um motor de jogo, cada uma dessas entradas deve ser reconhecida e convertida em comandos que dizem ao jogo como reagir. Este trabalho é do gerente de entradas ou *input manager* dentro do *Unreal Engine 3*.

➢ Infraestrutura de rede

Durante uma partida *multiplayer* usando a rede, cada computador de cada jogador esta em constante comunicação com outro computador que esta agindo como o servidor. Este computador servidor também pode executar o jogo como um cliente e fazer parte da mesma partida que esta sendo executada no servidor. Quando um computador esta sendo usado estritamente como servidor sem nenhum cliente sendo executado, este é chamado de “servidor dedicado”. Servidores dedicados são geralmente preferidos para um jogo *online* pois dessa forma pode aproveitar todos os seus recursos para executar o servidor.

O ponto chave para uma eficiência em jogos através da rede é limitar a informação enviada pela rede para apenas as informações importantes em relação ao jogo, como a posição do jogador e sua interação com o ambiente que se encontra. Enviando apenas os detalhes relevantes ao jogo pela rede, é possível ter um jogo rápido mesmo em conexões lentas, o *Unreal Engine* foi desenvolvido para fazer isso.

➢ Interpretador de *UnrealScript*

UnrealScript é um dos aspectos mais revolucionários do *Unreal Engine*. É uma linguagem de *script* que permite que programadores ou usuários ajustem virtualmente qualquer coisa que esteja sendo executada pelo *Engine*, sem a necessidade de alterar o código fonte do jogo. Esta linguagem foi criada com a intenção de ser relativamente fácil de usar, é muito parecida com C++ e Java, e seu criador Tim Sweeney diz que preferiu escolher a simplicidade do que a velocidade de execução.O interpretador de *UnrealScript* é

um componente do *engine* responsável por transformar o *UnrealScript* criados previamente em códigos que podem ser processados pelo *engine*.

- *Kismet*

Kismet é uma ferramenta de programação visual que faz parte do *Unreal Development Kit*, esta ferramenta é utilizada durante o desenvolvimento dos mapas e pode ser acessada através do *Unreal Editor*, pode ser utilizadas para facilitar a programação utilizando eventos, variáveis e etc. [***KISMET HOME]***

- *Unreal Editor*

O Unreal Editor, como pode ser visto na figura 2, é a ferramenta utilizada para criar os mapas, adicionar objetos, personagens, *static meshes* e vários outros *assets* que serão utilizados no mapa. Este foi utilizado para criar alguns mapas para serem utilizados pelo jogo em desenvolvimento.***[UNREAL EDITOR AND TOOLS]***

1. *Menu bar:*
2. *Toolbar*
3. *Toolbox*
4. *Viewports*
5. *Status bar*

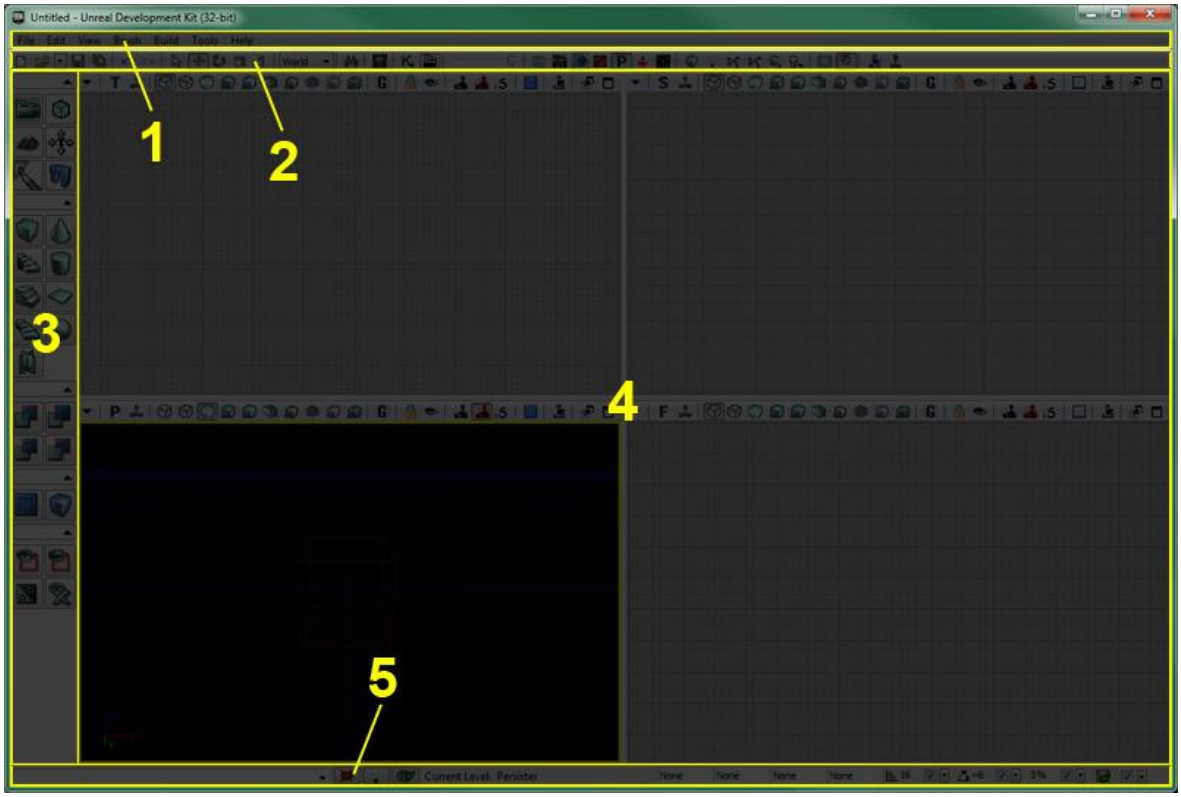

Figura 2 - Janela principal do Unreal Editor

A barra de menu ou *menu bar* no editor de nível é algo familiar a qualquer pessoa que já utilizou aplicações para Windows anteriormente. Ela provê acesso a uma grande gama de ferramentas e comandos que são necessários no processo de criar um nível com o *Unreal Editor* [***UNREAL EDITOR USER GUIDE***]

A barra de ferramentas, como na maioria das aplicações, é um grupo de comandos que provê acesso rápido aos comandos e ferramentas mais comumente usados na aplicação. Muitos dos itens encontrados nos menus do editor de níveis também podem ser encontrados como botões na barra de ferramentas. [***UNREAL EDITOR USER GUIDE***]

A caixa de ferramentas é um conjunto de ferramentas usadas para controlar o modo que o editor de níveis esta trabalhando, reformular *builder brush*, criar novos volumes e geometrias de *BSP* e controlar a visibilidade e seleção dos atores pelos *viewports*. [***UNREAL EDITOR USER GUIDE***]

Viewports no editor de níveis são as janelas para o mundo que esta sendo criado no editor. Oferece múltiplas visões ortográficas (Topo, Lateral, Frontal) e uma visão em perspectiva, lhe da controle completo sobre o que pode ser visto e como é visto. [***UNREAL EDITOR USER GUIDE***]

O *Unreal Editor* é utilizado para gerar o mapa ou cenário onde o jogo se situa, as classes e o código usado no jogo é feito com a ajuda da ferramenta de desenvolvimento da Microsoft, o *Visual Studio 2010*.

2.3.2 *Visual Studio 2010*

O *Microsoft Visual Studio 2010 Professional* é a ferramenta essencial para indivíduos que executam tarefas de desenvolvimento básicas. Ele simplifica a criação, depuração e implantação de aplicativos em uma variedade de plataformas que incluem o SharePoint e a nuvem. O Visual Studio 2010 Professional é fornecido com suporte integrado para desenvolvimento orientado a testes, bem como ferramentas de depuração que ajudam a garantir soluções de alta qualidade. [***VISUAL STUDIO 2010 PROFESSIONAL***]

A solução de desenvolvimento *Visual Studio 2010 Professional* pode ser adquirida graças a parceria da Universidade São Francisco com a *Microsoft* que disponibiliza vários programas de forma gratuita aos alunos de cursos de tecnologias através do MSDNAA.

A ferramenta *Visual Studio* sozinha não reconhece a linguagem *UnrealScript* que foi utilizada na programação das classes do jogo desenvolvido, para isso foi necessário adicionar um *plug-in* recomendado e usado pela própria *Epic Games,* criadora do *Unreal Engine,* o *nFringe*[***NFRINGE***] que adiciona ao *Visual Studio* a possibilidade de criar *scripts* e reconhecer a formatação da linguagem *UnrealScript* e reconhecer seus erros na estrutura do código.

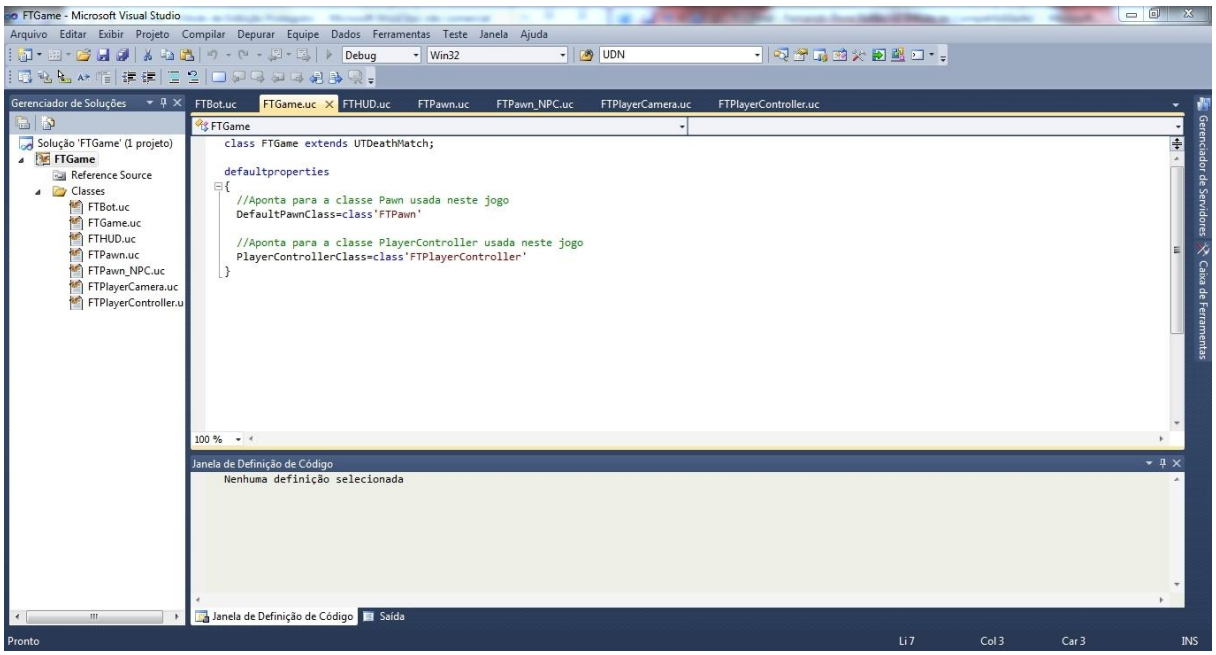

Figura 3 - Microsoft Visual Studio

3 METODOLOGIA

Este trabalho utilizou uma ferramenta de desenvolvimento de jogos gratuita para estudos e fins não lucrativos, esta ferramenta utiliza a *Unreal Engine 3* vencedora de vários prêmios e utilizada no desenvolvimento de vários jogos de alta qualidade hoje em dia como as séries de *Assassin's Creed, Gears of War, Unreal Tornament,* entre outros

O programa *Unreal Development Kit* pode ser adquirido na página do próprio site (www.udk.com). Esta ferramenta é capaz de gerar jogos com qualidade impressionante e possui varias ferramentas internas que facilitam a criação de mapas, animações, vídeos, ambiente com folhagem, terrenos, movimento de personagens, e programação do jogo em geral, uma das ferramentas que foi utilizada neste desenvolvimento foi o Kismet, que ajuda o usuário a criar programações de eventos, ações e condições através da programação de script visual. Com esta ferramenta a programação de eventos e outras ações é muito mais simples de ser feita mesmo sem grandes conhecimentos de programação por linha de código.

3.1 *Kismet*

A ferramenta *Kismet* de programação visual foi utilizada no desenvolvimento deste jogo para permitir que seja definido no gatilho que ativará o teletransporte para outro mapa hospedado em outro servidor utilizando para isso o comando de *console "open ipaddress"* onde o *IP* do servidor ao qual o jogador deseja se conectar poderá ser inserido pela interface que será criada no trabalho desenvolvido por Thomas Cintra Penteado, o qual criará uma interface que usará *actionscript e scaleform* para criar um campo na tela onde o usuário poderá inserir o *IP* e esta informação será adicionada a uma variável global do jogo, que por sua vez será utilizado pelo *kismet* para executar o comando de console, figura 4.1 e 4.2 que fará o cliente se conectar ao outro servidor na rede.

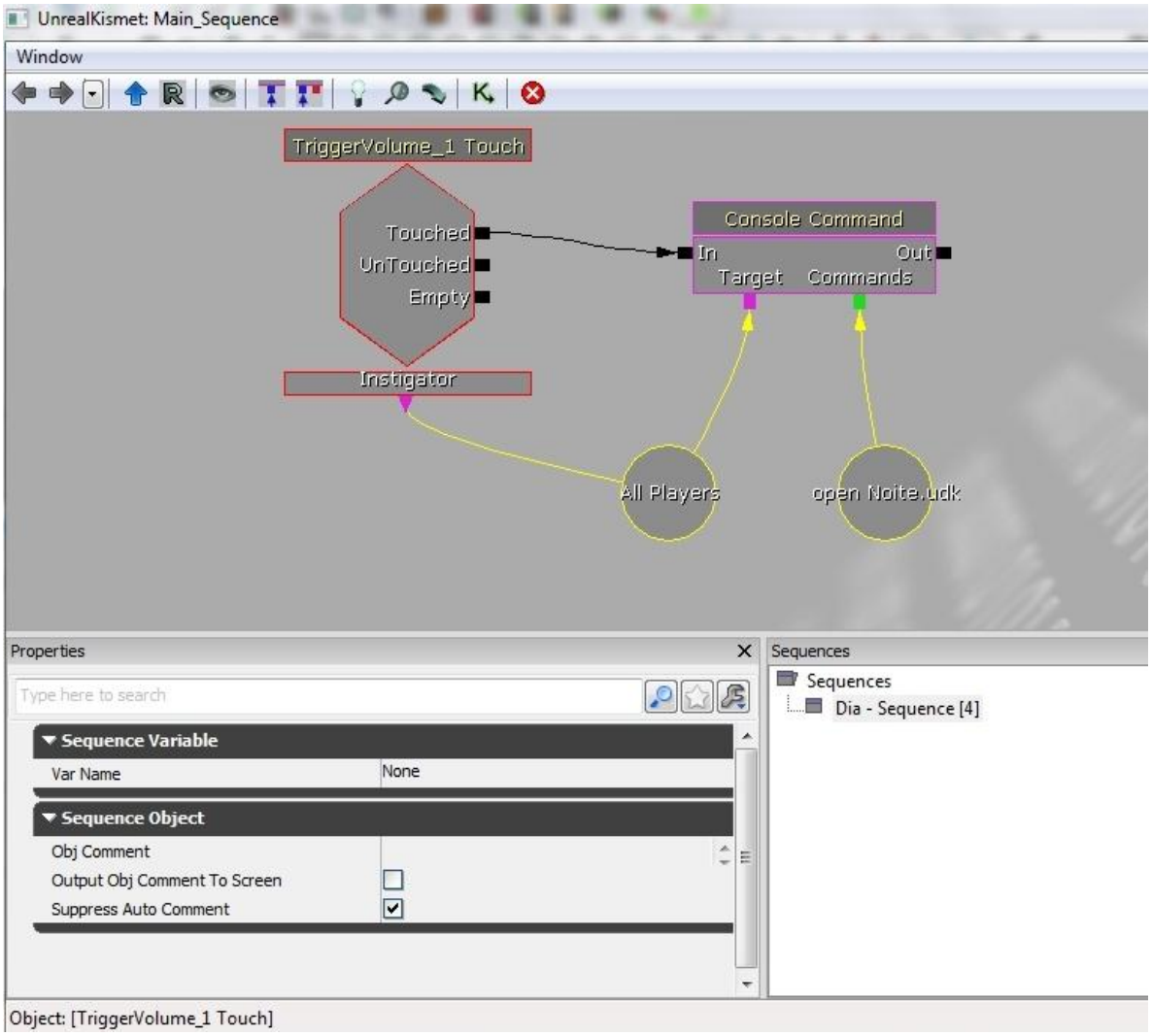

Figura 4.1 - Exemplo de *Script* utilizando *Kismet*

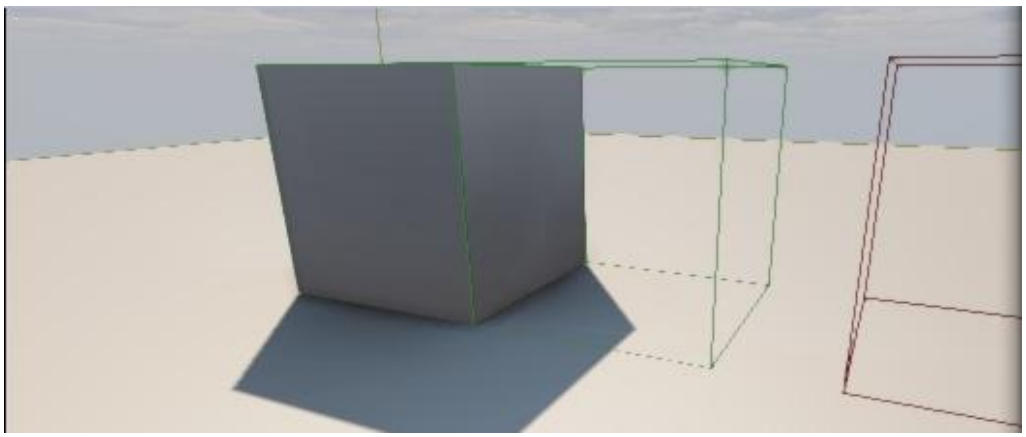

Figura 4.2 - Trigger Volume em verde usado para ativar o script

Para que isso seja possível foi criado um *script* Figura 4.1 simples onde uma variável guarda o comando que será executado, e um *Trigger Volume* Figura 4.2 foi criado dentro do

mapa, este *Trigger Volume* ativará o *script* assim que um jogador toca-lo, e o *script* por sua vez executará o comando enviando o jogador para outro mapa como podemos ver a seguir na Figura 4.3 – Mapa inicial e Figura 4.4 – Mapa após ativar o *Trigger*.

Figura 4.3 - Mapa inicial

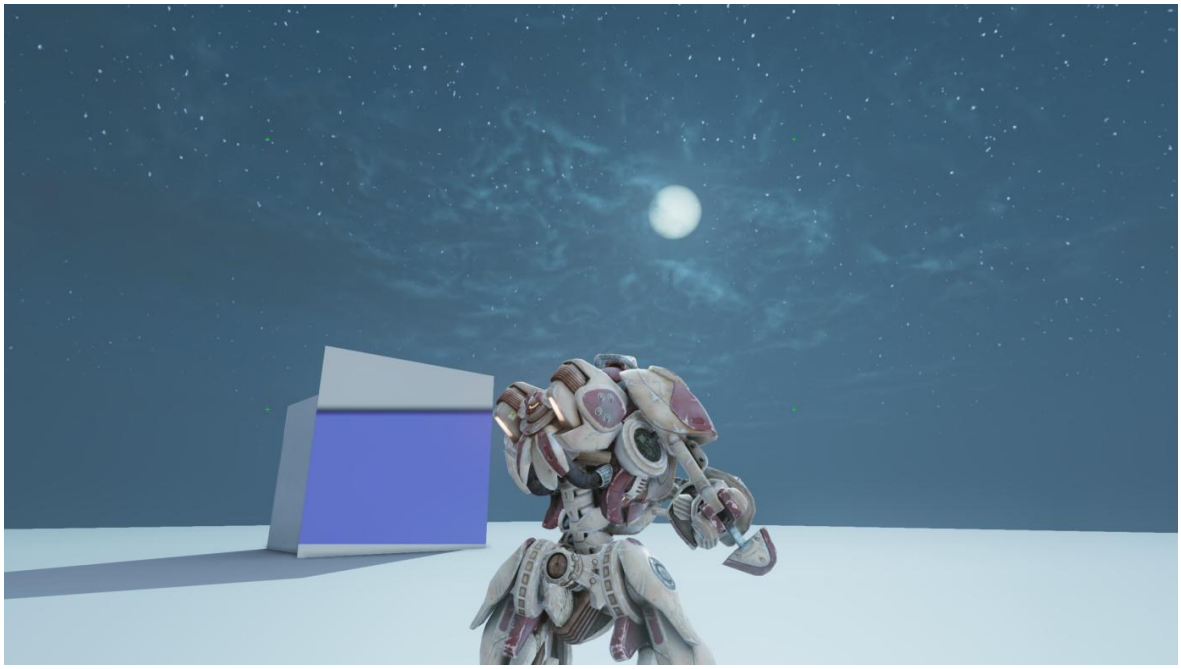

Figura 4.4 - Mapa após a ativação do *Trigger*

3.2 *Unreal Editor*

Pelo *Unreal Editor pode-se* criar alguns mapas simples para serem utilizados para testes de comunicação do jogo e para o aprendizado no uso da ferramenta, também foram utilizados alguns exemplos de mapa já prontos que acompanham a ferramenta em sua instalação, estes possuem uma qualidade de detalhes muito maiores e já são optimizados para jogos em rede para até 64 jogadores.

3.3 Configurações do ambiente

Antes de o projeto começar a ser desenvolvido é necessário fazer algumas configurações para o inicio do desenvolvimento, para isso foram seguidos alguns passos:

1. Instalar o *UDK* mais recente que pode ser adquirido no site www.udk.com
2. Instalar o *Visual Studio 2010 Professional*
3. Instalar o *nFringe*
4. Navegar até a pasta <caminho de instalação do *UDK*>\Develpment\Src e criar uma pasta onde serão salvos os arquivos de *script* do projeto, neste caso foi usada a pasta FTGame, dentro desta criar a pasta Classes o resultado sera: <caminho de instalação do *UDK*>\Development\Src\FTGame\Classes
5. Abrir o *Visual Studio* e criar um novo projeto utilizando o modelo de *UnrealScript* selecionando a pasta anteriormente para o local onde serão salvos os arquivos do projeto.
6. Após criada a estrutura, é necessário criar alguns *scripts* básicos para definir algumas configurações do jogo, foram criados os *scripts FTGame.uc*, *FTHUD.uc, FTPawn.uc, FTPlayerController.uc*
7. Criados os *scripts* é necessário configurar o *Unreal Editor* para utiliza-los para o novo jogo, para isso é necessário editar o arquivo *DefaultEngine.ini* localizado na pasta <instalação do *UDK*>\UDKGame\Config\DefaultEngine.ini e navegar até a área onde

diz: [UnrealEd.EditorEngine] e adicionar a linha +EditPackages=<Nome da pasta do projeto>

8. Feito o passo 7, ao abrir o *Unreal Editor*, este informará que os *scripts* precisam ser recompilados, depois do processo ter sido feito basta criar um novo mapa, navegar para o menu *View -> World Properties,* expandir a opção *Game Type* e selecionar em *Default Game Type e Game Type for PIE* a classe criada anteriormente, neste caso *FTGame*.

Seguidos estes passos, o ambiente de desenvolvimento estará pronto para criar níveis utilizando o modelo de jogo que for definido na classe *FTGame.uc*.

3.4 Diagrama

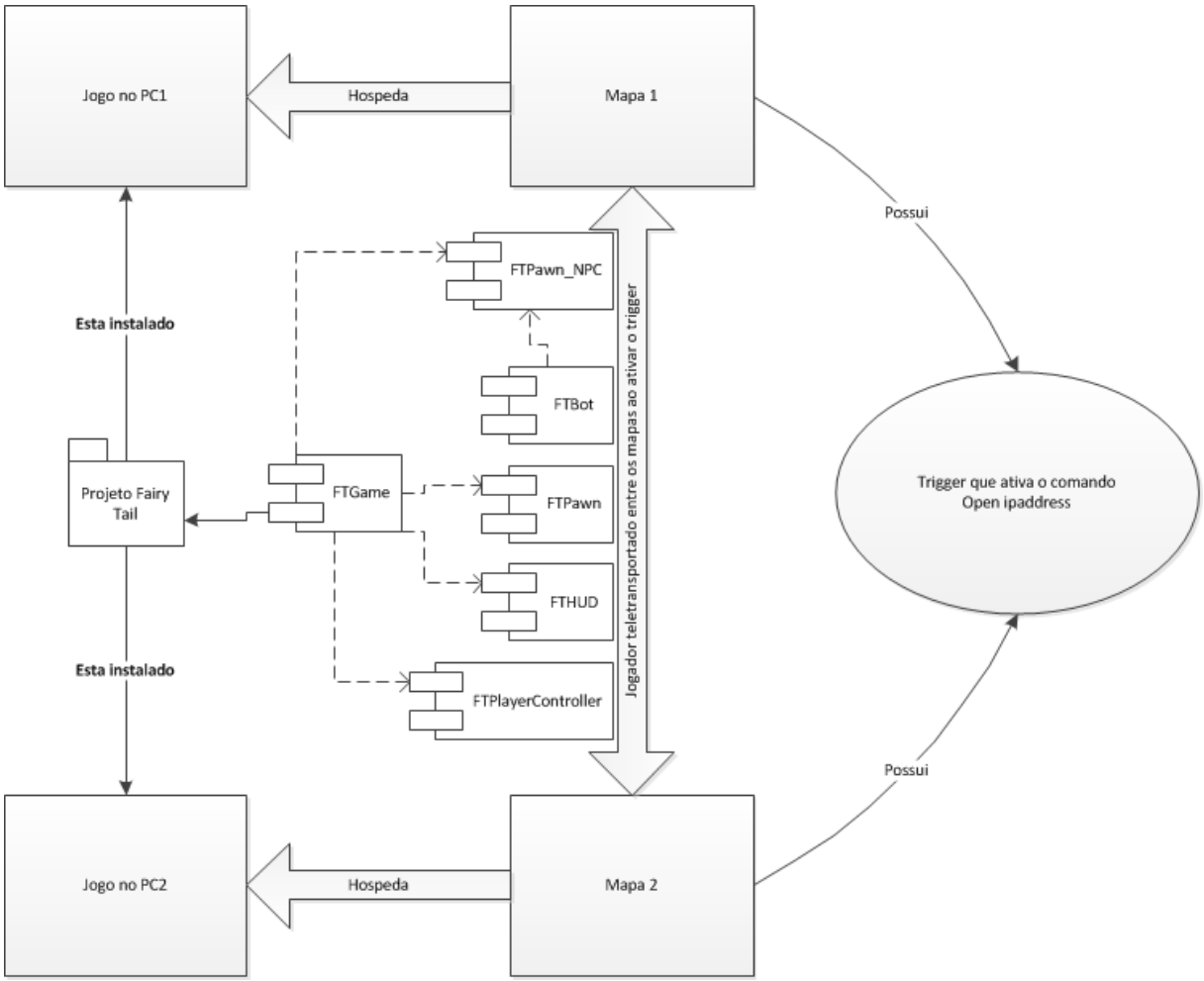

Figura 5 - Diagrama de funcionamento da aplicação

Como é visto neste diagrama, o aplicativo, chamado de “Projeto Fairy Tail”, será instalado em pelo menos dois computadores, o aplicativo é formado pelas classes *FTGame, FTPawn, FTHUD, FTbot, FTPawn_NPC e FTPlayerController*, estas classes são interdependentes e controlam algumas configurações do jogo. Cada um dos computadores onde o jogo estiver instalado hospedará um mapa diferente que possui um componente *Trigger* que ao ser ativado executa o comando de *console* “*open ipaddress”* onde o “*ipaddress*” é substituído pelo IP do outro computador na rede. Ao executar este comando, o aplicativo abre uma conexão de rede com o aplicativo no outro computador permitindo que o jogador seja transportado para o mapa hospedado no outro computador.

3.5 Classes

As classes são arquivos de código em *Unreal Script* com definições de padrões do jogo que serão utilizadas durante a execução do jogo, definindo o seu comportamento, comandos, personagens, inteligência artificial e etc.

3.5.1 *FTGame*

Esta classe define o modelo de *GameType* que será utilizado pelo jogo e define as classes que serão utilizadas por padrão no jogo,com isso é possível definir o estilo de jogo, como o personagem deverá se comportar, quais serão os comandos que o jogador poderá executar e etc.

```
class FTGame extends UTDeathMatch;

defaultproperties
{
  //Aponta para a classe Pawn usada neste jogo
  DefaultPawnClass=class'FTPawn'

  //Aponta para a classe PlayerController usada neste jogo
  PlayerControllerClass=class'FTPlayerController'
}
```

Código 1 – *FTGame.uc*

3.5.2 *FTHUD*

A classe *HUD* será usada pelo jogo para definir a interface usada durante o jogo, quais informações estarão disponíveis na tela, como a vida do personagem, a mira da arma, entre outras coisas, esta classe ainda não foi definida, pois a interface ainda esta em planejamento.

```
class FTHUD extends HUD;

defaultproperties
{
      //Será usada para definir a interface futuramente
}
```

Código 2 - FTHUD.uc

3.5.3 *FTPawn*

O código 3 apresenta a classe *Pawn* usada pelo jogo, a principio esta classe, utiliza as variáveis *ElapsedRegenTime, RegenAmount e RegenTime* para definir o tempo de regeneração e quantidade de saúde regenerada pelo personagem do jogador, esta classe é utilizada no jogo para definir o personagem do jogador, como esta classe esta extendendo a classe *UTPawn* que já vem pronta de exemplo junto com o *UDK* o personagem padrão do jogo e suas características serão semelhantes aos usados pelo personagem usado no jogo *Unreal Tornament*.

```
class FTPawn extends UTPawn;

var float ElapsedRegenTime;
var float RegenAmount;
var float RegenTime;

event Tick(float DeltaTime)
{
  //Calcula o tempo que se passou
  ElapsedRegenTime += DeltaTime;

  //passou o tempo suficiente?
  if(ElapsedRegenTime >= RegenTime)
  {
    //Cura a saúde do "Pawn" e reinicia o contador
    HealDamage(RegenAmount, Controller, class'DamageType');
    ElapsedRegenTime = 0.0f;
  }
}

defaultproperties
{
  //Define as propriedades padrões de regeneração de saúde
  RegenAmount=2
  RegenTime=1
}
```

Código 3 - FTPawn.uc

3.5.4 *FTPlayerController*

PlayerController é a classe usada neste jogo para definir a aparência do personagem do jogador e a câmera usada para definir a visão do personagem. Devido ao pouco tempo para o desenvolvimento do jogo e seus personagens, aqui também foi utilizada uma extensão de uma classe que já acompanha o *UDK*, ela já define os comandos que podem ser passados pelo jogador e como o personagem responde a eles.

```
class FTPlayerController extends UTPlayerController;

var class<UTFamilyInfo> CharacterClass;

simulated event PostBeginPlay()
{
  super.PostBeginPlay();

  SetupPlayerCharacter();
}

/** Definir a classe de informação de personagem do jogador e executar outra inicialização */
function SetupPlayerCharacter()
{
  //Define o personagem
  ServerSetCharacterClass(CharacterClass);
}

defaultproperties
{
  //Aponta para a classe UTFamilyInfo para o personagem
  CharacterClass=class'UTFamilyInfo_Liandri_Male'
        CameraClass=class'FTGame.FTPlayerCamera'
}
```

Código 4 – FTPlayerController.uc

3.5.5 *FTPawn_NPC*

Neste código 6 foi definido um personagem do jogo da mesma forma que o personagem do jogador, porem alguns detalhes a mais são definidos aqui, como o padrão de animação do personagem, qual o seu “esqueleto” que define como o personagem é capaz de se mover, a física que será aplicada ao personagem, e define que este personagem não será controlável e estará no mapa para combater o jogador. Esta classe ainda não esta finalizada, apesar de estar funcionando será modificada futuramente ou criadas varias classes diferentes para definir diferentes monstros que serão adicionados ao jogo.

```
class FTPawn_NPC extends UTPawn
  placeable;

var(NPC) SkeletalMeshComponent NPCMesh;
var(NPC) class<AIController> NPCController;

simulated event PostBeginPlay()
{
  if(NPCController != none)
  {
    //Define a classe ControllerClass existente para a nova classe NPCController
    ControllerClass = NPCController;
  }

  Super.PostBeginPlay();
}

//Substituir para fazer nada
simulated function SetCharacterClassFromInfo(class<UTFamilyInfo> Info)
{
}

defaultproperties
{
  //Constroi o NPC Padrão, com o seu "esqueleto", padrão de simulação de fisica e animações.
  Begin Object Class=SkeletalMeshComponent Name=NPCMesh0
    SkeletalMesh=SkeletalMesh'CH_LIAM_Cathode.Mesh.SK_CH_LIAM_Cathode'
    PhysicsAsset=PhysicsAsset'CH_AnimCorrupt.Mesh.SK_CH_Corrupt_Male_Physics'
    AnimSets(0)=AnimSet'CH_AnimHuman.Anims.K_AnimHuman_BaseMale'
    AnimtreeTemplate=AnimTree'CH_AnimHuman_Tree.AT_CH_Human'
  End Object
  NPCMesh=NPCMesh0
  Mesh=NPCMesh0
  Components.Add(NPCMesh0)

  //Aponta para a classe AIController customizada - Como valor padrão
  NPCController=class'FTBot'
}
```

Código 5 - FTPawn_NPC

3.5.6 *FTBot*

Define a inteligência artificial usada pela classe *FTPawn_NPC* que define as ações que serão tomadas pelo personagem dentro do jogo. Como o estado que fará o monstro vagar pelo mapa a procura do jogador e o que fazer quando encontrar outro personagem dentro do jogo, isto será usado para definir diferentes tipos de ações a serem tomadas, por diferentes classes de personagens, desde monstros a personagens que irão compor cidadãos de uma cidade dentro do jogo. Atualmente, a classe possui apenas o estado *Roaming* que faz com que o personagem fique vagando pelo mapa até que encontre outro personagem e o ataque.

```
class FTBot extends UTBot;

var Actor Destination;

protected event ExecuteWhatToDoNext()
{
  //Vai para o estado de "Roaming"
  GotoState('Roaming');
}

state Roaming
{
Begin:
  //Se o jogo acaba de começar ou se alcançou o destino
  //escolhe um novo destino de forma aleatória
  if(Destination == none || Pawn.ReachedDestination(Destination))
  {
    Destination = FindRandomDest();
  }

  //Encontrar um caminho para o destino e mover para o próximo nó no caminho
  MoveToward(FindPathToward(Destination), FindPathToward(Destination));

  //Dispara o proximo Loop de decisão
  LatentWhatToDoNext();
}

defaultproperties
{
}
```

Código 6 - FTBot.uc

4 Problemas

Durante os testes do jogo em rede, houve alguns problemas na comunicação via rede em um dos computadores onde não foi possível descobrir o que causa o bloqueio de conexões de entrada impedindo que outros jogadores se conectem ao servidor dedicado hospedado neste computador.

O problema foi contornado para os testes do jogo simplesmente mantendo o servidor dedicado em outro computador e executando o jogo pelo *Unreal Editor* o que permitiu que o jogo se conecta-se ao servidor sem problemas pelo gatilho criado no mapa.

A falta de tempo e habilidade com modelagem 3D, assim como um software para a modelagem de personagens, impediu que fossem criados personagens e animações para serem usadas neste trabalho, ficando assim para ser desenvolvido em trabalhos futuros.

5 Testes e Resultados

Para os testes foram utilizados 2 notebooks pessoais, um com configurações acima do recomendado para a execução do jogo, e outro com a configuração abaixo do recomendado.

Notebook1: Alienware M11x R2, processador Intel i7 1.2GHz, 8GB de RAM DDR3, placa de vídeo Nvidia GeForce 335M 1GB.

Notebook2: HP Pavilion dv6, processador Intel i3 2.13GHz, 4GB de RAM DDR3, placa de vídeo Intel HD Graphics integrada com 256MB de memoria compartilhada.

Ambos os notebooks estão rodando a mesma versão do jogo, sistema operacional Windows 7 Ultimate 64 bits e HD SATA2 de 500GB e ambos usando a mesma conexão de rede sem fio para se conectarem em rede.

Obs: o servidor do jogo foi executado no notebook2 pois por algum motivo o notebook1 esta bloqueando a conexão ao servidor do jogo, mesmo assim os testes puderam prosseguir.

O jogo foi pré-configurado em seu instalador para que o mapa “Dia” possuísse um *Volume Trigger* que ao ser tocado pelo jogador, abriria uma conexão com o servidor sendo executado no notebook2 como podemos ver nas Figuras 5.1 e 5.2.

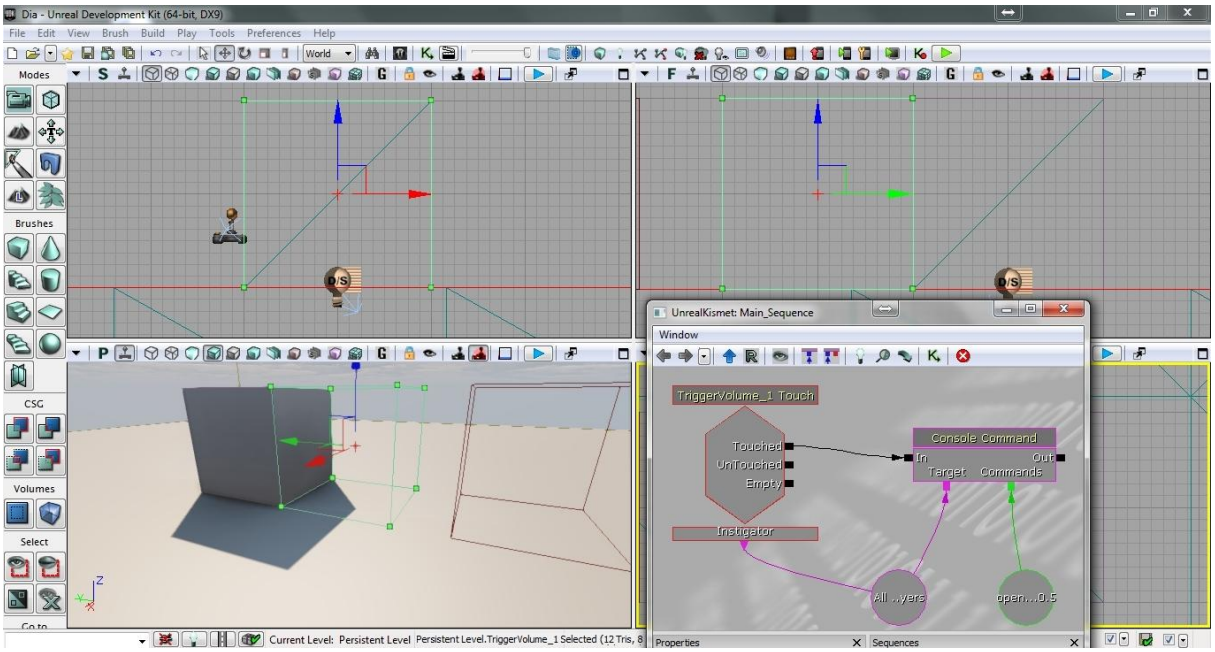

Figura 6.1 - *TriggerVolume* no mapa Dia

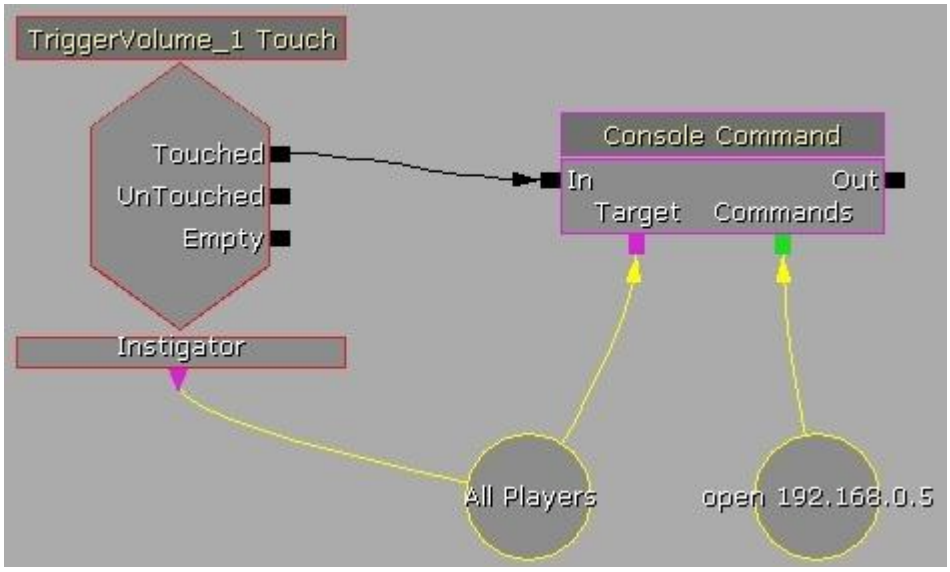

Figura 6.2 - Programação via *Kismet* no mapa Dia

Com o *script* definido via *Kismet* no mapa Dia, como pode ser visto na Figura 9, quando qualquer jogador tocar o *TriggerVolume_1* um comando de console é executado no jogo fazendo a conexão com o *IP* 192.168.0.5, neste caso o *IP* da rede que aponta para o notebook2 onde o servidor do jogo estava em execução com o mapa *EpicCidatelFTGame*.

A Figura 5.3 e 5.4 logo a seguir demonstra o que ocorre quando o jogador entra em contato com o *TriggerVolume_1* e ativa o comando do *script*.

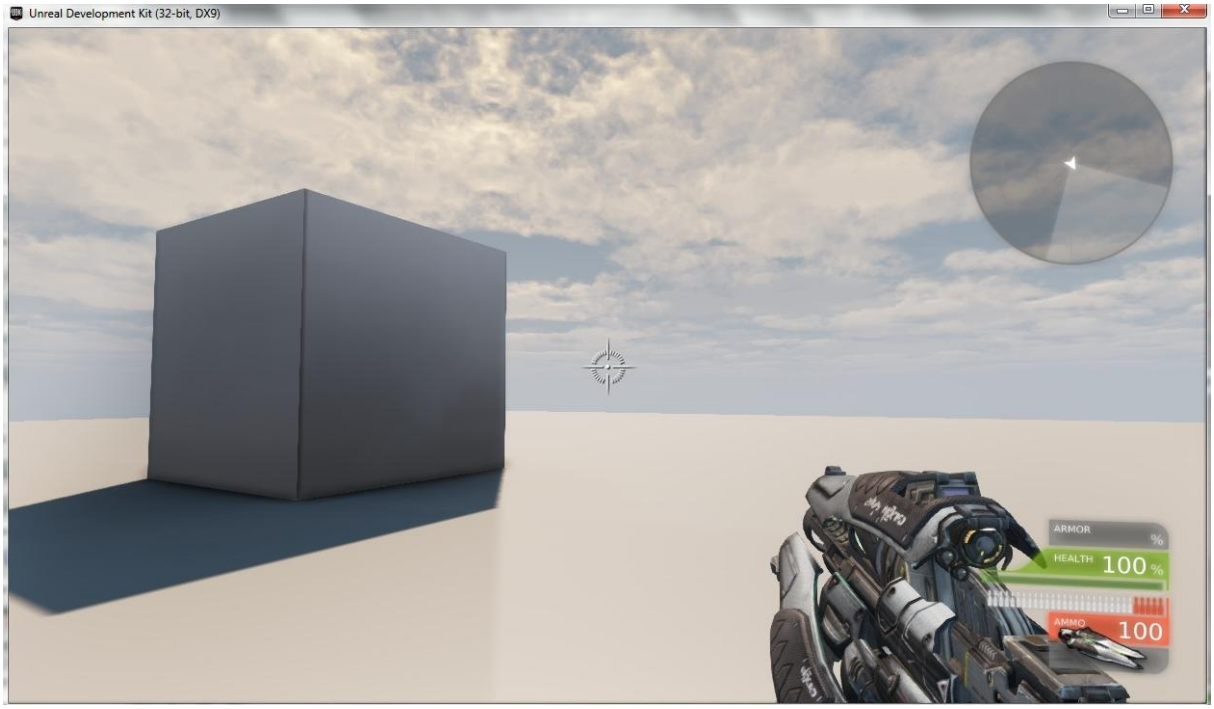

Figura 6.3 - O *TriggerVolume_1* é invisível para o jogador durante o jogo

Durante a execução do jogo figura 5.3, o *VolumeTrigger* não é visível para o jogadores, ao tocar o *VolumeTrigger o comando é ativado e inicia-se o carregamento do mapa no servidor em outro computador figura 5.4*

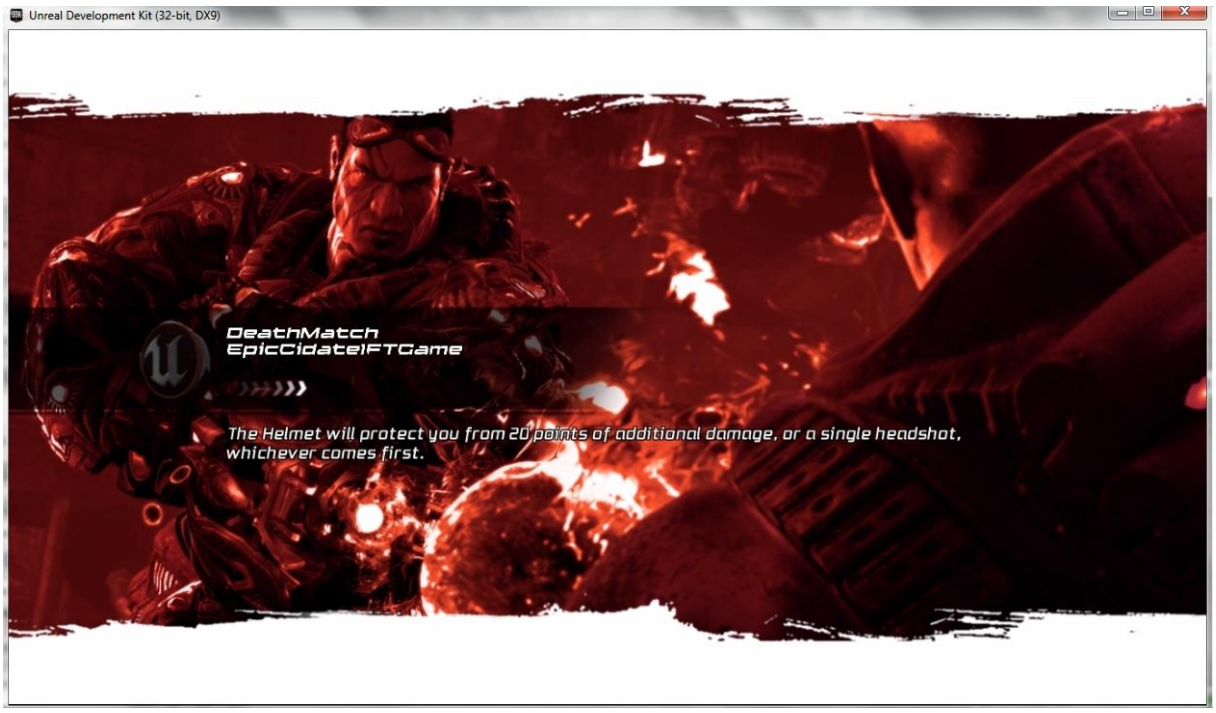

Figura 6.4 - Jogo se conectando ao mapa hospedado pelo servidor no notebook2

Como o jogo esta sendo executado em uma rede local de alta velocidade o jogo se conecta ao servidor em poucos segundos, mesmo o mapa hospedado possuindo mais de

130MB. O servidor foi preparado para o teste com 1 jogador e mais 2 *NPCs* totalizando no total 4 personagens no mapa contando o jogador conectado através de rede. Por experiência com outros jogos que utilizam redes mais lentas como adhoc ou internet para se conectar, já há um conhecimento prévio de que estes jogos funcionam bem com 4 a 8 jogadores conectados sem problemas, assim sendo, este jogo terá um limite de 4 a 8 personagens por missão em cada mapa, podendo o seu limite ser aumentado ou reduzido dependendo do desempenho que obtiver em testes em maior escala, ou em mapas com objetivo de promover a interação entre os jogadores, como uma cidade ou um mercado para a troca de itens entre os jogadores.

Figura 6.5 - Visão do jogador no notebook1 com o personagem do jogador do notebook2 no centro da tela

É possível ver na figura 5.5 que ambos os jogadores podem interagir dentro do mesmo jogo, que apesar de ser hospedado pelo servidor em um notebook com capacidades mais modestas, não apresentou lentidão em nenhum dos dois computadores mesmo no notebook2 onde se concentrava a carga de executar o servidor e o cliente do jogo ao mesmo tempo.

Dessa forma foi possível demonstrar que é possível criar um jogo com seus mapas distribuídos em servidores em rede aliviando o processamento e a carga de rede sobre cada um, assim sendo o resultado obtido com este projeto é um jogo de computador distribuível

entre computadores em uma rede local desenvolvido com *Unreal Development Kit* em um curto espaço de tempo e com uma qualidade gráfica considerável para os dias atuais.

Como o jogo pode ser distribuído em rede e cada servidor fica em uma maquina diferente obtivemos uma aplicação com boa tolerância a falhas, pois se um servidor parar de responder, os jogadores podem continuar jogando em outro mapa hospedado por outro servidor, também é possível se ter um aproveitamento melhor dos recursos, pois o maior espaço usado pelo jogo é o tamanho do mapa desenvolvido e cada mapa fica armazenado em um computador diferente dessa forma o espaço utilizado pelo jogo instalado e a memoria usada enquanto o mesmo esta em execução é reduzida se considerarmos um jogo com todos os mapas diferentes sendo salvos na mesma maquina. Também não é necessária a aquisição de uma maquina especifica para hospedar os mapas já que o próprio jogo também age como servidor dedicado.

Com o processamento e a carga de rede distribuída entre diferentes servidores na rede, cada um permitindo uma quantidade razoável de jogadores é possível evitar lentidões devido ao alto trafego de dados na rede, e ao processamento excessivo pelo servidor e pelos clientes ao tentar processar uma grande quantidade de personagens em um mesmo local no mapa.

6 Conclusão

É possível concluir com este trabalho que recursos computacionais ociosos podem ser aproveitados, assim sendo é possível utilizar tecnologias de desenvolvimento de jogos da atualidade para desenvolver aplicações que possam ter características de aplicativos de *grid computing* de forma a aproveitar estes recursos ociosos em aplicativos de entretenimento como jogos de computador, obtendo assim aplicativos com uma qualidade muito boa em termos de realismo gráfico e ainda assim poupando a aquisição de equipamentos mais modernos e mais caros.

Durante o desenvolvimento deste projeto, apesar de algumas dificuldades como falta de tempo, falta de conhecimento sobre a linguagem e problemas de comunicação de rede entre os computadores, foi possível provar que um jogo de computador distribuível em rede pode ser desenvolvido de maneira relativamente fácil com conteúdo expansível capaz de ser criado por qualquer jogador com um pouco de criatividade e poucas instruções sobre o funcionamento do mesmo em rede através da programação visual do *Kismet,* que facilitou o seu desenvolvimento. Com isso foi possível desenvolver um jogo 3D expansível e gratuito, que se aproveita dos recursos distribuídos entre os computadores da rede mesmo possuindo poucos recursos e pouco tempo para o desenvolvimento da aplicação.

7 Trabalhos Futuros

Como trabalho futuro, será necessário criar mapas, personagens, armas, animações e interface para o jogo de forma que permitam acrescentar conteúdo que promova a diversão dos jogadores e o trabalho em equipe.

É necessário criar monstros de diferentes níveis para promover desafio ao jogador, sistema de experiência e possibilidade de criar o próprio personagem modular e dinâmico que permita ao jogador alterar a aparência durante o jogo.

Precisa-se criar animações, diálogos, e tutoriais para ensinar aos jogadores o funcionamento do jogo, e os principais comandos.

8 REFERÊNCIAS

BUSBY, Jason, PARRISH, Zak, WILSON, Jeff. **MASTERING UNREAL TECHNOLOGY Volume 1: Introduction to Level Design with Unreal Engine 3**. Sarris Publishing, 800 East 96th St., Indianapolis, Indiana, 46240 USA 2009, 856 pág.

CLUA, ESTEBAN W.G.; BITTENCOURT, JOÃO R. **Desenvolvimento de Jogos 3D: Concepção, Design e Programação.** Centro de Ciências Exatas e Tecnológicas Universidade do Vale do Rio dos Sinos (UNISINOS)

PITANGA, Marcos. Computação em Grade - Uma Visão Introdutória Disponível em: <http://www.clubedohardware.com.br/artigos/124/2>. Acessado em 05 de julho de 2011

JOGOS EM REDE. Jogos em rede. Disponível em: < http://jogosemrede.zip.net/index.html>. Acesso em: 05 de junho de 2011

MORIMOTO, Carlos E. ***Dicionário técnico: Grid Computing*.** Disponível em: <http://www.hardware.com.br/termos/grid-computing> Acesso em: 18 de junho de 2011

Curso Grid Computing. Intel Next Gerenation Center oferece diversos cursos sobre tecnologias inovadoras. Disponível em: < http://www.nextg.com.br/courses.php?id_course=29>. Acesso em: 13 de março de 2011

O que é middleware. Rede Nacional de Ensino e Pesquisa. Disponível em: < http://www.rnp.br/noticias/2006/not-060926.html >. Acesso em: 13 de março de 2011

What is UDK?. *UNREAL DEVELOPMENT KIT.* Disponível em: < http://www.udk.com/> . Acesso em: 04 de junho de 2011

Unreal Tournament 3 Official Trailer. *Canal Viso Games no Youtube* Disponível em: <http://www.youtube.com/watch?v=if6Vv-5AlHs> Acesso em: 5 de junho de 2011

GAMEPLAY ELEMENTS. *UNREAL DEVELOPMENT NETWORK.* Disponível em: < http://udn.epicgames.com/Three/GettingStartedGameplay.html> Acesso em: 04 de junho de 2011

PROGRAMMING. *UNREAL DEVELOPMENT NETWORK.* Disponível em: < http://udn.epicgames.com/Three/GettingStartedProgramming.html> Acesso em: 04 de junho de 2011

UNREAL EDITOR USER GUIDE**,** *UNREAL DEVELOPMENT NETWORK*. Disponível em: < http://udn.epicgames.com/Three/UnrealEdUserGuide.html > Acesso em: 23 de outubro de 2011

KISMET HOME, *UNREAL DEVELOPMENT NETWORK.* Disponível em: < http://udn.epicgames.com/Three/KismetHome.html > Acesso: em 15 de novembro de 2011

UNREAL TECHNOLOGY SHOWCASE, *UNREAL TECHNOLOGY.* Disponível em: < http://www.unrealengine.com/showcase > Acesso em: 15 de novembro de 2011

UNREAL EDITOR AND TOOLS, *UNREAL DEVELOPMENT NETWORK.* Disponível em: < http://udn.epicgames.com/Three/EditorAndToolsHome.html > Acesso em: 15 de novembro de 2011

VISUAL STUDIO 2010 PROFESSIONAL, *MICROSOFT VISUAL STUDIO*. Disponível em: < http://www.microsoft.com/visualstudio/pt-br/products/2010-editions/professional > Acesso em: 23 de outubro de 2011

NFRINGE, *PIXELMINE GAMES.* Disponível em: < http://www.microsoft.com/visualstudio/pt-br/products/2010-editions/professional > Acesso em: 23 de outubro de 2011