ESTUDO COMPARATIVO ENTRE AS GAME ENGINES UNITY E OGRE

COMPARATIVE STUDY BETWEEN THE GAME ENGINES UNITY AND OGRE

Diego Passos Costa[1], Fabio Fonseca Barbosa Gomes[2], Ricardo Duarte[3]

RESUMO
Atualmente, existe o mercado de Game Engines, que vem se tornando cada vez mais popular, como resultado do surgimento de soluções gratuitas, além da existência de soluções pagas. Deste modo, as Game Engines com um nível maior de complexidade possuem preços acessíveis comparados aos de alguns anos atrás. Parte desse acontecimento se deve à concorrência entre as produtoras de games, pois elas observaram que é possível obter lucro ao facilitar a produção de jogos, sendo que uma área que pode se beneficiar com isso é a educacional. Tendo em vista tanto a facilidade técnica dada por elas, quanto os custos associados à sua aquisição, o objetivo principal deste artigo é realizar uma comparação entre as Game Engines Unity e OGRE.
PALAVRAS-CHAVE: Motores de Games. Edição de Games

ABSTRACT
Currently, there is the Game Engines market, has become increasingly popular, because of the emergence of free solutions, in addition to the existence of paid solutions. In this way, Game Engines with a higher level of complexity, have affordable prices compared a few years ago. Since the event has been due to competition between the producers of games. They noted that it is possible to make a profit by facilitating the production of games, and one area that may benefit is educational, both the technical ease given by Game Engines and the costs associated with its acquisition. The main objective of this article is to perform a comparison between the Game Engines Unity and OGRE.
KEYWORDS: *Game Engines. Games Edition*

[1] Graduado em Ciências da Computação com Ênfase em Análise de Sistemas pela Universidade Salvador (2001), graduação em Sistema de Informação pela Universidade Salvador (2009) e especialização em Engenharia de Software pela Universidade Salvador (2013). Atualmente é Professor de Banco de Dados da Faculdade Maurício de Nassau - Lauro de Freitas, Professor de Sistemas Computacionais da Faculdade Maurício de Nassau - Lauro de Freitas e Projeto de Banco de Dados da Faculdade Maurício de Nassau - Lauro de Freitas. Tem experiência na área de Ciência da Computação, com ênfase em Metodologia e Técnicas da Computação.
[2] Mestre em Sistemas e Computação pela Universidade Salvador (2016), especialização em Redes de Computadores e Telecomunicações pela Universidade Salvador (2010) e graduação em Bacharelado em Sistemas de Informação pelo Centro Universitário Estácio da Bahia (2007). Atualmente é professor mediador a distância no Instituto Federal da Bahia - Lauro de Freitas, professor mestre nas instituições de ensino superior Faculdade Regional da Bahia, Faculdade Uninassau - Lauro de Freitas e Faculdade Dom Pedro II.
[3] Universidade Salvador (UNIFACS).

1. Introdução

Hoje em dia existe no mercado uma grande quantidade de *Games Engines* (Motores de Jogos) acessíveis ao público. Isso começou na década de 1980, quando o processo de desenvolvimento de *games* ainda era bastante simples, pois não existiam ferramentas de apoio sendo que tais *games* eram codificados em um Ambiente de Desenvolvimento Integrado ou em um editor de texto.

Apesar do desenvolvimento do *game* ser considerado simples, ele necessitava de um grande conhecimento computacional por parte do desenvolvedor, assim como ainda ocorre nos dias atuais, pois ainda se desenvolve *games* simples, como, por exemplo, o jogo *Snake*, desenvolvido através de um editor de texto e uma linguagem de programação C# (Hossain e Rahman, 2011).

Desta forma, era natural a não existência de ferramentas que auxiliassem o desenvolvimento de *games,* pois, as pessoas não vislumbravam o potencial de um determinado acontecimento. Ao analisar os primeiros consoles, geralmente, os fabricantes distribuíam seus *games* em cartuchos com códigos-fontes definidos e imutáveis. Para o *game* funcionar, era preciso instalar fisicamente o cartucho em um console. Não era necessário que o jogador tivesse um grande conhecimento computacional ou de engenharia para fazer essa comunicação funcionar (Wolf, 2008).

Ainda assim, mesmo com todas essas restrições, existiam pessoas modificando *games*, pois já havia versões modificadas, como o caso do *game* Street Fighter II. Muitos fliperamas possuíam uma versão alterada neste *game, em que* o personagem Ken era capaz de lançar magias no ar, o que ele não fazia no *game* original. Esse recurso só foi inserido nos Street Fighter II original anos depois (Davison, 2014).

Ainda nesta época, havia a dúvida de como ensinar ou aprender através de *game*s, pois, o seu desenvolvimento possuía diversas restrições, o que dificultava a personalização de um determinado *game*. Isso começou a se tornar possível com a popularização do computador, que auxiliou no processo de aprendizado, principalmente, quando os *games* passaram a ser produzidos para tal plataforma. Como consequência, os fabricantes começaram a produzir um game disponibilizando ferramentas de criação de cenários, que eram conhecidas como *Level Editing*.

No presente artigo, o *Level Editing* será considerado um marco inicial, pois a partir do momento em que alguém começa a ter acesso para projetar um *Level*/Mapa (fase), também já começou a trabalhar em um projeto de entretenimento, buscando como trazer a diversão ao jogador. Ainda que essa pessoa não possua contato direto com a codificação ou alto conhecimento computacional, trata-se de uma abordagem *top-down*, a depender de sua vontade, ela um dia também poderá se tornar um programador ou um *designer* de *level*, visto que esse é mais um caminho para oportunidades.

Apesar de o participante estar fortemente agregado a determinado *game,* quando ele lida com um Editor de *Level*, já começa a entender o processo de construção de um cenário e como alguns elementos podem interagir com o jogador. Neste processo, ganham o usuário, que adquire conhecimento, e a empresa. Se o *Level* vier a ser interessante, existirá uma boa divulgação do produto, além desse produto continuar a atrair a atenção dos jogadores com mais entretenimento (World of Level Design, 2012).

Por conseguinte, ferramentas com alto grau de abstração começaram a surgir, facilitando o processo de desenvolvimento e, assim, as empresas observaram que poderiam dar ainda mais poder aos usuários. Companhias passaram a disponibilizar ferramentas que permitissem também a criação de *scripts*, com o intuito de modificar comportamentos de inimigos, personagem e cenários. Um grande exemplo disso é o *game* Quake C.

Muitos dos *games atuais* possuem as ferramentas necessárias para a possibilidade da criação de uma **MOD**ificação quase que completa, daí surgem os (Microsoft Office Developers) MODs. O caso do *game* Counter Strike, com o MOD do *game* Half-Life, da Valve é um forte exemplo disso. Mais uma vez, jogadores e fabricantes ganham com isso. Assim, a empresa pode criar uma história, um game ou um personagem, mas a extensão deste universo é muito maior com a participação do jogador ou pessoas ligadas ao lazer.

Porém ainda havia duas restrições claras e naturais: as modificações nem sempre permitiriam que um determinado gênero de *game* se transformasse em outro, como, por exemplo, o FPS (*First Person Shooter* – Atirador em Primeira Pessoa) se transformar em um *Puzzle;* e, ainda, que a modificação pudesse ser considerada completa, para que outras pessoas fizessem uso do MOD, seria necessário efetuar a compra do game associado à modificação. Os usuários já estavam chegando a um nível que conseguiam criar *games* apenas com as ferramentas de modificações. Apesar dessa evolução, a utilização dos *game engines* tem a vantagem de fornecer liberdade para o desenvolvedor.

2. No Início o Código, *Old School*.

Nesta seção, será abordado um trecho inicial dos primórdios das *Game Engines* em que, só existia o código e, através de um pequeno experimento com Duke Nuken 3D, serão ilustrados alguns dos passos para modificar o *game.*

É importante citar que, além dos Editores de *Levels*, algumas empresas passaram a liberar o código-fonte de seus *games* após algum tempo. O Duke Nuken 3D, da 3D Realms e o Quake, da ID Software, são alguns exemplos. O mesmo também ocorreu com *games* anteriores, como o caso de Wolfenstein 3D. Por conseguinte, neste trabalho será feita uma pequena abordagem do *game* Duke Nuken 3D, devido a um pequeno experimento feito sobre o seu código. O *site* da 3DRealms passou por modificações, mas mantiveram até o momento uma parte representando o legado da empresa, incluindo o *layout* original, assim como demos e códigos-fontes que foram utilizados neste trabalho. Após ter sido efetuado o *download* do Demo do Duke Nuken 3D, assim como os seus códigos-fonte, o arquivo de README aconselha que seja feito uso do compilador WATCOM versão 1.0.

Realizado esse trabalho, é só navegar e estudar o código-fonte do game através do WATCOM. Na figura 1, pode-se observar que o *game* não possui muitos arquivos fontes. Isso, porque a *Game Engine*, desenvolvida por Ken Silverman, já fazia algum papel de abstração, assim, a *Engine* era apenas código, conforme ilustrado na figura 1. Isso também é um marco muito importante, é o início de um grande processo de abstração para o desenvolvedor.

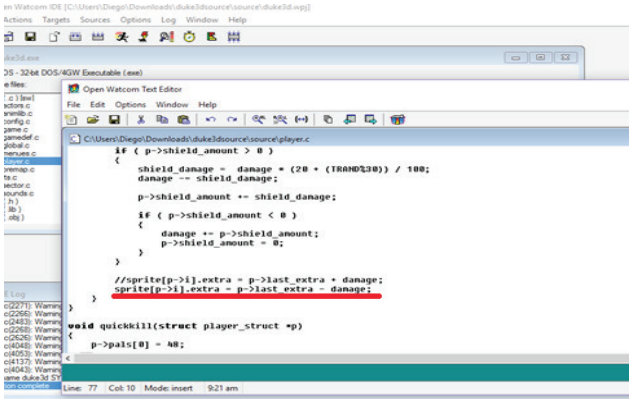

Figura 1. Modificação no *source* do game a fim do jogador ganhar *Life* ao levar dano.

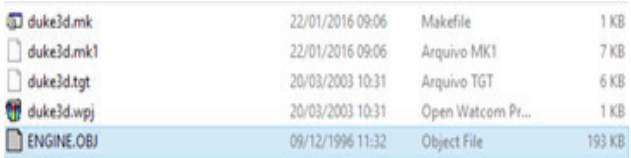

Figura 2. *Game Engine* de Duke Nuken 3D desenvolvida por Ken Silverman

Ainda na figura 1, é possível observar o código fonte responsável pelo Jogador, o arquivo player.c. Nele, foi feita uma pequena modificação com o objetivo de ilustrar como eram modificados os atributos de um personagem; o dano agora será revertido para mais vida assim que o jogador sofrer algum dano. Após a modificação, é feita a compilação. Depois, basta substituir o executável compilado pelo existente na pasta do Demo do Duke Nuken 3D.

Ao iniciar o *game*, basta que o personagem começar a receber dano pelo inimigo, que seu Health, localizado no canto inferior esquerdo, começará a subir em vez de descer e chegar a zero, conforme ilustrado na figura 3.

Figura 3. Resultado da modificação do código Duke ganha *Life* após sofrer danos.

Hoje, ao ter realizado um curso superior de Sistema de Informação, Ciências da Computação ou Engenharia, montar este ambiente tende a se tornar mais simples, devido aos conceitos adquiridos de compiladores, bibliotecas, *debug,* etc. Porém, os Editores de *Levels* surgiram para auxiliar na abordagem *top-down.* Desta maneira, os usuários também poderiam agora participar do processo de modificações dos games, através de um grau de abstração maior.

Outro *game* que utiliza bastante as modificações é o Goat Simulator. Este *game* é um simulador de cabras, em que o jogador tem a liberdade de movimentar a cabra em um cenário e ganha pontos por destruir objetos para completar certas missões. O jogo possui uma incrível gama de modificações que podem fazer o personagem principal do game ter poderes que inicialmente não tinha. Além disso, o cenário pode ser completamente alterado, fazendo com que o personagem principal possa sobrevoar o espaço, conforme pode ser visto na figura 4 (Carvalho, 2015).

Figura 4. Goat Simulator em um Cenário Espacial (Carvalho, 2015).

3. Editores de *Level*

Os Editores de *Levels* são responsáveis pela criação de cenários de jogos pré-existentes, no caso do game em questão, Duke 3D, existe um software chamado Mapster32. Eles estão diretamente ligados ao *layout* do *level*, também é responsável pela imersão do jogador no *game*. No início, era natural que os softwares de edição e mapas fossem manipuláveis, afinal, os *games* ainda funcionavam em MS-DOS. Isso necessitava do uso de comando pelo teclado, mas já era um grande avanço e não impediu a criação de inúmeros cenários. Neste ponto, o usuário já conseguia projetar um *level*, ou uma história paralela, com esta ferramenta, ainda que o modelo dos personagens fossem os mesmos. A figura 4 ilustra a construção de um *level* do mapa de Duke3D. Já a figura 5, demonstra o processo de criação dos mapas, que alternavam entre a visão 2D para a 3D.

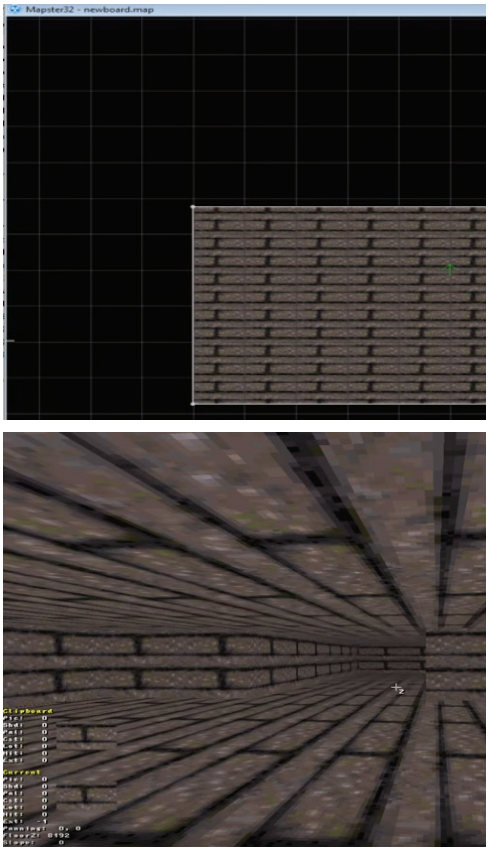

Figura 5 - Imagem do Mapster32 fonte: Mapster32 *Lesson* / Tutorial #1 : Duke Nukem 3D (Build Engine) *Level Editor*

Os limites começaram a serem quebrados, no final, mais poder foi dado ao jogador, agora em mãos de códigos fontes e editores de *levels,* os MODs chegaram.

4. MODification

Apesar de terem tido diversos outros títulos após Duke Nuken 3D, o título Quake do time de Jonh Carmak marcou uma época antes do Half-Life e MOD Counter Striker, que também será abordado. No final, o *multiplayer* foi um ponto focal do Quake, embora seus capítulos em *single player* serem bons e desafiadores, junto com os gráficos 100% 3D, novidade na época, ao contrário dos personagens de Duke 3D que ficavam sempre a olhar para o jogador mesmo depois de mortos, dessa maneira, não era possível pegá-los de surpresa.

Apesar da existência do *multiplayer* cooperativo, foi o *deathmatch* que realmente prevaleceu e se expandiu. A inteligência do computador é ilimitada, mas a capacidade do jogador inventar e inovar excede as expectativas. Outro ponto interessante é que a jogabilidade era extremamente comprometida na década de 1990, pois as conexões eram feitas através de modems de 14.400 kbps à 36.600 kbps, mas, mesmo com os atrasos de 5 a 7 até 15 segundos, a diversão na época era garantida. No início as batalhas ocorriam nos mapas convencionais do próprio game, mas com a liberação de ferramentas de edição, as modificações foram além.

No MOD Team Fortress de Quake, armas foram modificadas, assim como classes, os jogadores agora poderiam escolher personagem com características diferentes: um batedor que corre mais rápido, um engenheiro para cuidar da segurança, até *snipers*. Os jogadores continuavam a inventar, modificar e a aumentar o tempo de vida útil do game sem que a empresa responsável pela criação do game tivesse que fazer muita coisa, apenas disponibilizar as ferramentas (Id Software, 1996).

Neste momento, além das texturas, passaram a existir jogadores que modelavam objetos 3D e os inseriam no game. Na figura 6, é possível ver um jogador da classe *sniper segurando* uma arma que não existe originalmente no título Quake. Assim como o cenário em questão. Esse é mais um avanço que permitiu jogadores a se inserirem no mundo de desenvolvimento de games.

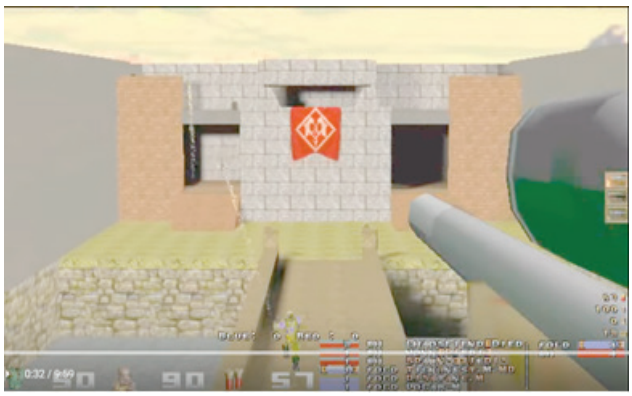

Figura 6. Arma modificada no game Quake 3D (ID SOFTWARE, 1996).

Somado às pessoas que poderiam agora modelar, o Quake C permitiu que pessoas que possuíam algum conhecimento de programação, mas não tão avançados, participassem do processo de modificações. Embora o Quake C seja baseado na sintaxe e estrutura da linguagem C, existe um grande diferencial na abstração que ele oferece.

Caso o usuário desejasse modificar o game diretamente na *Engine*, ele deveria possuir um alto conhecimento de C, além de implementações de baixo nível. Com o Quake C o usuário poderia apenas escrever pequenos trechos de códigos que sobrescreveriam os existentes durante a execução do *Game*.

Logo depois veio o Half-life, jogo de primeira pessoa, na história do *game*, um cientista consegue ativar um portal que traz alienígenas para sua dimensão, porém o Counter Striker é que se consagrou, uma MODficação feita por fãs.

5. *Game Engines* Atuais

Este capítulo tem como principal objetivo apresentar algumas das game *engines* atuais mais utilizadas pelo mercado. Essa tecnologia permite que seja possível construir toda a infraestrutura em um determinado jogo.

5.1 Unity

Segundo Passos (2009), a ferramenta Unit (visto na figura 7) é utilizada para desenvolvimento de aplicativos com principal enfoque em lazer, ou seja, jogos. Tal tipo de entretenimento é muito comum em toda a sociedade e uma ferramenta como essa vem como mais uma oportunidade de desenvolver *games* de forma mais fácil e eficiente.

Figura 7: Tela do Unity (UNITY, 2017)

O Unity é um software proprietário, criado pela empresa Unity Technologies e ele é dividido em duas versões principais, que são o Unity Pro e o Unity. O Unity Pro só pode ser utilizado ao se pagar uma licença em torno de U$ 1500,00 ou ao utilizar uma versão de testes por 30 dias e depois pagar a licença. Essa versão é a mais profissional das duas, pois é voltada para aplicativos mais comerciais, enquanto que o Unity é uma versão gratuita, voltada para fins educacionais (Unity, 2017).

Como pode ser visto na figura 8, o “Unity é um ecossistema de desenvolvimento de jogos: um potente mecanismo de renderização totalmente integrado com um conjunto completo de ferramentas intuitivas e fluxos de trabalho rápidos, para criar conteúdo interativo em 3D e 2D; fácil publicação em multiplataforma; milhares de recursos prontos de alta qualidade na Asset Store e uma comunidade que compartilha conhecimento. ” (Unity, 2017).

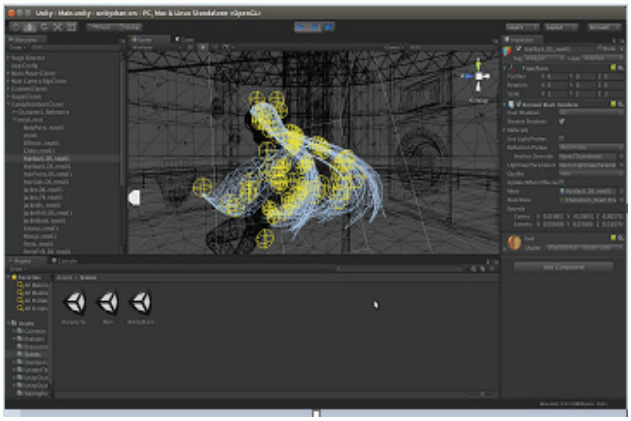

Figura 8: Tela do Unity (Unity, 2017)

O Unity é um programa que tem um preço acessível (para quem quiser a versão Pro) e uma série de aplicativos, que tornam mais fácil para o desenvolvedor a arte de criar jogos, pois ele não precisa ter conhecimentos profundos em DirectX ou OpenGL para poder criar um game. Essa ferramenta pode ter seus códigos feitos, inclusive, na ferramenta da Microsoft .NET (Kelner, 2017).

Esta ferramenta de realidade aumentada possui um motor de jogos 3D completo, inclusive para dispositivos móveis e tem suporte para diversas plataformas, além disso, o Unity possui uma série de vantagens, dentre elas: a extrema facilidade de uso (como pode ser visto na figura 9), a utilização de diversas linguagens utilizadas no mercado, além de ser utilizada para várias plataformas diferentes, tais como Windows e Mac OS.

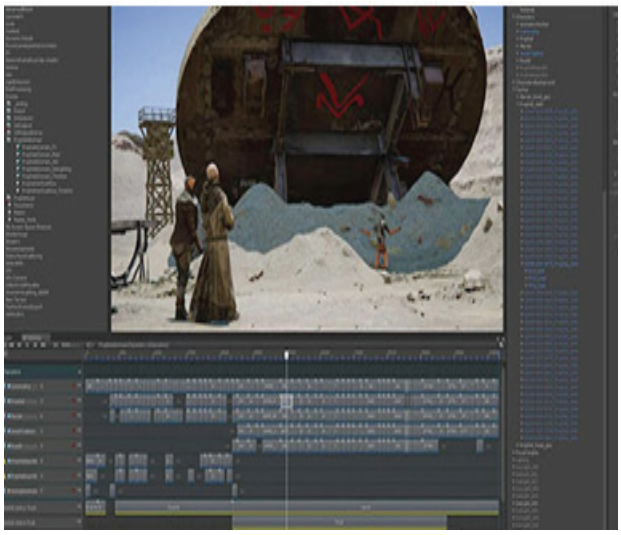

Figura 9: Facilidade de Uso (Unity, 2017)

O Unity possui algumas desvantagens, tais como a versão gratuita contar com muitas limitações, essa estratégia é uma forma de fazer com que o desenvolvedor precise comprar a versão Pro que é mais completa. Esse aplicativo possui suporte para as linguagens C#, JavaScript e Boo (Unity, 2017).

5.2 OGRE

O OGRE (*Object-Oriented Graphics Rendering Engine* ou Motor de Renderização Gráfica Orientada a Objetos) é uma ferramenta gráfica de código aberta e utilizada, inicialmente, como uma *engine* para jogos. Esse aplicativo, além disso, também é capaz de realizar diversos tipos de representações de cenários utilizando objetos em três dimensões, que são representados em arquivos denominados de *mashs*, como pode ser visto na figura 10.

Figura 10: Utilização de um *mesh* no OGRE (Kelner, 2017)

A figura 10 mostra o exemplo da utilização de um objeto no OGRE, esse objeto, que representa a deusa grega Athena, na verdade é um arquivo denominado de *mesh*, nesse arquivo estão distribuídos todos os códigos que formam o objeto a ser criado. O OGRE é uma ferramenta de código aberto que segue os princípios da GPL (GNU *Lesser Public Licence* ou Licença Pública da GNU), que define que o código fonte desta ferramenta pode ser utilizado gratuitamente, mas muitas organizações também a utilizam para fins comerciais. Essa ferramenta gráfica está disponível para os sistemas operacionais Windows, Mac OS e Linux, na figura 11 é mostrada a tela inicial do OGRE.

Figura 11: Tela Inicial do OGRE (Kelner, 2017)

Os movimentos dos objetos, bem como suas interações são baseados em uma linguagem de programação que, no caso do OGRE, pode ser em C++, Java ou Python. “A arquitetura adotada pelo OGRE utiliza vários padrões de projeto (*Factories, Singletons, Observers*, etc.), que a tornam mais simples e fácil de usar. Vários conceitos e abstrações são introduzidos na arquitetura, sendo os principais: *Root, SceneManager, RenderSystem, Entity, Mesh, SceneNode* e *Material*.” (Kelner, 2017).

O objeto *Root* marca o início do OGRE, pois ele deve ser instanciado primeiro já que é o responsável pela origem do sistema e da administração do acesso a esse, com isso tal objeto *Root,* também, deve ser a última instância a ser destruída quando o processo for encerrado na biblioteca.

O *SceneManager* reúnerá e organizará o conteúdo de uma determinada cena e como ela será construída pelo motor, esse objeto é responsável pela aplicação dos códigos-fontes e pela otimização

da exibição da cena, administrando as câmeras, objetos móveis, luzes, malhas e as partes estáticas de uma determinada cena.

O *RenderSystem* define a interface entre o OGRE e a API (*Application Programing Internface* ou Interface de Programação de Aplicativos) gráfica, sendo responsável pela execução dos comandos de renderização e configuração das opções da API gráfica. Essa classe é considerada abstrata, pois possui comandos que são tratados por esta API de renderização.

A entidade é uma instância de um objeto móvel na cena, enquanto que o *SceneNode* é usado para agrupar todos esses objetos, desde entidades, luzes, até câmeras em um único ambiente. Ele também é o responsável pelo armazenamento de informações sobre localização e tamanho desses objetos numa cena, já o objeto Material é o principal responsável pelo controle de como será feita a renderização dos objetos em uma determinada cena e os *meshs* são os arquivos que estão localizadas as classes dos objetos.

Além da facilidade gerada ao usuário para desenvolver seus ambientes e objetos, a interação entre os métodos desses objetos com outros dos cenários montados fará com que a simulação passe a ser bem utilizada, de forma que pareça com um ambiente real (Kelner, 2017).

No OGRE, para adicionar um objeto a um cenário, é necessário criar uma entidade. Ao carregar a entidade, é preciso definir um nome para ela e, também, definir qual é o arquivo de *mesh* que ele será ligado, este arquivo de *mesh* dará o formato ao objeto no cenário, formando objetos, conforme demonstrado na figura 12:

Figura 12: Objetos criados no OGRE

Antes de adicionar o objeto ao cenário, é necessário criar um nó. Esse nó pode ser criado como um objeto filho do cenário, além de ser definida a localização deste elemento. Com esses dois elementos criados, agora deve-se realizar o atrelamento da entidade com o nó para ele ser adicionado ao cenário.

5.3 Análise das *Game Engines*

As duas ferramentas de *game engines* têm algumas semelhanças, porém possuem grandes diferenças. O Unity tem aplicações mais bem trabalhadas, pois é voltada para o mercado comercial, principalmente em sua versão Pro.

O OGRE é uma aplicação completamente acadêmica, por não ter um grande apelo comercial, seu ambiente pode ser um pouco mais trabalhoso de ser montado em relação ao Unity, porém não existe uma versão paga do OGRE, o que pode torná-lo mais completo e eficiente se for comparado com o Unity gratuito.

Por conseguinte, foi verificado que o Unity é a ferramenta mais vantajosa do ponto de vista de produção, pois tem um formato mais profissional e comercial, enquanto que o OGRE é uma ferramenta mais voltada para o mundo acadêmico e educativo, com isso ela se torna também vantajosa para esse nicho do mercado.

Outro ponto que foi verificado nessa atividade, é que o Unity pode funcionar com mais facilidade em máquinas de baixo custo, enquanto que o OGRE precisa de uma boa configuração de um computador para que o software funcione de forma adequada. Apesar disso, o OGRE tem mais opções disponíveis para a utilização do desenvolvedor de jogos ou ambientes de simulação gráfica em comparação ao aplicativo Unity.

Considerações Finais

Neste trabalho, foi apresentado o histórico dos games, bem como as formas de modificações deles através dos MODs. Também se verificou que as *game engines* têm papel fundamental para o desenvolvimento de novos games, visto que a evolução da tecnologia da informação permitiu que um único desenvolvedor tivesse a capacidade de desenvolver um game que, no passado, necessitaria de muito investimento e de uma vasta quantidade de equipes de desenvolvedores e animadores.

Através deste trabalho, foi visto que as duas ferramentas são eficientes para a criação de jogos e simulações, elas atendem às demandas solicitadas pelo público e pelos desenvolvedores de aplicações de realidade aumentada. A sociedade está exigindo cada vez mais novos tipos de aplicativos e ela quer interagir com os objetos desses aplicativos como se fossem objetos do mundo real.

Para trabalhos futuros, pretende-se desenvolver um game nas diferentes *game engines* e realizar um efeito comparativo, com o objetivo de identificar qual dos motores de games consegue desenvolver

o jogo mais eficiente e com um menor custo de processamento.

REFERÊNCIAS

CARVALHO T. **Conheça melhores MODs para goat simulator, bizarro simulador de cabras.** Disponível em: http://www.techtudo.com.br/listas/noticia/2015/10/conheca-melhores-mods-para-goat-simulator-bizarro-simulador-de-cabras.html. Acesso em: dez. 2016.

DAVISON, P. **How hackers reinvented street fighter 2.** Disponível em: http://www.usgamer.net/articles/how-hackers-reinvented-street-fighter-2. Acesso em: dez. 2016.

HUDLICKA, E. Affective game engines: motivation and requeriments. In: **Proceedings of the 4th International Conference on Foundations of Digital Games.** Estados Unidos.

ID SOFTWARE. **Quake 3d** – Team Fortress. Disponível em: http://www.idsoftware.com/. Acesso em: abr. 2015.

KELNER. J. **O Engine Gráfico Ogre**. Disponível em: http://www.cin.ufpe.br/~sbgames/proceedings/tutorials/SBGames06TC05_Ogre.pdf. Acesso em: ?

MOORE, C. Hats of Affect: A Study of Affect, Achievements and Hats in Team Fortress 2. In **International Journal of Computer Game Research.** Suécia.

PASSOS. E. B. **Tutorial**: desenvolvimento de Jogos com Unity 3D. Rio de Janeiro, 2009. Disponível em: https://unisalesianogames.files.wordpress.com/2011/08/tutorialunity.pdf. Acesso em: ?

UNITY. Disponível em: https://unity3d.com/pt. Acesso em: dez. 2017.

WOLF, M. **The Video Game Explosion**: a history from PONG to playstation and beyond. Inglaterra: Greenwood Press, 2008. p. 15.

WORLD OF LEVEL DESIGN. What Level Editor and Game Engine Should You Use. Disponível em: http://www.worldofleveldesign.com/categories/level_design_tutorials/what-level-editor-game-engine-should-you-use-how-to-choose.php. Acesso em: dez. 2016.