UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

JONAS GOULART PEREIRA

AVALIAÇÃO DAS FUNCIONALIDADES DOS MOTORES DE JOGOS, UNITY3D E PANDA3D

CRICIÚMA

2015

JONAS GOULART PEREIRA

AVALIAÇÃO DAS FUNCIONALIDADES DOS MOTORES DE JOGOS, UNITY3D E PANDA3D

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. MSc. Luciano Antunes

CRICIÚMA 2015

JONAS GOULART PEREIRA

AVALIAÇÃO DAS FUNCIONALIDADES DOS MOTORES DE JOGOS, UNITY3D E PANDA3D

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Jogos Digitais

Criciúma, 23 de junho de 2015.

BANCA EXAMINADORA

Prof. MSc. Luciano Antunes - (UNESC) - Orientador

Prof. MSc. Paulo João Martins - (UNESC)

Prof. Esp. Willian Bertan da Silva - (UNESC)

AGRADECIMENTOS

A Universidade UNESC, pela oportunidade de fazer o curso de Ciência da Computação.

Ao professor Luciano Antunes, pela orientação, apoio e confiança.

Agradeço a minha mãe, heroína que me deu apoio, incentivo nas horas difíceis, de desânimo e cansaço.

Meus agradecimentos aos amigos, companheiros de trabalhos e irmãos na amizade que fizeram parte da minha formação e que vão continuar presentes em minha vida com certeza.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“Nunca se renda, exceto às convicções de honra e bom senso.”

Winston Churchill

RESUMO

Atualmente a área de desenvolvimento de jogos digitais vem crescendo muito, e um dos motivos é o rápido avanço em microprocessadores. Hoje em dia, qualquer dispositivo com um microprocessador pode executar um jogo digital, além de que os jogos não tem mais como alvo apenas jovens do sexo masculino. Assim desenvolver jogos digitais virou um grande negócio, e desenvolvedores independe também entram nessa esquema. Os jogos indie tem conquistado um grande público, tanto pelos seus gráfico simples ou quem sabe pela suas histórias envolventes, ou também muitas vezes por um estilo de jogo mais casual que é criado por muitos desse desenvolvedores. Porém, desenvolver um jogo demanda custos, esses custos são muitas vez arcados pelo próprio desenvolvedor independente. Diante disto, este trabalho fez uma comparação entre dois motores de jogos, ferramenta utilizada pelos desenvolvedores de jogos digitais, entre um com uma licença paga, Unity Pro, e uma *free,* Panda3D. Foram utilizado métricas demostrando funcionalidades existentes em cada motor, e para diferentes tipo de aplicações. Assim indicando que ambos possuem quase as mesma funcionalidade, excerto por algumas avançada onde apenas o Unity possui. O Unity se mostrou um motor mais prático para o desenvolvimento, com uma interface amigável, já o Panda3D possui a necessidade do desenvolvedor conhecer as linguagem utilizadas pelo motor. Além de que os dois possuem uma comunidade grande e ativa, com um grande acervo de documentos disponíveis para novos usuários.

Palavras-chave: Motor de jogo, Panda3D, Unity, Jogos digitais.

ABSTRACT

Nowadays, the gaming development field of work have been increasing thanks to the fast advance in microprocessor technology. Today, any technologic device with a microprocessor can run a digital game, also these games don't target only the young male audience. Thus, gaming development became a great deal, wich independent developers also entered in this kind of business. The indie games have constantly increased it's audience, both by it's simple graphics and gripping stories, or by it's casual play-style focused and created by many these developers. However, to develop a game demand high cost that is funded by the it's own independent developer. As a result, this academic material makes a comparison between two game development engines, wich are tools used by digital game developers, where one has a paid license, Unity Pro, and the other has a free license, Panda3D. Evaluation metrics were used to show actual resources avaliable for each engine, for differente kinds of applications. Thus, indicating that both have almost the same functionalities except by some avaliable only in Unity. The Unity has shown a more practical engine for development and friendly user interface; on the other side, Panda3D obliges the developer to have a certain knowledge about the programming languages used by this engine. Both engines have a big and active community, with a great collection of documentation avaliable to new users.

Keywords: Game Engine, Panda3D, Unity, Digital Games

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

BSP	Binary Space Partitioning
FMV	Video Full Motion
FSAA	Full-Screen Anti-Aliasing
GUI	Graphical User Interface
GUID	Globally Unique Identifier
HDA	High Dynamic Range
IGC	In-Game Cinematic
MMORPG	Massively Multiplayer On-line Role-Playing Games
NPC	Non Player Character
MVC	Model-View-Controller
PC	Player Character
PVS	Potentially Visible Set
RAP	Rapid Application Development
SCEA	Sony Computer Entertainment

SUMÁRIO

1 INTRODUÇÃO

O primeiro jogo digital foi inteiramente construído com hardware, mas avanços rápidos em microprocessadores mudaram tudo isso. Hoje em dia, jogos digitais são jogados em PCs versáteis e consoles de videogame especializados. Foram 50 anos desde daqueles primeiros jogos primitivos, mas a indústria ainda é considerada por muitos imatura. Ela pode ser jovem, mas quando você olha mais atentamente, você vai achar que as coisas têm vindo a desenvolver-se rapidamente. Os jogos digitais são hoje uma indústria multibilionária que cobre uma ampla gama de demografia.

A área de desenvolvimento de jogos digitais tem atualmente apresentando um grande crescimento no exterior, e agora mais recentemente no Brasil. O Brasil é quarto maior mercado de jogos digitais. Um dos grandes motivos para esse crescimento se dá que ultimamente os jogos digitais não são mais exclusividade para consumidores jovens do sexo masculino, mas também por crianças, mulheres e até idosos. Além que os jogos ultrapassam o entretenimento, adquirindo muitas vezes um caráter também utilitário.

O grande aumento de downloads de jogos casuais adicionou ainda mais complexidade ao mundo diversificado de jogos comerciais. Mesmo assim, grandes ainda são um negócio muito rentável. À medida que a indústria amadurece, sempre há a procura de melhores formas e mais eficientes para construir os produtos, e as equipes de desenvolvimento começaram compensar o aumento da complexidade, começando a utilizar ferramentas para o desenvolvimento como software reutilizáveis e middleware.

O Jogo digital, em sua base é um software, como qualquer outro, ele necessita de ferramentas para criação do mesmo. A ferramenta utilizada pelo desenvolvedor de jogos digitais é o motor de jogo, e como um middleware, ele vai conectar várias funcionalidades necessárias para o desenvolvimento do software. De modo tanto a agilizar, como felicitar para o desenvolvedor

Um motor de jogo é basicamente um software que fornece muitas das funções básicas que normalmente são usados para construir um jogo digital. A maioria dos motores de jogo permitim a renderização 2D ou formas 3D através da importação de arquivos gerados em um software de terceiros

como o Blender, ou 3D Studio Max ou Maya.

Motores de jogos 3D têm uma história relativamente curta na computação em geral. Sua evolução tem sido rápida. Começando por volta da década de 1990, o processador 486 finalmente fez o seu caminho para a população em geral como um processador rápido o suficiente para permitir que todos possam jogar. Ou, pelo menos, pseudo-3D, imagens 2D desenhados na forma de geometria tridimensional. A tecnologia para os verdadeiros polígonos 3D levaria mais alguns anos para torná-lo mais acessível aos desktops.

Enquanto motores de jogo variam muito nos detalhes de sua arquitetura e implementação. Praticamente todos os motores de jogo contêm um conjunto familiar de núcleo, conjunto de componentes principais, incluindo o motor de renderização, o motor de colisão e física, o sistema de animação, o sistema de áudio, o modelo de objeto do mundo do jogo, o sistema de inteligência artificial, e assim por diante.

Há uma grande variedade de motores de jogo disponível no mercado, para os mais diversos tipos de aplicações. Existe tanto motores pagos quanto *free*. Então esse trabalho aplicou métricas comparativas, assim de avaliar dois desses motores de jogo que disponíveis.

1.1 OBJETIVO GERAL

Avaliar as funcionalidades dos motores de jogos, Unity3D e Panda3d

1.2 OBJETIVOS ESPECIFICOS

Os objetivos desta pesquisa baseiam-se em:

a) Aplicar o conceito de jogos digitais;

b) Aplicar o conceito de motor de jogos;

c) compreender e utilizar os motores de jogos: Unity3D e Panda3D;

d) empregar métricas para a análise dos motores de jogos;

1.3 JUSTIFICATIVA

Atualmente a percepção é que o mercado brasileiro de jogos digitais desvaloriza as produções nacionais, pois tentam comparar as pequenas produções brasileiras com os jogos AAA (com orçamento de milhões de dólares), e as definem de baixa qualidade (PGT, 2014).

Mas o jogo independente tem crescido no mercado, tanto quem sabe pelos seus gráficos simples ou pelas suas histórias envolventes, jogos independentes são feitos com o próprio dinheiro e precisam de um tempo e de grande atenção para deixar um produto final recompensador (HONORATO, 2012).

Independentes da sua plataforma, em sua natureza um jogo digital é software interativo. Para desenvolver qualquer tipo de software, normalmente o desenvolvedor utiliza alguma ferramenta que facilite o seu trabalho. Nesse caso não é diferente. A ferramenta usada no auxílio ao desenvolvimento de jogos é o motor de jogo (TORI; KIRNER; SISCOUTTO, 2006).

Os motores de jogo são uma peça-chave no desenvolvimento, pois permitem adicionar recursos sofisticados aos jogos, tornando-os mais interessantes, com um maior realismo, melhor jogabilidade, e por decorrência com um maior apelo comercial e competividade junto ao mercado (LEMES, 2009).

Com tantos estilos diferentes de jogos em uma ampla gama de plataformas, não pode haver qualquer solução de software ideal único. No entanto, existem certos padrões de desenvolvimento, e há um vasto cardápio de soluções potenciais. O problema hoje é escolher a solução certa para atender às necessidades do projeto em particular. Indo mais fundo, uma equipe de desenvolvimento deve considerar todos os diferentes aspectos de um projeto e como eles se encaixam. É raro encontrar qualquer software que se adapte perfeitamente a todos os aspectos de um novo jogo (TORI; KIRNER; SISCOUTO, 2006).

Há uma grande gama de motores de jogos no mercado para escolher, mas ao mesmo tempo em que a arquitetura e implementação de motores tendem a variar muito nos detalhes, normalmente os motores de jogo

tendem a possuir um conjunto de componentes principais muito familiares entre si, que seriam, motores de renderização, colisão e física, animação, áudio, inteligência artificial entre outros (GREGORY,2009).

Este trabalho propõem fazer uma análise comparativa entre dois motores, um gratuito e outro pago, para ver se existe a necessidade do desenvolvedor independente, mover fundos para a escolha de uma ferramenta. Panda3D é um motor Open Source completo, com uma estrutura para renderização em 3D e desenvolvimento de jogos em Python e C++, e o motor pago Unity3D versão pro, possui uma interface muito amigável e simples, que tem por objetivo a facilitação do desenvolvimento de jogos digitais.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho está subdividido em seis capítulos. No capítulo 1 uma breve descrição sobre o trabalho desenvolvido, onde é possível tomar conhecimento do objetivo geral e dos objetivos específicos como também saber qual é a justificativa para a realização deste trabalho.

No segundo capítulo falo sobre jogo digitais, da essência de ser em princípio um jogo com regras, e também aborda a parte digital de jogos, além da estrutura e a engenharia de software.

O terceiro capítulo é voltado ao assunto de motor de jogo. É descreve nesse capítulo o conceito de motor de jogo, arquitetura sobre a visão de alguns autores sobre o assunto, componentes do motor: física, renderização, áudio, animação e jogabilidade, além de falar sobre os motores a serem comparados.

O quarto capítulo aborda as métricas para a comparação entre os motores apresentados no terceiro capítulo

No quinto é descrição dos trabalhos correlatos que possuem relacionamento com este trabalho.

Já no sexto capitulo fala sobre a metodologia, de como as métricas são aplicadas, e os resultados que foram obtidos da aplicação nos motores avaliados.

O último parte deste trabalho, mostra a conclusão de todo o desenvolvimento realizado, além de dificuldade e do conhecimento adquirido, Também falar sobre proposta de trabalhos futuros.

2 JOGOS DIGITAIS

Jogos digitais engloba um conceito mais extenso que apenas o videogame. O termo esteve associado apenas aos jogos voltado aos consoles e fliperamas. Mas as Tecnologias digitais são com base na microinformática, que envolve jogos para computadores, consoles, fliperamas, smartphones, *tablets* e qualquer outro equipamento que venha a ser criada. Desse modo esse termo se mostra mais abrangente em todo o contexto do formato de jogos, seja em vídeo ou outros (ARRUDA, 2014).

Jogo digital vêm em todas as formas e tamanhos, classificado em categorias ou gêneros cobrindo tudo, de Paciência à MMORPG (*Massively Multiplayer On-line Role-Playing Games*), e esses jogos são jogados em praticamente qualquer coisa com um microchip. Atualmente, você pode obter jogos para o seu PC, seu telefone celular, bem como uma série de diferentes jogos especializados em consoles, tanto de mão, quanto aqueles que se conectam a sua TV em casa. Estes consoles domésticos especializados tendem a representar a vanguarda da tecnologia de jogos, e o padrão destas plataformas sendo liberados em ciclos, que tem vindo a ser chamado de "gerações". As potências desta última geração são o Microsoft Xbox e o PlayStation da Sony (BITTENCOURT, 2012).

2.1 PRINCIPIOS DE JOGOS

Atualmente é comum ouvimos falar sobre, vídeo game, jogos digitais, jogos eletrônicos. É provável que você também já tenha jogado algum ou joga. Embora os jogos estejam de certa forma conectados com o lazer, a criação de um jogo é um processo árduo e de grande demanda de tempo, planejamento, organização e conhecimento. De certo modo é essencial conhecer e compreender todos os aspectos da criação de um jogo (BRANCO; MALFATTI; VINICIUS, 2013).

É de suma importância que o desenvolvedor de jogos conheça conceitos de jogos e jogos digitais. A uma grande variação de autores que comentam sobre esse assunto. Não seria viável citar todos, mas será mostrado

alguns mais importantes.

> Poderíamos considera-lo uma atividade livre, consciente tomada como “não seria” e exterior à vida habitual, mas ao mesmo tempo capaz de absorver o jogador de maneira intensa e total. É uma atividade desligada de todo e qualquer lucro, praticado dentro de limites espaciais e temporais, segundo uma certa ordem e certas regras. (HUIZINGA, 2007, p. 16).

Segundo Huizinga (2007) o jogo em si, seria qualquer tipo de atividade tomada conscientemente, fora da rotina normal, sem fins de lucro, em um local especifico, seguindo certas regras.

E segundo Caillos (1998) de uma forma formal, jogo seria como uma atividade livre, incerta, separada, regulamentado, improdutiva e fictícia. As formas psicológicas apontadas por Caillos são pensadas a partir das formas culturais que apresentam relativa autonomia em relação ao sistema, e aquelas que já foram incorporadas pela sociedade como valores institucionais. Como exemplos temos os esportes, loterias, cassinos, hipódromos, apostas em corrida, carnaval, teatro, cine, culto a artistas e etc.

Juul (2010), diz que os jogos podem ser separados em duas categorias dependendo como é apresentado o desafio ao jogador: *Emergence* (Emergente) e *Progression* (Progressivo). Os jogos emergentes, são apresentado como um pequeno número de regras simples, quando juntas, formam uma grande variedade de jogos. Os jogos que se condizem nessa categoria são os tradicionais como jogos de carta, tabuleiro e esportivos. Os progressivos são historicamente mais recentes e apresentam os objetivos em forma de uma sequência de ações para ser realizado pelo jogador.

Crawford (2003) evidencia quatro pontos fundamentais nos jogos: representação, interação, conflito e segurança. Uma das finalidades fundamentais dos jogos é educar. Os jogos tende a acrescentar novas experiências e conhecimento ao jogador. O autor também coloca que os

jogos promovem o desenvolvimento do ser, de modo a preparar para a vida adulta e independente.

2.2 CONCEITO DE JOGOS DIGITAIS

Jogos digitais são ações e decisões partidas de uma atividade consciente que resultará em uma condição final. Essas ações e decisões estão limitadas por um conjunto de regras e por um universo. O universo do jogo demonstra um ambiente de narrativa, conforme as ações e decisões do jogador, enquanto as regras restringem o que ele pode fazer (SCHUYTEMA, 2007, tradução nossa).

O jogo digital pode ser dividido em três partes: enredo, motor e interface interativa. O enredo demonstra temática do jogo e a trama, dando um objetivo e uma sequência aos acontecimentos. O motor do jogo é o controlador das reações do ambiente envolvendo as tomadas de ações e decisões efetuadas pelo jogador. A interface interativa é a conexão entre o jogador e o motor do jogo, onde demonstra repostas audiovisuais as decisões tomadas pelo jogador referente a mudanças do estado do ambiente (BATTAIOLA, 2000).

Em uma análise direta pode se constatar que os jogos digitais estão ligados aos computadores, ou de maneira mais abrangente PC’s, consoles e dispositivos moveis. Uma das características dos jogos digitais são a existência de mundos fictícios, que fazer uma certa distinção de jogos digitais de não digitais. Outra característica muito evidente nos jogos digitais seria à rigidez das regras. Apesar dos jogos, de modo geral, possuírem regras, quando se trata de jogos não-digitais, sempre há lugar para negociação das regras. Exemplo, pode se optar ou não por uma punição caso ocorra um caso especifico, a decisão deve ser realizada e respeitada pelos participantes que estão presentes. Já nos jogos digitais, não é comum essa tolerância, sendo essas regras são lidas através de algoritmos de computador, sendo seguida metodicamente. Há algumas exceções de jogos digitais, em que este personalizam algumas regras em alguns casos, mas tais modificações são triviais e não tão flexíveis como de jogo não-digital (JUUL, 2010, tradução

nossa).

2.3 ENGENHARIA DE SOFTWARE DE JOGOS DIGITAIS

Os desenvolvedores de jogos tendem a evitar a utilização de processos formais de engenharia de software, e tendem a partir direto para a produção sem uma decisão clara de ciclo de desenvolvimento. Antigamente isso funcionava, as equipes eram pequenas e todos sabiam de sua responsabilidade no desenvolvimento. Além dos jogos serem muito mais simples de serem criados. Hoje em dia as equipes de desenvolvimento estão cada vez maiores e jogos cada vez mais complexos e laçados para diversas plataformas e idiomas. (THORN, 2011, tradução nossa).

O processo pessoal de software, do inglês *Personal Software Process* (PSP) e o Scrum, que são metodologias rápidas, que vem sendo utilizadas por desenvolvedores de jogos nos últimos anos, esses dois processos de desenvolvimento têm sido aplicados com sucesso (CHANDLER, 2009).

2.2.1 Processo pessoal de software

Software Engineering Institute de Carnegie Mellon University, estabeleceu O Processo Pessoal de Software, ensina engenheiros otimizar seu próprio código. Dando destaque ao gerenciamento do código de modo a melhorar estimativas e o planejamento e redução de bugs no código. Um grande componente desse processo é a revisão de código em cada parte do desenvolvimento de software (REZENDE, 2005).

2.2.2. Scrum

É um conjunto de metodologias dedicadas a reparar produto pelo meio de iteração e feedback, voltada ao desenvolvimento ágil. Ele não encoraja a construir componentes isolados de um jogo, depois reunir e esperar que funcione.

O Scrum destaca a iteração na construção de um jogo desde o início, abrangendo uma funcionalidade muito básica e desenvolvendo essa no decorrer de iterações, até o jogo ser concluído (PHAM; PHAM, 2012).

A metodologia de foco no gerencial do Scrum, é flexível para poder ser usada por uma grande variedade de ambiente de desenvolvimento de jogos. Fácil implementação, por não requerem um treinamento convencional, apenas uma iteração da equipe em utilização do processo. Um enfoque básico do Scrum é a criação de subequipes auto direcionadas dentro da equipe maior do projeto. De modo a serem guiadas pelo “mestre de Scrum” que pode remover qualquer atraso no projeto (CHANDLER, 2009).

3 MOTOR DE JOGO

O desenvolvimento e entretenimento de jogos digitais mostra um grande crescimento no mundo e agora no Brasil. E a criação e uma parte importante, tanto quanto uso de ferramentas para possibilitar um desenvolvimento mais prático e de maior qualidade nos jogos. Os motores de jogo são uma dessas ferramentas Desenvolvimento Rápido de Aplicação do *Ingles, Rapid Application Development* (RAD), por que permite acrescentar muitos recursos complexos aos jogos digitais. Como realismo, jogabilidade, e consequentemente um maior mercado (HARRISON, 2013, tradução nossa).

Uma tarefa importante no desenvolvimento é escolha das ferramentas, conectada a definição do motor de jogo que é uma peça essencial para a criação do jogo. O motor de jogo permitirá ao desenvolvedor um grande reaproveitamento de código, e por assim um enorme ganho de tempo, ao utilizar-se das muitas funcionalidades, permitindo um desenvolvimento mais rápido de aplicações (WATT; POLICARPO, 2003, tradução nossa).

A criação de jogos, seja um jogo casual para web 2D ou jogo mais complexo em 3D, pode ser acelerado de um jeito drástico, através do uso do motor de desenvolvimento de jogo. Existem diferentes motores para diferentes tipos de jogos, podemos citar o GameMaker que seu foco seria os jogos desenvolvidos em 2D, o RPG Maker focado em jogos de RPG, e ferramentas para jogo 3D como o OGRED, Unreal Engine, Unity, Panda3D entre outros (BITTENCOURT, 2012).

3.1 CONCEITOS DE MOTOR DE JOGOS

Um motor de jogo, do inglês *game engine,* é basicamente uma biblioteca, ou um pacote de funcionalidades que são disponibilizadas para facilitar o desenvolvimento de um jogo e impedir que sua criação tenha que ser feita do zero. Também chamado de motor gráfico, o pacote é normalmente utilizado na modelagem de imagens 2D e 3D, além de trazer animações e sons padronizados. De modo a garantir um bom visual, ele é responsável por diversos itens da jogabilidade que são pouco percebidos pelos jogadores,

como o sistema de colisão entre personagem e objetos e a inteligência artificial de inimigos ou parceiros, essenciais na composição de um bom *game*. Mas tudo isso levou muito tempo (e muitos jogos) para chegar ao complexo estado em que se encontram os motores atuais (GREGORY, 2009, tradução nossa).

Sempre foi um desafio desenvolver jogos, mesmo nos primeiros consoles e computadores. O lançamento ocorria para apenas um videogame: transportá-lo para outra plataforma era uma tarefa corajosa, pois exigia a reprogramação de todos os aspectos do título, já que quase nada era aproveitado da versão anterior. A terceirização desses aspectos técnicos começou apenas como suporte gráfico, no formato de softwares de renderização criados por certas companhias, com o objetivo de processar as imagens criadas digitalmente (CHANDLER, 2009).

A primeira companhia a utilizar um motor foi a Freescape, da *Incentive Software*. Ela serviu jogos como Driller, em 1987, e Dark Side, lançado no ano seguinte. Já o termo só apareceu mesmo em meados da década de 1990, com a popularização definitiva dos jogos de tiro em primeira pessoa, do inglês *First Person Shooter*(FPS). Esse gênero estourou de vez com o lançamento do trio Quake, Doom e Wolfenstein 3D, todos na mesma década. A partir deles, vários sucessores usaram o mesmo motor de suas inspirações, mas alteravam armas, cenários, inimigos e mapas, para mostrar traços de originalidade (LENGYEL, 2011, tradução nossa).

Em muito pouco tempo, os motores gráficos tornaram-se populares em todos os tipos de jogos, como RPG e aventura. Já a capacidade das criações mais modernas transformou a programação de Motores em um verdadeiro trabalho de artista: são pacotes bastante completos que tentam deixar sons e imagens o mais próximo possível da realidade, exigindo um pouco menos do desenvolvedor. Além disso, ter uma base para criar jogos é uma garantia de benefícios para a empresa. Entre eles, podemos destacar o corte de custos na criação do que já está disponibilizado pela motor, velocidade no lançamento e a possibilidade de transportar o game para outras plataformas mais facilmente. Ainda há o suporte completo da empresa que criou tal motor, caso ele apresente algum problema. Se ela lançar atualizações no sistema,

quem possui a licença de uso poderá também receber as melhorias livremente (CHANDLER, 2009).

3.2 ARQUITETURAS DE MOTORES DE JOGOS DIGITAIS

Independente da criação de novos motores, tanto livre ou de uma solução proprietária, um motor de jogo deve conter uma arquitetura de software, que ligar uma série de elementos e proporcionam uma experiência em tempo real e interativa para o jogador (BITTENCOURT, 2012).

Técnicas de engenharia de software, como, *frameworks*, padrões de projeto e componentização, são importantes para arquiteturas de motor de jogo. Mesmo em grandes empresas é necessário ter o conhecimento da estrutura de software, para realizar as extensões utilizando os chamados *hot spots*. É importante nessa etapa o quanto de generalização pretende-se atribuir a arquitetura, e quanto maior o nível de abstração no desempenho da aplicação maior será o impacto. De modo criar um motor com a capacidade de gerar títulos de diferentes gêneros para várias plataformas diferentes, irá requerer uma arquitetura com muitas abstrações, assim generalizando as diferentes características dos diferentes gêneros e plataformas. De outro ponto de vista, criar um motor para uma especifica plataforma e para o único gênero, é possível explorar o máximo do hardware criado uma solução com menor abstração, e consequentemente com um alto desempenho. Deve ser considerado a portabilidade e a necessidade do determinado jogo digital requer ou não um alto desempenho. Um jogo casual não, normalmente, não irá requer um processamento de um jogo AAA (EBERLY, 2007, tradução nossa).

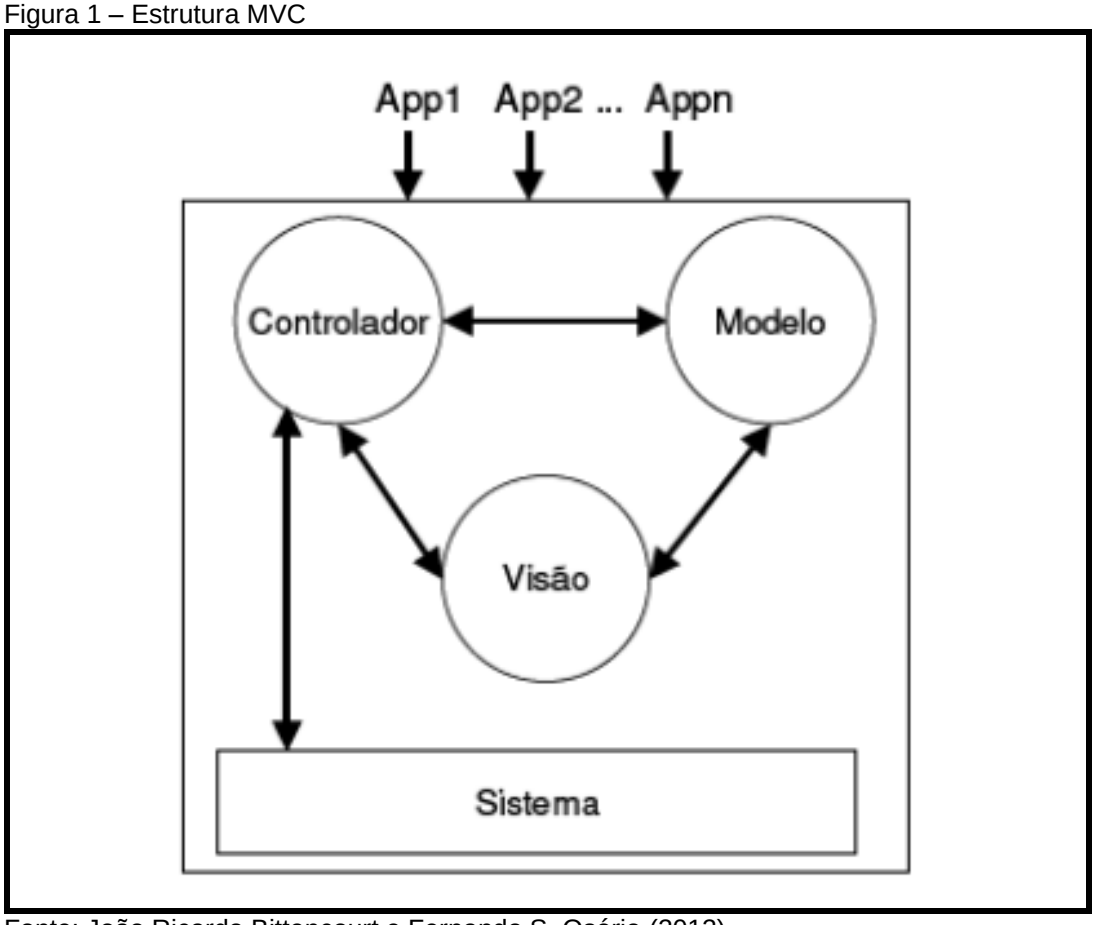
Existe inúmeras sugestões de arquiteturas, será abordado apenas alguns exemplos de arquiteturas de motores, para demonstrar a estrutura de software destas soluções computacionais. O modelo arquitetural Domingues foi baseado em Hodorowicz e Madeira para o motor Forge v8 3D. Baseado no padrão arquitetural *Model-View-Controller* (MVC). O foco de Domingues e criação de jogos voltados ao 3D.

A figura 1 demonstra o *framework* desenvolvido por Domingues, o modelo trata-se de características visuais dos objetos de uma cena que promove as funcionalidade e organização da mesma. As gerações de imagens

são de responsabilidade do módulo de visão. Tem classes responsáveis pelo volume de visão, projeção, câmera, área de visualização e renderizador. O controlador é incumbido pela dinâmica do sistema. Não é abordado por Domingues os controladores de rede e de Inteligência Artificial. O modulo apresenta classes de sinais de comunicação entre objetos do modelo, gerenciador de entrada, temporizador, gerenciador de animação e gerenciador de simulação de física (MADEIRA, 2001).

O hardware e o Software, são as duas abstrações da camada fornecidas pelo sistema. Os dispositivos físicos e o sistema operacional são comunicados pela a abstração de hardware, já a abstração de software providencia alguns artifícios fundamentais na criação de jogos, como as funções matemáticas e estruturas de dados (MADEIRA, 2001).

Figura 1 – Estrutura MVC



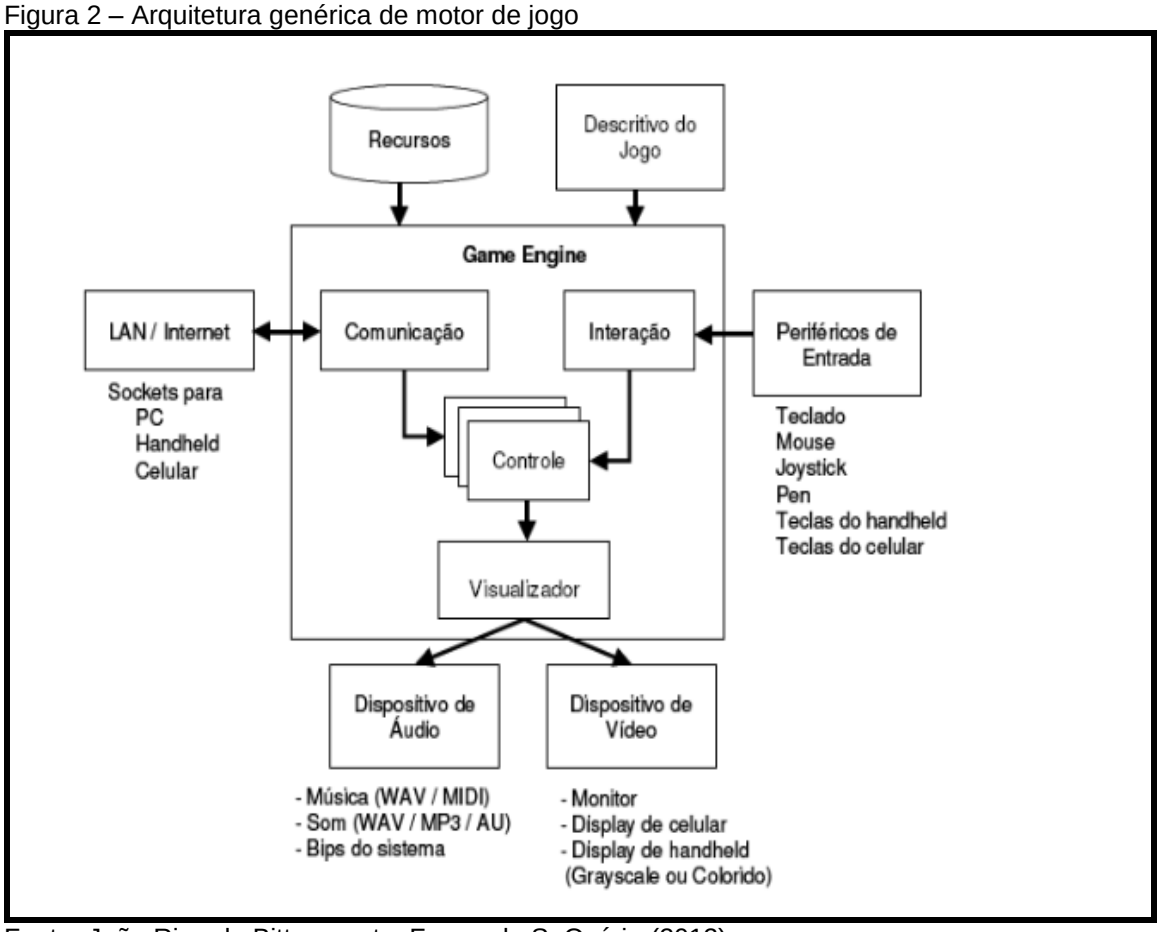
Fonte: João Ricardo Bittencourt e Fernando S. Osório (2012)

Já Bittencourt (2012) determina uma estrutura comum de um motor de jogo com destaque na modularização, como pode ser visto na figura 2. O

motor possui os seguintes componentes básicos: Comunicação, controle, interação e visualização. O componente de interação está associado com os periféricos de entrada, como joystick, teclado, mouse e etc. ele é responsável pela interação do usuário com esses periféricos, tratando os eventos gerados por estes.

Desenvolvimento de jogos de rede com múltiplos participantes, é feito atrávés do componente de comunicação. A grande maioria das aplicações utiliza processo de comunicação através de sockets e esses por sua vez são implementados de várias formas para cada tipo de plataforma de execução. Deste modo existe sockets para diferentes plataformas, computadores pessoais, smartphone e etc. sendo assim os responsáveis por fornecer as entradas para o motor de jogo são os componentes de comunicação e de interação. A lógica do jogo e manipulação dos objetos do jogo, são feitas pelo componente de controle. Para o componente cumprir tais funções, ele utiliza do jogo uma descrição, que utiliza das entradas fornecidas pelos componentes de iteração e/ou comunicação. O motor de jogo pode ter vários controladores, e cada um desses controladores é responsável por uma operação logica. Exemplo, controladores de inteligência artificial e simulações de física (BITTENCOURT, 2012).

Figura 2 – Arquitetura genérica de motor de jogo



Fonte: João Ricardo Bittencourt e Fernando S. Osório (2012)

Depois dos processos concluídos pelos componentes de controle, o visualizador cria a imagem de forma a representar o atual estado do jogo pela a perspectiva do jogador. O visualizador necessita da utilização de um dispositivo de vídeo, como monitor e telefones celulares para apresentação da imagem. Reproduções de sons também podem representar mudanças de estados nos objetos de cena do jogo, é interessante destacar a grande variedade de componentes de entrada de áudio e vídeo existente. Esta arquitetura tem uma proposta de ser genérica de forma que a integração destes componentes criar um motor independente de sua implementação. Assim o jogador pode interagir com jogo através de dispositivos como teclas, de teclados de computadores ou de um telefone (MILLINGTON,2010, tradução nossa).

Um motor de jogos dentro do conceito de engenharia de software trata da parte que executa certas funcionalidades do projeto de um programa. Na área de jogos, o motor de jogo lida com o hardware gráfico, renderização de modelos, entrada de dados fornecida pelo jogador, processamento de baixo

nível. Existe uma grande variedade de definição para um motor, contudo, estas definições tendem em algumas características:

a) permite o desenvolvimento de vários jogos diferentes utilizando-se do mesmo motor;
b) o reaproveitamento de códigos desenvolvido em antigos projetos;
c) a manipulação de APIs padronizadas, tal como OpenGL, mas em muitos casos, o desenvolvedor irá usar as próprias APIs dentro do ambiente do motor, para implementar funcionalidades específicas.

Enquanto os motores de jogo variam muito nos detalhes de sua arquitetura e implementação, os padrões de funcionalidades estão surgindo para motores de jogo licenciados ao público ou privados. Praticamente todos os motores de jogo contêm um conjunto familiar de componentes. Em suma, pode-se afirmar que um motor é composto por diversos “submotores”, sendo cada um responsável por tratar um tipo de problema envolvido em jogos. Os principais componentes são de interação, de visualização, de som, de animação, de física e de Inteligência Artificial. Nas próximas subseções cada um desses componentes será detalhado. (BITTENCOURT, 2012).

3.3 COLISÃO E FÍSICA

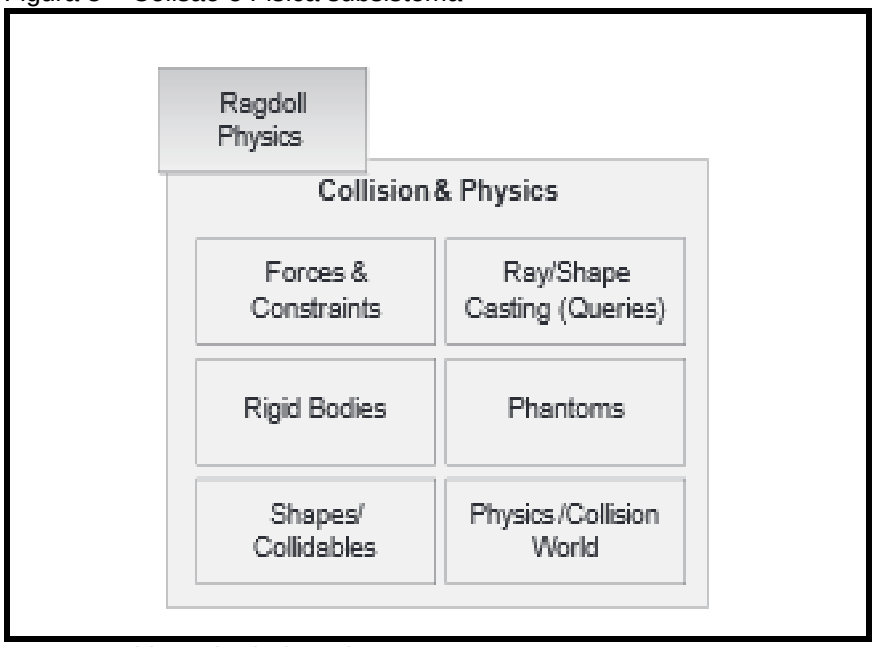
A detecção de colisão é importante para cada jogo. Sem ele, os objetos seriam penetráveis, e seria impossível interagir com o mundo virtual de qualquer forma razoável. Alguns jogos também incluem um realista ou semi-realista simulação de dinâmica. Chama-se isso de "sistema de física" na indústria do jogo, embora o termo dinâmico “corpo rígido” é realmente mais adequado, porque são geralmente apena o movimento (cinemática) de corpos rígidos e as forças e torques (dinâmica) que causam esse movimento. Esta camada está representado na figura 3 (GREGORY, 2009, tradução nossa).

Colisão e física são geralmente muito fortemente acoplados. Isto é porque
quando são detectadas colisões, eles são quase sempre resolvidos como parte do
integração física e lógica de satisfação de restrições. Hoje em dia, muito

poucos

empresas de jogos escrever o seu próprio motor de colisão / física. Em vez disso, terceirizam parte do SDK que é tipicamente integrado no motor (MILLINGTON, 2010, tradução nossa).

Figura 3 – Colisão e Fisica subsistema



Fonte: David H. Eberly (2007)

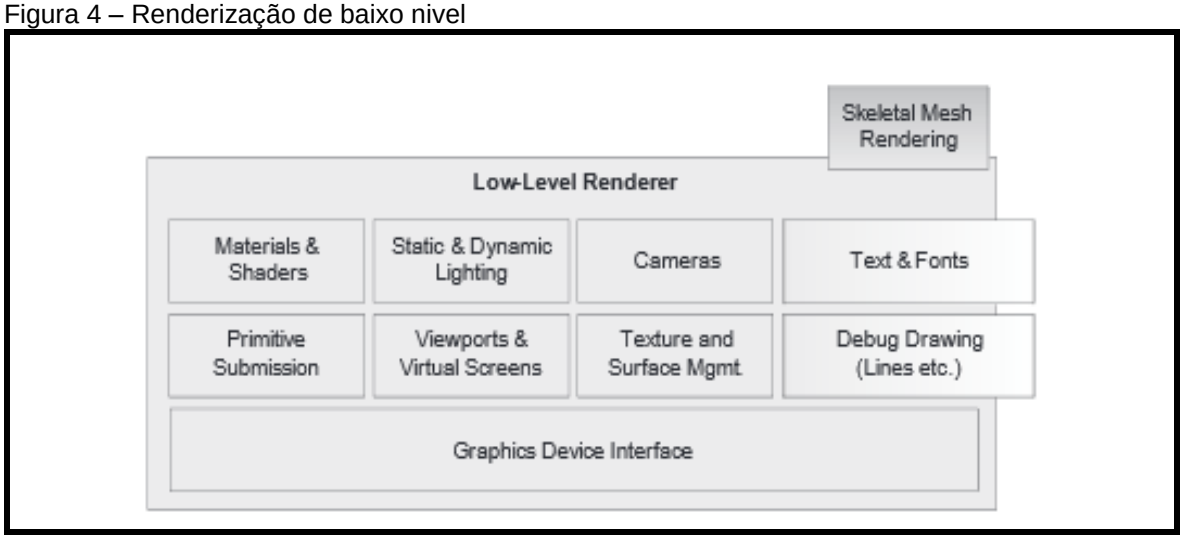
3.4 MOTOR DE RENDERIZAÇÃO

O motor de renderização é um dos maiores e mais complexos componentes de qualquer motor de jogo. Renderizações podem ser arquitetadas de muitas maneiras diferentes. Embora a maioria dos motores de renderização modernos compartilham algumas filosofias de design fundamentais, impulsionado em grande parte pelo projeto do hardware de gráficos 3D. Uma abordagem comum e eficaz para criar um projeto de um motor é empregar uma arquitetura em camadas da seguinte maneira (EBERLY, 2005, tradução nossa).

3.4.1 Processo de baixo nível

O processo de baixo nível, mostrado na Figura 4, abrange todas as instalações de processamento de texturas do motor. A este nível, o projeto está focado na prestação de um conjunto de primitivas geométricas como forma quanto possível, sem muita consideração para que partes de uma cena podem ser visíveis. Este componente é quebrado em vários subcomponentes, que são discutidas abaixo (GREGORY, 2014, tradução nossa).

Figura 4 – Renderização de baixo nivel



Fonte: David H. Eberly (2007)

3.4.1.1 Graphics device interface

SDKs gráficos, tais como DirectX e OpenGL, necessitam de uma quantidade razoável de código a ser escrito apenas para enumerar os dispositivos gráficos disponíveis, inicializa-los, e assim por diante. Isso normalmente é tratado por um componente que é chamado de interface gráfica de dispositivo (embora cada motor use sua própria terminologia) (EBERLY, 2007, tradução nossa).

3.4.1.2 Outros componentes de renderização

Os outros componentes do processador de baixo nível cooperam afim de recolher as submissões de primitivas, tais como malhas, listas de linha,

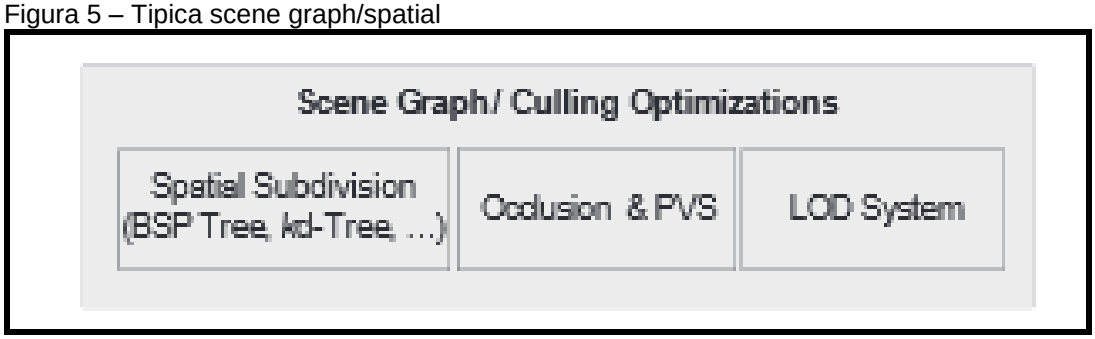
listas de ponto, partículas, texturas de terreno, cadeias de texto, e o que mais você quiser colocar, e cria-los tão rapidamente quanto possível (WATT; POLICARPO, 2003).

O processador de baixo nível normalmente fornece uma abstração de uma janela com uma câmera associada à matriz mundo e parâmetros de projeção 3D, como campo de visão diferenciando próximos e distantes. O processador de baixo nível também gerencia o estado do hardware de gráficos e *shaders* do jogo, através do seu sistema material e seu sistema de iluminação dinâmica. Cada primitiva apresentada está associada com um material e é afetado por “n” luzes dinâmicas. O material descreve a textura(s) utilizada pela primitiva, o dispositivo de configurações de estado precisa estar em vigor. As luzes determinam como dinâmica de cálculos da iluminação será aplicado ao primitivo (LENGYEL, 2011, tradução nossa).

3.4.1.3 Scene graph/culling optimizations

O processador de baixo nível desenha toda a geometria que lhe é submetida. Um componente de nível mais elevado é normalmente necessário, a fim de limitar o número de primitivas apresentados para o processamento, com base em alguma forma de determinação de visibilidade. Esta camada é mostrada na figura 5. (GREGORY, 2009, tradução nossa).

Figura 5 – Tipica scene graph/spatial



Fonte: David H. Eberly (2007)

Em pequenos mundos do jogo, a remoção de objetos que a câmera não pode "ver" é, provavelmente, tudo o que é necessário. Para mundos maiores, uma estrutura de dados de subdivisão espacial mais avançada

poderia ser usada para melhorar a eficiência de renderização, ao permitir que o conjunto potencialmente visível, do inglês *Potentially Visible Set* (PVS) de objetos a ser determinado muito rapidamente. Subdivisões espaciais podem ter muitas formas, incluindo uma árvore de particionamento de espaço binário, do inglês Binary space partitioning (BSP), uma quadtree, uma octree, uma árvore-kd, ou uma hierarquia de esfera. Essa subdivisão espacial é às vezes chamado de um grafo de cena (GREGORY, 2014, tradução nossa).

O ideal é que o processador de baixo nível deve ser completamente agnóstico com o tipo de subdivisão espacial ou de cena gráfica a ser utilizado. Isso permite que diferentes equipes de jogos possam reutilizar o código de submissão primitivo. O projeto do motor de renderização open source OGRE 3D é um grande exemplo desse princípio em ação. OGRE fornece um *plug-and-play* para grafo de cena. Os desenvolvedores de jogos pode selecionar a partir de uma série de desenhos de gráficos pré-implementado na cena, ou eles podem fornecer uma implementação de cenário gráfico personalizado (BUSCHMANN et al, 2009, tradução nossa).

3.4.1.4 Efeitos visuais

Motores de jogo modernos suportam uma ampla gama de efeitos visuais, como mostrado na Figura 6, incluindo (WATT; POLICARPO, 2003):

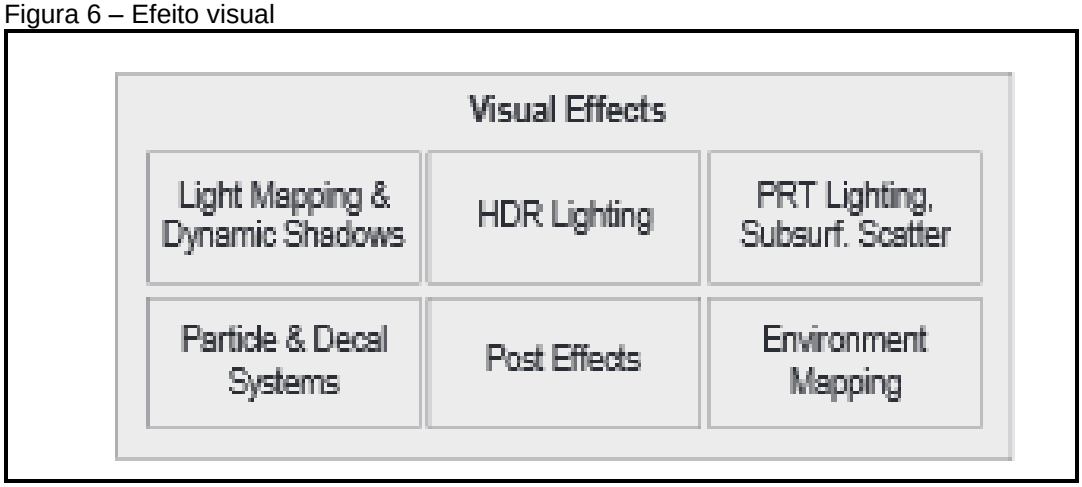
a) sistema de partículas (para a fumaça, o fogo, salpicos de água, etc.);
b) sistemas de decalque (por buracos de bala, impressões do pé, etc.);
c) mapeamento de luz e mapeamento de ambiente;
d) sombras dinâmicas.

Os efeitos em tela cheia, são aplicados após a cena 3D que foi processado para um buffer fora da tela, alguns exemplos de efeitos pós-tela cheia incluem:

a) *high Dynamic Range* (HDR) Iluminação e flor;
b) full-screen anti-aliasing (FSAA);

c) correção de cor e *cor-shift* efeitos, incluindo desvio de *lixívia*, saturação e saturação de efeitos, etc.

Figura 6 – Efeito visual



Fonte: David H. Eberly (2007)

É comum para um motor de jogo ter um componente do sistema que gerencia os efeitos de renderização especializados em partículas, decalques e outros efeitos visuais. Os sistemas de partículas e decalques são geralmente componentes distintos do motor de renderização e agem como entradas para o processador de baixo nível. Por outro lado, o mapeamento de luz, mapeamento de ambiente, e as sombras são normalmente tratadas internamente no motor de renderização adequada. Os efeitos em tela cheia ou são implementados como parte integrante do renderizador ou como um componente separado que opera em buffers de saída do processador (GREGORY, 2014, tradução nossa).

3.4.1.5 Front end

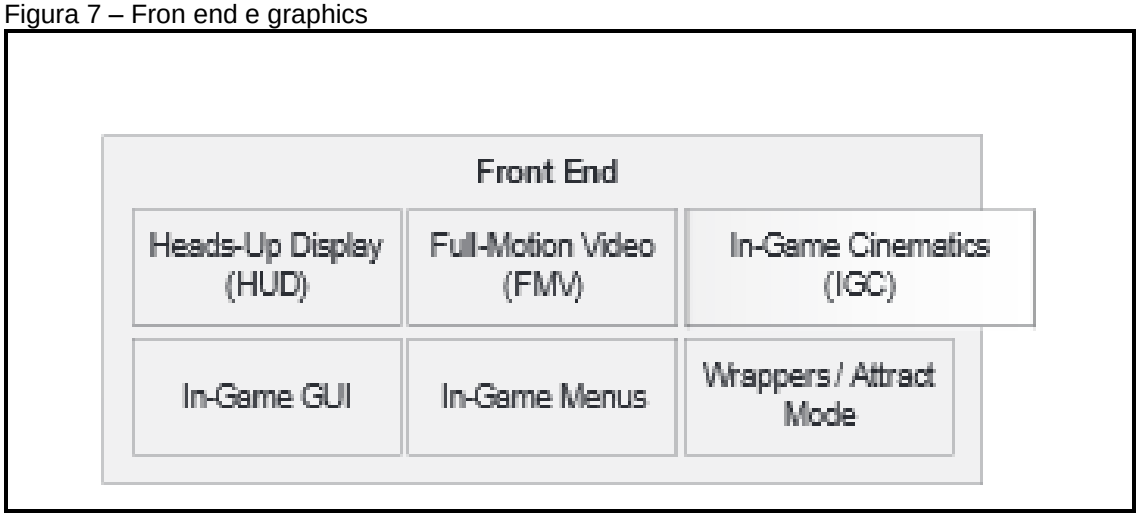
A maioria dos jogos empregam algum tipo de gráficos 2D sobrepostas na cena 3D para diversos fins. Estes incluem:

a) heads-up display do jogo (HUD);
b) no menu do jogo, um console, e / ou outras ferramentas de desenvolvimento, que podem ou não ser fornecidos com o produto final;
c) possibilidade de uma interface gráfica para o usuário dentro do jogo do inglês *Graphical User Interface* (GUI), permitindo que o

jogador manipule o inventário dele ou de seu personagem, configure as unidades para a batalha, ou executar outras tarefas complexas em jogo.

Esta camada é mostrada na figura 7. Gráficos bidimensionais como estes são geralmente implementadas pelo desenho quads texturizados (pares de triângulos) com uma projeção ortográfica. Ou eles podem ser renderizados em 3D, com os quads para que eles sempre estejam frente da câmera (MILLINGTON, 2010, tradução nossa).

Figura 7 – Fron end e graphics



Fonte: David H. Eberly (2007)

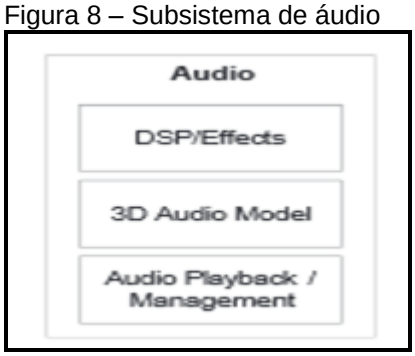
Também incluímos o sistema de full *motion video* (FMV) nesta camada. Este sistema é responsável pela reprodução de filmes em tela cheia que foram gravados antecipadamente (DÜVEL, 2004, tradução nossa).

Um sistema relacionado é o sistema de *cinematics* do jogo, do Inglês *In-Game Cinematics* (IGC). Este componente permite tipicamente sequências cinemáticas para ser coreografada dentro do jogo por conta própria em 3D. Por exemplo, um jogador caminha por uma cidade, uma conversa entre dois personagens principais pode ser implementado como um vídeo no jogo. IGC pode ou não incluir o jogador. Eles podem ser feitos como um corte deliberado durante o jogo, ao qual o jogador não tem nenhum controle, ou eles podem ser sutilmente integrados no jogo sem o jogador perceber que uma IGC está acontecendo (CREIGHTON, 2011, tradução nossa).

3.5 AUDIO

O áudio é tão importante quanto os gráficos em qualquer motor de jogo. Infelizmente, o áudio muitas vezes tem menos atenção do que a renderização, física, animação, IA e jogabilidade. Caso em questão: os programadores muitas vezes desenvolvem o seu código com os seus alto-falantes desligados. No entanto, nenhum grande jogo está completo sem um mecanismo de áudio impressionante. A camada áudio está representado na Figura 8 (GREGORY, 2009, tradução nossa).

Figura 8 – Subsistema de áudio



Fonte: David H. Eberly (2007)

Motores de áudio variam muito em sofisticação. As equipes do jogo normalmente costumam melhora-los com funcionalidade personalizada ou substituí-los com um motor existente. Para plataformas DirectX (PC e Xbox), a Microsoft oferece um excelente conjunto de ferramentas de áudio chamado XACT. A Electronic Arts desenvolveu um motor de áudio avançado, de alta potência chamado internamente OT SoundR!. Em conjunto com os estúdios first-party, Sony Computer Entertainment America (SCEA) fornece um poderoso motor de áudio 3D chamado Scream, que tem sido utilizado em uma série de títulos de PS3. No entanto, mesmo se uma equipe de um jogo utiliza um motor de áudio pré-existentes, cada jogo exige um grande investimento de desenvolvimento de software personalizado, trabalho de integração, de ajuste delicado, e atenção aos detalhes, a fim de produzir áudio de alta qualidade no produto final (HARRISON, 2003, tradução nossa).

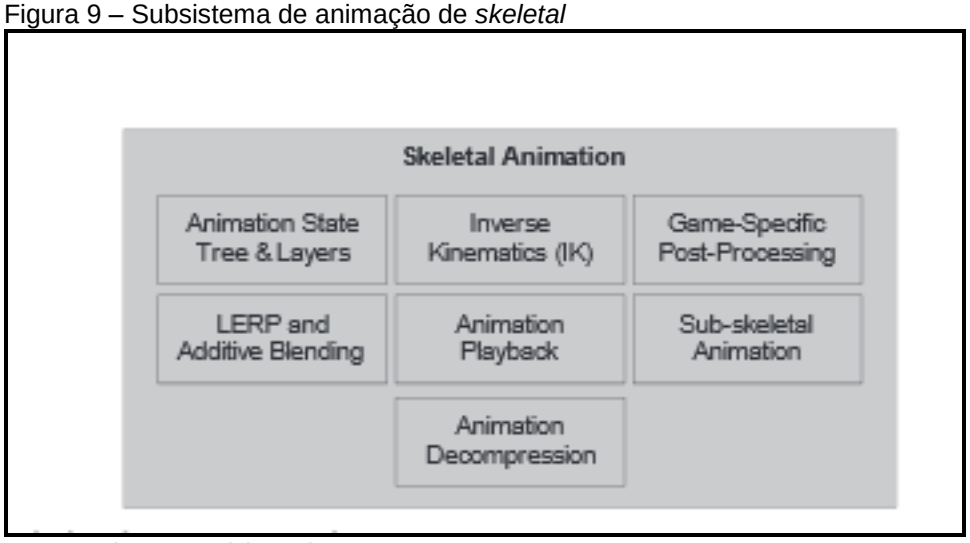
3.6 ANIMAÇÃO

Qualquer jogo que tem personagens orgânicos ou semi-orgânicos (humanos, animais, personagens de desenhos animados, ou até mesmo robôs) precisa de um sistema de animação. Existem cinco tipos básicos de animação usados em jogos:

a) sprite / animação textura;
b) rígida hierarquia animação corporal;
c) animação esquelética;
d) animação vértice;
e) *morph targets*.

Animação esquelética permite o movimento do personagem 3D, colocando um animador e utilizando um sistema relativamente simples de ossos. À medida que os ossos se movem, os vértices do movimento 3D movem-se com eles. Embora as *morph targets* e animação vértice são usados em alguns motores, animação esquelética é o método de animação que mais prevalece em jogos de hoje; Um sistema típico de animação esquelético é mostrado na Figura 9 (GREGORY, 2014, tradução nossa).

Figura 9 – Subsistema de animação de *skeletal*



Fonte: Eric Lengyel (2011)

O sistema de animação produz uma pose para todos os ossos do esqueleto, e depois essas poses são passadas para o mecanismo de

renderização como uma paleta de matrizes. O processador transforma cada vértice pela matriz ou matrizes na paleta, afim de gerar uma posição final do vértice. Este processo é conhecido como *Skinning* (EBERLY, 2005, tradução nossa).

Há também uma forte ligação entre a animação e sistemas de física, quando bonecas de pano (ragdoll) são empregados. A boneca de pano é um personagem animado (muitas vezes mortos) mole, cujo o movimento corporal é simulado pelo sistema de física. O sistema de física determina as posições e orientações das várias partes do corpo, tratando-as como um sistema restrito de corpos rígidos. O sistema calcula a animação da paleta de matrizes exigido pelo motor de renderização, a fim de desenhar o personagem na tela (MILLINGTON, 2010, tradução nossa).

3.8 SITEMAS FUNDAMENTAIS PARA JOGABILIDADE

O termo jogabilidade refere-se à ação que ocorre no jogo, as regras que governam o mundo virtual em que o jogo acontece, as habilidades do personagem do jogador e dos outros personagens e objetos no mundo, e as metas e objetivos do jogador. Jogabilidade é normalmente aplicada, quer na língua nativa na qual o resto do motor está escrito, ou em um alto nível linguagem de script ou, às vezes ambos. Para preencher a lacuna entre o código de jogabilidade e os sistemas de motores de baixo nível, a maioria dos motores de jogo introduzem uma camada que é chamada de fundamentos de jogo (a uma falta de padronização no nome dessa camada). Mostrado na figura 10, esta camada oferece um conjunto de instalações, em que a lógica específica do jogo pode ser implementado convenientemente (SCHUYTEMA, 2007, tradução nossa).

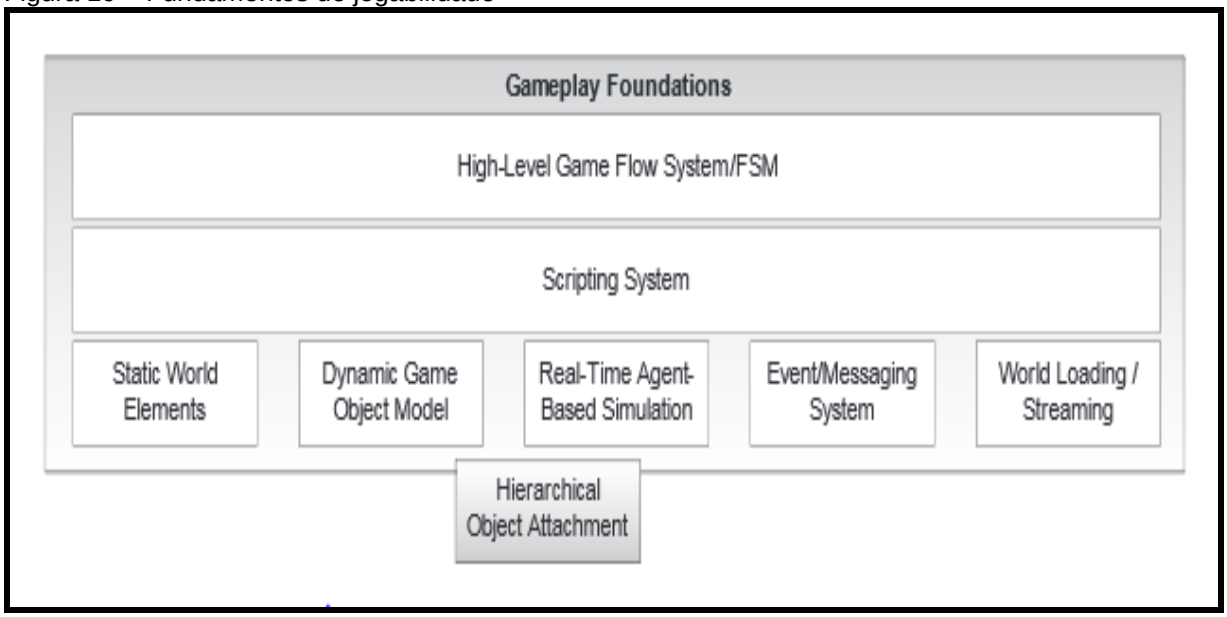
3.8.1 Mundos de jogo e modelos de objeto

A camada de fundações de jogo introduz a noção de um mundo artificial, contendo elementos estáticos e dinâmicos. O conteúdo dos mundos geralmente são modelados em uma maneira orientada a objetos (muitas vezes,

mas não sempre, usando uma linguagem de programação orientada a objetos). A coleção de tipos de objetos que compõem um jogo é chamado de modelo de objeto do jogo. O modelo de objetos do jogo fornece uma simulação em tempo real de um conjunto heterogéneo de objetos no mundo virtual do jogo. Os tipos mais comuns de objetos do jogo incluem (LENGYEL, 2011, tradução nossa):

a) geometria fundo estático, como edifícios, estradas, terrenos (muitas vezes um caso especial), etc .;
b) corpos rígidos dinâmicos, como pedras, latas de refrigerante, cadeiras, etc .;
c) personagens dos jogadores (PC);
d) personagens não-jogadores (NPCs);
e) armas;
f) projéteis;
g) veículos;
h) luzes (que podem estar presentes na cena dinâmica em tempo de execução, ou apenas usado para a iluminação estática).
i) cameras;
j) Etc.

Figura 10 – Fundamentos de jogabilidade



Fonte David H. Eberly (2007)

O modelo do mundo do jogo está intimamente ligado a um modelo de objeto de software, e este modelo pode acabar permeando todo o motor. O modelo de objeto de software; termo refere-se ao conjunto de recursos de linguagem, políticas e convenções usadas para implementar um software orientado a objetos. No contexto de motores de jogo, o modelo de objeto de software responde a perguntas, tais como (BUSCHMANN et al, 2009, tradução nossa):

a) é o seu motor de jogo desenvolvido de forma orientada a objetos?
b) que linguagem você vai usar? C? C ++? Java? OCaml?
c) como será a hierarquia de classe estática a ser organizado? Uma hierarquia monolítica gigante?
d) conjuntos de componentes são fracamente acoplados?
e) vai usar modelos e design baseado em políticas, ou o polimorfismo tradicional?
f) como são objetos referenciados? Ponteiros inteligentes? Manipula?
g) como os objetos serão identificados de forma exclusiva? Por endereço na memória só? Por nome? Por um identificador exclusivo global, do inglês Globally Unique IDentifier (GUID)?
h) como é a vida útil dos objetos do jogo?
i) como estaram os estados dos objetos do jogo simulado com o tempo?

3.8.2 Sistema de eventos

Objetos do jogo, invariavelmente, precisam se comunicar um com o outro. Isso pode ser feito de varios tipos de formas. Por exemplo, o objeto que envia uma mensagem pode simplesmente chamar uma função membro do objeto receptor. Uma arquitetura orientada a eventos é muito parecida com o que se poderia encontrar em uma interface gráfica do usuário típico, também é uma abordagem comum para a comunicação entre objetos. Em um sistema orientado a eventos, o remetente cria uma estrutura de dados pequena chamada de evento ou mensagem, contendo o tipo de mensagem e quaisquer

dados e argumentos que estão sendo enviada. O evento é passado para o objeto receptor chamando a função que manipula os eventos. Os eventos podem também ser armazenados em uma fila para serem lidados em algum momento futuro (EBERLY, 2007, tradução nossa).

3.8.2 Sistema de scripting

Muitos motores de jogo empregam uma linguagem de script, a fim de tornar o desenvolvimento de regras específicas de um jogo mais fácil e mais rápida. Sem uma linguagem de script você deve recompilar e vincular novamente o executável do jogo toda vez que é feita uma alteração à lógica ou a estruturas de dados utilizados no motor. Mas quando uma linguagem de script está integrada no seu motor, alterações à lógica do jogo e os dados podem ser feitas através da alteração e recarregamento do código de script. Alguns motores permitem que o script recarregue enquanto o jogo continua em execução. (EBERLY, 2005, tradução nossa).

3.8.4 Fundamentos para Inteligência Artificial

Tradicionalmente, a Inteligência Artificial (IA) é inserida em um software específico de um jogo, que normalmente é considerado parte do motor de jogo em si. Recentemente, no entanto, as empresas de jogos têm padrões que surgem em quase todos os sistemas IA conhecido, e estas bases estão lentamente começando a cair sob a alçada do motor adequado.

Uma empresa chamada Kynogon desenvolveu um motor IA comercial chamado Kynapse, que atua como uma "camada de fundação IA" sobre a qual logica de IA do jogo pode ser facilmente desenvolvida (MILLINGTON, 2010).

Kynapse fornece um conjunto de recursos, incluindo:

a) uma rede de nós de caminho ou volumes de *roaming*, que define as áreas ou caminhos onde os personagens IA são livres para se

movimentar sem medo de colidir com geometria estáticas no mundo;

b) informações sobre colisão simplificado em torno das bordas de cada área de livre circulação;
c) conhecimento das entradas e saídas de uma região, e de onde cada região um inimigo pode ser capaz de ver e / ou emboscada;
d) ganchos para o sistema de colisão, vestígios de linha de visada (LOS) e outras percepções;
e) um modelo de mundo que diz ao sistema AI onde todas as entidades de interesse (amigos, inimigos, obstáculos) estão, permite evitar dinâmica de objetos em movimento, e assim por diante.

Kynapse também fornece uma arquitetura para a camada de decisão AI, incluindo o conceito de cérebros (de um personagem), agentes (cada um dos quais é responsável por executar uma tarefa específica, como mover de um ponto a ponto, disparar contra um inimigo, buscar inimigos, etc.) e ações (responsáveis por permitir o personagem realizar um movimento fundamental, que muitas vezes resulta em criar animações esqueléticas no personagem) (DÜVEL, 2004, tradução nossa).

3.7 OPEN SOURCE

Motores de jogo 3D de código aberto são motores construídos por desenvolvedores de jogos amadores e profissionais e fornecidos online gratuitamente. O termo "open source" normalmente implica que o código fonte está disponível gratuitamente e que um modelo de desenvolvimento um pouco aberto é empregado, ou seja, quase qualquer um pode contribuir com o código. Licenciamento, se é que existe, geralmente é fornecido sob a licença GNU pública (GPL) ou licença GNU Lesser pública (LGPL). O antigo código de autorização para ser utilizado livremente por qualquer um, desde que o seu código também está livremente disponível; este último permite que o código a ser usado até mesmo em aplicações proprietárias com fins lucrativos. Lotes de outros regimes de licenciamento livres e semi-livres também estão disponíveis

para projetos de código aberto. Há um número impressionante de motores de código aberto disponíveis na web. Alguns outros motores de código aberto conhecidas estão listados aqui **(**RUSSELL; COHN, 2010, tradução nossa):

a) panda3D é um motor baseado em script. Interface principal do motor é a linguagem de script Python personalizado. Ele é projetado para fazer jogos de prototipagem 3D e mundos virtuais prático e rápido;
b) ake é relativamente novo motor de jogo totalmente caracterizado construído em cima do OGRE;
c) crystal Space é um motor de jogo com uma arquitetura modular extensível;
d) torque e Irrlicht também são bem conhecidos e amplamente utilizados motores.

3.9 PANDA3D

O motor de jogo Panda3D foi inicialmente um projeto fechado da Disney, mas mais tarde foi aberto à comunidade, permitindo que qualquer pessoa possa utilizar o mecanismo ou contribuir com o código. O Desenvolvimento do Panda3D agora é impulsionado e coordenado em um esforço conjunto da Entertainment Technology Center of the Carnegie Mellon University e a Disney Interactive. Juntos, eles estão adicionando novas funcionalidades, corrigindo bugs, e preparando novas versões do motor (RUSSELL; COHN, 2012, tradução nossa).

Panda3D é distribuído sob uma versão da licença BSD de código aberto muito liberal, que permite que qualquer pessoa interessada possa baixar, visualizar, alterar e redistribuir o código-fonte sem ter que pagar quaisquer taxas de licenciamento. Isso também pode se aplica a projetos comerciais. Assim, a criação de um jogo usando Panda3D e vendê-lo não é problema e nunca vai exigir qualquer montante de dinheiro a ser pago. Panda3D é um motor de jogo abundante em recursos necessário para a criação de jogos digitais modernos. Usando Python como uma linguagem de script para fazer a interface com as bibliotecas de programação de baixo nível, torna mais fácil

para criar rapidamente jogos, porque esta camada de abstração ordenadamente esconde muitas das complexidades de lidar com ativos, recursos de hardware, ou renderização de gráficos (MATHEWS, 2011, tradução nossa).

Panda3D é um motor de jogo completo. Isso significa que ele não é apenas uma coleção de bibliotecas de programação de jogos com uma interface agradável em Python, mas também inclui todas as ferramentas complementares para a pré-visualização, conversão e exportação de dados (RUSSELL; COHN, 2010, tradução nossa). É um motor de jogo, para renderização em 3D e desenvolvimento de jogos para programas em Python e C ++. Ele inclui gráficos, áudio, I / O, detecção de colisão, e outras habilidades relevantes para a criação de jogos 3D. Este motor é destinado a desenvolvedores independentes que estão interessados em criar os seus próprios jogos ou outras aplicações 3D para distribuição pessoal ou comercial com despesa mínima. Um entendimento básico de programação geral, como saber o que é uma variável, é necessário. Alguma familiaridade com programação orientada a objetos e com linguagem Python (LANG, 2011, tradução nossa).

Ele tem sido usado com sucesso por amadores, bem como grandes estúdios para criar jogos que variam de protótipos rápidos para MMOs comerciais de grande escala. Panda3D torna fácil ouso de modelos, texturas e sons para criar experiências interativas impressionantes. Ativando física e trabalhando com efeitos de sombreamento (MATHEWS, 2011, tradução nossa).

O Panda3D foi criado para o desenvolvimento de jogos comerciais, e seus usuários primários ainda são desenvolvedores de jogos comerciais. Por causa disso, a Panda precisa realçar quatro áreas: *poder*, *velocidade*, *integralidade* e *tolerância a erros*. Todos sabem o que poder e velocidade são. Mas integralidade e tolerância a erros merecem algum comentário extra (LANG, 2011, tradução nossa).

a) integralidade quer dizer que a Panda3D contém muitas ferramentas monótonas porém essenciais: navegação pela *scene*

graph, monitoramento de performance, otimizadores de animação e muito mais. Estas coisas podem não parecer atraentes, e como resultado, frequentemente as *engines open-source* não possuem tais ferramentas. Mas quando você está programando sério, e não apenas jogando, estas ferramentas precisam estar ai;

b) tolerância a erros é sobre o fato de que todos os desenvolvedores de jogos criam *bugs*. Quando você a criar, você vai querer que a sua *engine* mostre uma mensagem clara de erro e o ajude a encontrar o erro. Muitas *engines* vão parar de rodar instantaneamente se você passar o valor errado para uma função;

c) o Panda3D quase nunca para, e muito código é dedicado ao problema de identificar e isolar erros.

Finalmente, voltando ao poder e velocidade: o melhor jeito de estimar as capacidades da Panda3D é dar uma olhada nos programas de exemplo. Eles são pequenos programas que demonstram as capacidades da *engine*. O Panda3D foi desenvolvida pela Disney para o seu jogo online, Toontown (RUSSELL; COHN, 2012).

Apesar do motor em si ser completamente livre, ele vem com várias bibliotecas de terceiros que não são gratuitas. Algumas delas (como FMOD) até restringem você de a usar em jogos comerciais a não ser que você tenha uma cópia licenciada do FMOD. Devido a esta razão, Panda3D torna fácil desabilitar ou remover essas bibliotecas restritas, e a maior parte do tempo ele oferece uma alternativa. Por exemplo, ao invés de usar o FMOD, você pode usar OpenAL, que também está incluso no panda3D (LANG, 2011, tradução nossa).

3.9 UNITY

Unity é um motor de jogo ou uma ferramenta de criação de jogos que permite construir jogos digitais, que se destaca pelos games em 3D que rodam direto do navegador. Mas além de jogos web, a *engine* cria projetos para Windows, Mac, iOS e até mesmo para sistemas Android. De um modo geral, os games desenvolvidos com a Unity são leves e contam com uma boa qualidade gráfica (CREIGHTON, 2011, tradução nossa).

O motor de jogos Unity possui uma interfarce muito amigável e simples que objetifica a facilitação do desenvolvimento de jogos digitais e de diversos gêneros e outras plataformas. Sua interface de desenvolvimento é composta de uma gama variada de janelas chamadas de *views*, cada uma com um objetivo especifico. A figura 11 demonstra uma captura de uma janela (CREIGHTON, 2010, tradução nossa).

Figura 11 – Janela de intercafe unity

Fonte: Ryan Henson Creighton (2010)

Janela *scene* é a principal forma de manipulação dos elementos de visualização no editor de cenas da unity, que possibilita a orientação e posicionamento desses elementos com um feedback imediato dos efeitos e alterações efetuadas. Pode-se manipular graficamente objetos tais como câmera, cenário, personagens e todos os elementos em cena (SUVAK, 2014, tradução nossa).

Como todo motor gráfico, a Unity conta com uma série de bibliotecas que auxiliam na composição de games. Por exemplo, a ferramenta conta um

método *FPS*, basta arrastá-lo para a área do editor para que seu game incorpore toda a movimentação e a câmera de um jogo em primeira pessoa. Para adicionar as leis da física a um modelo 3D, você precisa apenas selecionar a opção *Rigid Body* como é mostrado na figura 12.

Figura 12 – Exemplo *Rigid Body*

Fonte: Alex Okita (2014)

Além de contar com formas básicas, a Unity permite que você importe modelos de outros editores como o Autodesk 3D Max. Para auxiliá-lo na criação de mapas, o motor gráfico oferece um editor de cenários com ferramentas para terrenos, águas e até florestas. O editor conta com o apoio de *scripts* que reconhecem tanto linguagem Java e C# (NORTON, 2013, tradução nossa).

O Unity possui duas versões principais: Unity pro, que custa r$ 75,00 mensais e a versão gratuita, simplesmente chamada Unity, que pode ser usada tanto para fins educacionais, quanto para fins comerciais, a versão pro pode ser testada por um período de 30 dias (OKITA, 2014, tradução nossa).

A licença mais simples é completamente gratuita e permite até que você distribua seus jogos livremente. Porém, alguns de seus recursos são limitados como a texturização avançada de imagens, efeitos de sombras e

água realista. Também não é possível distribuir seus jogos para formatos iOS e Android, mas apenas web e desktop. Seu projeto pode até ser comercializado com a versão *free*, mas ganha um ícone da engine (NORTON, 2013, tradução nossa).

4 METRICAS PARA MOTOR DE JOGOS

Para a comparação entre motores alguns elementos têm que ser destacados. Os principais fatores a serem avaliados são as suas funcionalidades, cinco categorias podem ser avaliadas: gráficas, acessórias, suporte, software e aplicações. As funcionalidades gráficas são entre as atividades do motor para a renderização do ambiente. Nesta categoria entram os elementos como a geração de terreno, *skybox*, efeitos especiais como neblina, fogo e *flare*, texturas suportadas, geração de sombras, iluminação, etc. Física; conexão em rede; inteligência artificial são funcionalidades relacionadas acessórias. Entre as funcionalidades as relacionadas a suporte correspondem a documentação, suporte e a atividade da comunidade de desenvolvedores e usuários (MILLINGTON, 2010, tradução nossa).

Outros fatores importantes de comparação são as linguagens suportadas e as plataformas de execução de um motor. A estrutura do motor é um elemento a ser considerado, que determina tanto a construção de um projeto ou aprendizagem. Essas funcionalidades compõem as métricas de software. Das métricas citadas, dependendo do tipo de aplicação algumas são mais ou menos importantes. Por exemplo, uma aplicação voltada ao simulador possui diferentes necessidades com relação a aplicação voltada ao entretenimento. Desse modo, dependendo do desenvolvimento da aplicação algumas métricas devem ser ressaltadas em relação a outras. As métricas de qualidade de software como portabilidade, eficiência e usabilidade estão relacionadas às métricas que serão apresentadas direta ou indiretamente, através das plataformas suportadas pelos motores (portabilidade), da documentação (usabilidade) e facilidade de criação dos elementos do ambiente (eficiência) (REBEIRO; SANTOS, 2009).

4.1 FORMATOS SUPORTADOS

Esse quesito será determinado pela quantidade de formatos utilizados pelo o motor: imagens, malhas e áudio.

4.2 API GRAFICA

A construção de um motor é feita sobre uma API gráfica, que é um grupo de funções para acessar os componentes gráficos do dispositivo. As duas principais APIs gráficas utilizadas são OpenGL e DirectX (SUVAK, 2014, tradução nossa).

4.3 STATUS

De acordo com o andar do desenvolvimento em que o motor se encontra atualmente.

4.4 CUSTO

Na escolha de um motor o custo em muitos casos é um fator decisivo. Que vai depender do recurso destinado ao desenvolvimento, que normalmente em jogos *indies* é pouco (PGT, 2014).

4.5 SISTEMA OPERACIONAL

Esse quesito trata da quantidade de sistemas operacionais ao qual o motor pode gerar aplicações, e a variedade de plataformas, que o motor dá suporte a distribuição do aplicativo. Este fato automaticamente abrange a influência na quantidade de usuários e consumidores que o produto pode atingir (OKITA, 2014, tradução nossa).

4.6 LINGUAGEM DE PROGRAMACAO

Este fato é muito importante para a escolha de um motor de jogo. A linguagem de programação muitas vezes determina o desempenho da aplicação.

4.7 ANÁLISE POR TIPO DE APLICAÇÃO

A análise por tipo de aplicação, se dá pelo fato que dependendo da aplicação, algumas métricas dever ser levadas mais em conta que outras, por que tente a ter mais importância no desenvolvimento do tipo especifico da mesma. Isso deve ser um importante fator na escolha de um motor (RIBEIRO; SANTOS, 2009).

4.7.1 Educacionais

O objetivo das aplicações educacionais é o auxílio do usuário no processo de aprendizagem. Recursos computacionais são muito necessário para este tipo de aplicação, afim de atingir o maior número de usuário possível. A sombra, terrenos, iluminação entre outros efeitos especiais, colocados por *shaders*, tente a não ser importante nesse tipo de aplicação. Além de normalmente não utilizar funcionalidades de comunicação em rede, inteligência artificial ou recursos de física, mesmo assim não signifique que eles não sejam empregados em jogos educacionais mais específicos (RIBEIRO; SANTOS, 2009).

4.7.2 Simuladores

Na computação, simulação é empregar padronizações em computadores, tais como expressões matemáticas ou especificações mais ou menos formalizadas, com o propósito de imitar um processo ou operação do mundo real. Desta forma, para ser realizada uma simulação, é necessário construir um modelo computacional que corresponda à situação real que se deseja simular (PASSOS, 2009).

4.7.3 Visualização

São tipos de aplicações de visualização de ambiente em 3D, ou reconstruções tridimensionais. Podem também ser utilizadas para a visualização e manipulação de informações ou dados com muitas dimensões. Os visualizadores são aplicações em que o principal foco está na forma como

será feita a visualização dos ambientes. Portanto, recursos como *shaders*, efeitos especiais, e inteligência artificial não são imprescindíveis para aplicações desse tipo (RIBEIRO; SANTOS, 2009).

5 TRABALHOS CORRELATOS

Reuso, extensibilidade e simulação de física em motores para jogos digitais, trabalho feito por Natal (2010). Este trabalho visa avaliar a simulação física, reuso e extensibilidade de alguns dos motores disponíveis atualmente (bullet, ode e physx), para isso foi feito um estudo detalhado destes itens de cada um dos motores, para então, poder realizar testes, comparações e avaliações entres estes motores. Com a análise dos resultados pôde-se notar que nenhum dos motores é melhor em todos os aspectos avaliados, porém, pode-se indicar *bullet* como a melhor alternativa de código livre, tendo em vista seus resultados e funcionalidades.

Realismo visual: renderização de imagens em jogos computacionais, trabalho feito por Lazzari (2011), neste trabalho apresentam-se os processos, as fases de renderização, assim como as técnicas utilizadas para obtenção do realismo visual, de maneira a explicitar o motivo de ele ser uma tendência na computação gráfica atual e provavelmente se tornar algo permanente. Expõem-se vários tipos de jogos computacionais, que são conceituados por vários autores e citam-se vários títulos de games conhecidos e famosos, que também colaboram para demonstrar o quanto os realismos nos jogos eletrônicos evoluíram. Apresenta-se as game engines, as *engines* gráficas, o que são e quais as principais, que são o Opengl e o DirectX, e outros *engines* derivados desses principais, todos são conceituados e são mostrados as características e diferenças de ambos, onde são aplicadas e para qual motivo foram desenvolvidos. Esses conceitos são vistos para obtenção de avaliação de técnicas de renderização de imagens em jogos computacionais, por meio da identificação e contextualização dos atributos que compõem o realismo, assim como estudos em ambientes de desenvolvimento 2D/3D e dos resultados adquiridos no protótipo gerado com a ferramenta blender, para essa avaliação.

Desenvolvimento de um motor de jogos 3D, utilizando WebGL, esse trabalho foi desenvolvido por Pereira (2012), este trabalho apresenta a implementação de um motor de jogos desenvolvimento em WebGL e também uma aplicação que o utiliza. o motor de jogos disponibiliza funcionalidades como grafo de cena, gerenciamento de objetos da cena, texturas e iluminação.

Para o desenho da cena e dos objetos é utilizada a linguagem javascript, utilizando elementos disponíveis na linguagem html5. O trabalho aborda as técnicas utilizadas para abstração da linguagem gsgl e da api do webgl e apresenta o resultado obtido com os testes de desempenho realizados através de um dos navegadores web suportados.

Mj3i-pa – um motor de jogos 3d para iOS com personagens articulados, trabalho feito por Schmitt (2012), esta monografia apresenta a construção de uma biblioteca gráfica que utiliza o conceito de personagens articulados para dispositivos móveis da plataforma iOS, juntamente com um aplicativo que utiliza a biblioteca para desenhar um braço mecânico articulado. A biblioteca disponibiliza ferramentas para a criação de personagens articulados simples. O desenho destes personagens foi realizado com a biblioteca opengl es 2.0 disponível no ios. A interação com o aplicativo ocorre através de toques na tela, sendo possível selecionar e rotacionar cada uma das articulações. Por fim foi demonstrado o desempenho prático com a aplicação rodando em um dispositivo da plataforma ios.5.

6 AVALIAÇÃO DAS FUNCIONALIDADES DOS MOTORES DE JOGOS, UNITY3D E PANDA3D

O mercado de jogos digitais tem crescido, tanto quem sabe pelos seus gráficos simples ou pela suas histórias envolventes, jogos independentes são feitos com o próprio dinheiro e precisam de um tempo e de grande atenção para deixar um produto bom, o motor de jogo é a ferramenta para o desenvolvedor de jogos digitais. Então foram aplicados métrica entre os Panda3D open-source e Unity pro, para levantar possíveis desvantagens e vantagens em utilizar um motor pago ou gratuito*/open source*.

6.1 METODOLOGIA

Existe uma variedade de motores de jogos disponíveis para diversos objetivos, como a criação de jogos, simulações, animações entre outros. Contudo, também existe uma grande necessidade de definição de mecanismos que permitam realizar comparações entre os motores, levando em consideração, por exemplo, parâmetros como qualidade de software, funcionalidades suportadas, facilidade de uso. Outro fator determinante que dever ser levado em consideração em uma eventual comparação de motores é a natureza da aplicação que se deseja desenvolver com o auxílio do motor.

Os motores de jogos foram comparados através de métricas, alguns fatores foram levados em consideração para ser realizada essa comparação entre esses motores, as funcionalidades são partes essenciais a serem consideradas, é comum ser avaliados em cinco diferentes fatores: acessórios, gráficas, software, suporte e aplicações. Correlação a funcionalidade gráfica compreendem-se os recursos de rederização do ambiente disponíveis no motor, ao suporte, no caso a atividade de comunicação entre desenvolvedor e usuários. Na funcionalidade de acessórios podemos destacar a simulação de física, que é de grande importância para o realismo e dinamismo no jogo; conexão de rede para partidas online através de redes entre computadores; inteligência artificial, que corresponde ao controle de NPCs. Correlação as métricas de qualidade de software podemos citar, portabilidade, eficiencia e

usabilidade.

Parâmetros de métrica adiciona uma pontuação variando de zero à três, de acordo com o grau de dificuldade tecnológica para a existência do recurso no motor e a facilidade que ela introduz na criação de uma aplicação 3D. A categorização da pontuação é descrita a seguir:

Zero, inexistente: recurso não presente no motor.

Um, básica: recurso essencial, que deve estar presente em praticamente todo motor.

Dois, recomendada: recurso intermediário mas que aumenta a produtividade da aplicação e conduz a um melhor Efeito final para aplicação; presente na maioria dos motores.

Três, avançada: recurso existente em poucos motores, normalmente associado a efeitos e características específicas, que diferenciam o motor dos demais; muito útil no desenvolvimento.

Por exemplo, a presença de algum tipo de gerente de cena recebe pontuação 1, enquanto que se o mesmo motor apresentar uma técnica de otimização de cenário, como octree ou paginação de terreno, receberia nota 2. O cálculo total da pontuação de um motor é o somatório de todos os pontos obtidos nas diversas categorias da métricas.

As métricas que são incorporadas em um motor na forma de complemento (add-ons) também entram na contagem dos pontos. A figura 13 contém todas as métricas e suas respectivas pontuações, localizada ao lado de cada métrica.

As métricas onde a pontuação difere da forma estabelecida no início desta seção são marcadas com um “*” na Tabela e serão discutidas.

Figura 13 – Métricas

Gerente de cena (G)		Iluminação (G)		Efeitos Especiais (G)		Texturização (G)		Animação (G)		Malhas (G)	
Geral	1	*Lightmaps*	1	*Billboarding*	1	Básica	1	*Keyframes*	2	Leitura	1
BSP	1	Por-Vértice	1	*Lens flare*	2	Multitextura	1	Por esqueletos	2	Deformação	3
Portais	2	Por-Pixel	2	Sist. partículas	2	*Bump mapping*	2	*Blending*	2	Progressivas	3
Octrees	2	Anisotropia	2	Prof. de campo	2	Formatos	*	*Inv. knematics*	3	Formatos	*
Pag. paisagem	3	Volumétrica	3	Névoa ou Céu	2			*Fwd. knematics*	3		
		Radiosidade	3	Água ou Fogo	3			*Morphing*	3		
		BRDF	3	*Motion blur*	3			Facial	3		
				Climáticos	3						
Shaders (G)		**Terreno (G)**		**Sombras (G)**		**API Gráfica (G)**		**Sist. Op. (Sf)**		**Linguagens (Sf)**	
Vertex shader	2	Básico	1	*Shadow map*	1	OpenGL	*	Tipo	*	C/C++,	
Pixel shader	2	Deformável	3	*Shadow volume*	3	DirectX	*	Quantidade	*	Java, etc.	*
Gerais (G)		**Rede (A)**		**Física (A)**		**Int. Artificial (A)**		**Som (A)**		**Suporte (Sp)**	
GUI	1	Estruturação	1	Básico	1	*Pathfinding*	2	Básico	1	Exemplos	1
Entrada/saída	2	Cliente/servidor	2	Detecção colisão	1	Tomada decisão	2	3D	2	Fórum	2
Scripting	2	Ponto a ponto	2	Veículo	3	*Script*	2	*Streaming*	2	*Chat online*	2
Possui algum		Segurança	3	*Ragdoll*	3	Formação	3	Formatos	*	Documentação	2
tipo de editor	3			Terreno	3						
Estado desenv.	*			Fluído	3						
Custo	*										

Fonte: Ítalo Mendes da S. Ribeiro (2009)

6.1.1 Gerente de cena

É uma estrutura de dados geral comumente usados por aplicativos de edição baseada em gráficos vetoriais e jogos de computador modernos:

a) particionamento binário de espaço (BSP) é um método para recursivamente subdividir um espaço em convexos definidos de hiperplanos. Esta subdivisão dá origem a uma representação de objetos dentro do espaço por meio de um estrutura de dados em árvore conhecido como árvore BSP.

b) octree é uma árvore, onde cada nó que não seja folha possui interligação com mais outros oito nós de uma estrutura de dados, esta interligação se faz normalmente por meio de ponteiros. A Octree é uma técnica de modelagem bastante comum no uso de tratamento de colisões.

5.1.2 Iluminação

A iluminação é um dos elementos importantes e influentes nos ambientes. Ele tem o poder de fazer ou quebrar o visual, tema e atmosfera. A iluminação é tão importante quanto a geometria. Iluminação tem a capacidade de dar vida a um grupo de objetos e levá-los para o próximo nível de qualidade.

Sua finalidade vai mais longe do que apenas dando aos jogadores a capacidade de ver onde eles estão indo. Cria atmosfera. Faz lugares assustador, aconchegante, quente e frio. Ele aumenta a sensação de objetos tridimensionais e ele cria composição e equilíbrio.

a) *lightmap* é um mapa bidimensional (textura) que representa a incidência de um conjunto de fontes luminosas em uma cena tridimensional. Este mapa é utilizado para acelerar a renderização de uma cena 3D, recurso muito utilizado em jogos de computadores.
b) iluminação por-pixel iluminação refere-se a qualquer técnica para a iluminação de uma imagem ou uma cena que calcula a iluminação para cada pixel em uma imagem renderizada. Isto está em contraste com outros métodos mais populares de iluminação, tais como iluminação vértice, que calcula iluminação em cada vértice de um modelo 3D e, em seguida, interpola os valores resultantes sobre as faces do modelo para calcular os valores finais da cor por pixel.
c) anisotrópico: é um método de melhorar a qualidade de imagem das texturas em superfícies que estejam à distância e num ângulo pronunciado em relação ao ponto de vista.
d) iluminação volumétrica é uma técnica usada em computação gráfica 3D para adicionar efeitos de iluminação para a cena renderizada. Ele permite ao espectador ver feixes de luz que brilham através do ambiente; vendo os raios de sol que fluem através de uma janela aberta é um exemplo de iluminação volumétrica, também conhecido os raios crepusculares.
e) métodos de radiosidade permitem computar as intensidades das energias radiantes chegando a uma superfície. Estas intensidades podem ser utilizadas para determinar as áreas de penumbra em uma superfície.
f) a função de distribuição de refletância bidirecional (BRDF) é uma função de quatro variáveis reais que define como a luz é refletida numa superfície opaca.

5.1.3 Efeitos especiais

Os efeitos especiais são usados para adicionar esse fator especial para o seu jogo. Quando usado corretamente, eles podem adicionar um impulso extra à jogabilidade.

a) em cenas tridimensionais, o termo *billboarding* é aplicado a uma técnica na qual os objetos são, por vezes, representados por imagens bidimensionais aplicadas a um único polígono, que é normalmente mantido perpendicular à linha de visão;

a) o efeito *Lens Flare* é uma técnica baseada em imagem que simula o espalhamento da luz ao visualizar objetos brilhantes devido a imperfeições em lentes de câmera;

b) sistema de partículas é uma técnica em computação gráfica que utiliza um grande número de muito pequenos *sprites* ou outros objetos gráficos para simular certos tipos de fenómenos "fuzzy", que são de outra maneira muito difícil de reproduzir com técnicas de renderização convencionais - geralmente altamente caótico sistemas, fenómenos naturais, ou processos causados por reações químicas;

c) profundidade de campo (DOF) Aplica-se um borrão para a cena com base na distância na frente ou atrás do ponto focal. Isso é para simular o que acontece em câmeras. O efeito pode ser usado para chamar a atenção telespectador e para fazer a renderização aparecem mais como uma fotografia ou como um filme;

d) *motion blur* embaça objetos com base em seu movimento. O sistema funciona por meio de um mapa de velocidades de tela cheia que é criado com uma resolução reduzida e objetos são borrados com base na sua posição no mapa.

5.1.4 Texturização

Textura (ou texturização) é a parte da computação gráfica, que se ocupa do estudo da simulação de materiais e texturas sobre planos. Transparência variável, de modo que você pode ter uma transição gradual entre o campo gramado e uma praia de areia, por exemplo.

a) *bump mapping* é uma técnica de computação gráfica onde pega-se cada pixel do objeto que está sendo renderizado e se aplica uma perturbação em sua superfície normal, baseada num mapa de altura previamente especificado, que como consequência varia a intensidade de luz "refletida" por este pixel. A iluminação é aplicada após os cálculos dando a cada pixel seu respectivo brilho. O resultado é uma superfície renderizada com mais detalhes e imperfeições lembrando o mundo real;
b) normal *map* ou Normal *mapping* é uma variante da técnica conhecida como bump mapping. É utilizada para simular o relevo em uma superfície, calculando o ângulo das sombras numa textura e, consequentemente, propiciando a impressão de maior de profundidade. É usada para dar um maior nível de detalhamento sem a necessidade de usar mais polígonos;

5.1.5 Animação

Animação por computador engloba uma variedade de técnicas. Técnicas de animação em 2D tendem a se concentrar em manipulação de imagem, enquanto técnicas 3D geralmente constroem mundos virtuais nos quais os personagens e os objetos se movem e interagem.

a) um *keyframe* na animação é um desenho que define os pontos inicial e final de qualquer transição suave. Os desenhos são chamados "frames" porque a sua posição no tempo é medida em quadros. Uma sequência de *keyframes* que define o movimento

que o espectador verá, ao passo que a posição dos *keyframes*, animação define o momento do movimento;

b) animação esquelética é a maneira padrão para animar personagens ou objetos mecânicos por um período prolongado de tempo (normalmente mais de 100 quadros). É comumente utilizada por artistas de jogos digitais e na indústria do cinema, e também pode ser aplicado a objetos mecânicos e qualquer outro objeto composto de elementos e articulações rígidas;

c) *blending* em gráficos de computador, o processo de combinar uma imagem com um fundo para criar a aparência de transparência parcial ou completa. Muitas vezes, é útil para processar elementos de imagem em passagens separadas, em seguida, combinar as várias imagens em 2D, resultando em uma única imagem;

d) *inv. knematics* é importante para programação de jogos e animação 3D, onde é usado para conectar fisicamente os personagens do jogo para o mundo, com pés firmemente que aterram em cima de um terreno;

e) *fwd. knematics* pode ser usado como um método em computação gráfica 3D para animar modelos;

f) *morphing* é um efeito especial usado em filmes e animações que muda uma imagem ou forma para outra através de uma transição suave.

g) animação facial é principalmente uma área de computação gráfica que encapsula os métodos e técnicas para gerar e animar imagens ou modelos de um rosto de personagem. O personagem pode ser um ser humano, um humanoide, um animal, uma criatura da fantasia ou personagem, etc.

5.1.6 Malhas

Uma malha de polígono é um conjunto de vértices, arestas e faces que define a forma de um objeto poliédrico em computação gráfica 3D e

modelagem sólida. As faces geralmente consistem de triângulos (malha de triângulos, quadriláteros), ou outros polígonos convexos simples, uma vez que isso simplifica o processamento, mas pode também ser composta de polígonos côncavos mais gerais, ou polígonos com furos.

5.1.7 Shader

No campo da computação gráfica, um *shader* é um programa de computador que é usado para fazer sombreamento: a produção de níveis adequados de cor dentro de uma imagem, ou, também para produzir efeitos especiais ou fazer vídeo de pós-processamento.

5.1.8 Terreno

Abrange uma variedade de métodos de que descreve mundo real ou mundo com superfícies imaginárias. A maioria prestação terreno comum é a representação da superfície da Terra. Ele é utilizado em várias aplicações para se obter um observador do quadro de referência. Ele também é usado muitas vezes em combinação com renderização de objetos e não terreno, tais como árvores, prédios, rios, etc.

5.1.9 Shadow

Sombreamento é o processo pelo qual as sombras são adicionadas a computação gráfica 3D.

a) *shadow mapping* são sombras criadas por meio de testes se um pixel é visível a partir da fonte de luz, comparando-o com uma imagem ou a profundidade de vista da fonte de luz, armazenada sob a forma de uma textura;
b) volume de sombra é uma técnica usada em computação gráfica 3D para adicionar sombras a uma cena renderizada. Para construir um volume de sombra, um raio projeto da fonte de luz através de cada vértice no objeto de projeção de sombra para algum ponto. Estas projeções, em conjunto, formar um volume;

qualquer ponto dentro desse volume é sombra, tudo fora é iluminado pela luz.

5.1.10 GUI

Interface gráfica do utilizador ou usuário, do inglês *Graphical User Interface* (GUI), é um tipo de interface do utilizador que permite a interação com dispositivos digitais através de elementos gráficos como ícones e outros indicadores visuais, em contraste a interface de linha de comando.

5.1.11 Física

A física do motor fornece um valor aproximado de simulação de certos sistemas físicos, tais como a dinâmica de corpo rígido (incluindo detecção de colisão), dinâmica corporal suave e dinâmica de fluidos, de utilização nos domínios da computação gráfica, jogos e filme. Seus principais usos são em simulações que estão em tempo real.

a) *ragdoll* ou em uma tradução literal boneca de pano na computação motores de física, física *ragdoll* é um tipo de animação processual que é frequentemente utilizado como um substituto para morte estática tradicionais em jogos e filmes de animação.

5.1.12 Inteligência artificial

Em jogos de vídeo, inteligência artificial é usada para gerar comportamentos inteligentes, principalmente em personagens não jogáveis, do inglês *non player character* (NPCs), muitas vezes simulando inteligência semelhante à humana. As técnicas utilizadas normalmente recorrem a métodos existentes a partir do campo da inteligência artificial (AI).

a) *pathfinding* ou *pathing* é a aplicação de uma rota mais curta entre dois pontos. É uma variante mais prático na resolução de labirintos.

b) Tomada de decisão é o processo cognitivo pelo qual se escolhe um plano de ação dentre vários outros (baseados em variados cenários, ambientes, análises e fatores) para uma situação-problema.

5.1.13 Formatos suportados

A quantidade de formatos suportado de: imagens, malhas e áudio, pelo motor vai definir a pontuação:

a) 1 à 5 formatos (1);
b) 6 à 10 formatos(2);
c) acima de 10 formatos(3).

5.1.14 API gráfica

As principais APIs gráficas utilizadas são OpenGL e DirectX. A pontuação é de acordo com as APIs suportadas pelo motor:

a) Somente OpenGL ou DirectX(1);
b) OpenGL e DirectX(2).

5.1.15 Status

A pontuação adicionada para a avaliação é definida pelo o estado de desenvolvimento, mostrado a seguir:

a) Inativa: o motor não está mais sendo mantido ou atualizado(1);
b) Beta: o motor ainda está em um estágio de testes(2);
c) Estável: o motor atingiu um estado estável, e pode ser utilizada para desenvolvimento sem problemas(3).

5.1.16 Custo

Nesse quesito motores gratuitos ou de baixos custos recebem as maiores pontuações. A pontuação atribuída ao motor nesta categoria depende da faixa de valores em que a sua licença se enquadra:

a) acima de US$10000 (1);
b) entre US$3000 e US$9999 (2);
c) entre US$1000 e US$2999 (3);
d) entre US$1 e US$999 (4);
e) gratuita (5).

Para um mesmo motor a licença pode variar de gratuitas com restrição de uso para algumas funcionalidades, até o preço completo, onde todas as suas funcionalidades estão disponíveis. Portanto, o processo de pontuação deve considerar apenas as funcionalidades presentes no motor de acordo com a licença escolhida.

5.1.17 Sistema operacional

Os tipos de SOs suportados foram organizados de acordo com os dispositivos suportados: *computadores; consoles*; e *dispositivos móveis.*
A pontuação do tipo de SO é atribuída da seguinte forma:

a) Suporta somente um dos tipos de sistemas operacionais (1);
b) Suporta dois tipos (2);
c) Suporta os três tipos. A pontuação relacionada a métrica de quantidade de SO é adicionada conforme o número de SOs suportados pelo motor (3).

5.1.18 Linguagem de programação

A pontuação aqui é definida por quanto mais linguagens um motor de jogos suportar.

5.1.19 Análise por tipo de aplicação

Cada métrica apontada nos tipos de aplicações deve ter 3 pontos acrescentados aos já estabelecidos anteriormente. Por exemplo, se 2 ponto é normalmente atribuído à métrica *octrees* e este critério é relevante para um certo tipo de aplica, a pontuação deve ser acrescida de mais 3 pontos, somando 5 para o quesito.

5.1.19.1 Educacionais

As aplicações educacionais objetivam auxiliar no processo de aprendizado dos seus usuários sobre um determinado assunto. Assim, as métricas que devem ser acentuadas para aplicações educacionais são:

a) sombras: *shadow volume*;
b) texturização: multitextura;
c) animação: *keyframes*, baseada em esqueletos, *blending*, facial;
d) efeitos especiais: *billboarding*, sistemas de partículas;
e) gerais: *scripting*, editor de partículas, interface gráficas com usuários (GUI);
f) som: som 3D.

5.1.19.2 Simuladores

São aplicações que contém muitos detalhes em sua composição, utilizam muitos recursos computacionais e possuem ótima qualidade gráficas. Muitas das funcionalidades de um motor são importantes no seu desenvolvimento.

a) gerenciamento de cena: particionamento binário do espaço (BSP), *octrees*, portais, paginação de paisagem;

b) iluminação: anisotropia, volumétrica, radiosidade,BRD;
c) sombras: voluma de sombras;
d) texturização: multitextura, *bump mapping*;
e) *shaders*: *vertex* e *pixel shaders*;
f) *malhas*: deformação;
g) efeitos especiais: *lens flare*, sistema de partículas, profundidade de campo, céu, água, fogo, névoa;
h) terreno: deformável;
i) gerais: GUI; editores de *script*, caminhos, terreno, partícula; *(10)* Som: som 3D e *streaming*;
j) física: detecção de colisão, veículo, *ragdoll*, fluídos, corpos rígidos; e
k) inteligência artificial: *pathfinding*

5.1.19.3 Visualização

Esse tipo de aplicação serve para a visualização de ambientes 3D, ou reconstruções tridimensionais a partir de imagens médicas bidimensionais. Assim métricas importantes são:

a) sombras: volume de sombras;
b) texturização: multitextura, *bump mapping*;
c) animação: *keyframes*, baseada em esqueletos;
d) efeitos especiais: *lens flare*, sistema de partículas, profundidade de campo, céu, água, fogo, névoa;
e) gerais: interface gráfica com usuário e suporte a vários formatos de imagem;
f) som: som 3D e *streaming*;
g) física: detecção de colisão, veículo, *ragdoll*, fluídos, corpos rígidos; e
h) rede: conexão cliente-servidor ou *multi-cast*.

6.2 RESULTADO OBTIDOS

As figuras 14, 15 e 16 monstram as fucionalidades e a comparação entre elas, destacados em amarelos são as fucionalidades que o Panda3D possuí, em azul o Unity e verde onde ambas possuem.

Figura 14 – Funcionalidades geral Panda3D

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Animção(G)	Malhas(G)	Shaders(G)	Terreno(G)	Rede(A)
Geral 1	Lightmaps 1	Billboarding 1(E)	Basica 1	Keyframes 2(E)(V)	Leitura 1	Vertex Shader 2(S)	Basico 1	Estruturação 1
BSP 1(S)	Por-Vértice 1	Lens Flare 2(S)(V)	Multitextura 1(E)(S)(V)	Por esqueletos 2(E)(V)	Deformação 3(S)	Pixel Shader 2(S)	Deformavel 3(S)	Cliente/servidor 2 (V)
Portais 2(S)	Por-Pixel 2	Sist. Particulas 2(E)(S)(V)	Bump mapping 2(S)(V)	Blending 2(E)	Progressivas 3			Ponto a ponto 2 (V)
Octrees 2(S)	Anisotropia 2(S)	Profud. De Campo 2(S)(V)	Formatos *(V)	Inv. Knematics 3	Formatos *	4	4	Segurança 3
Pag paisagem 3(S)	Volumétrica 3(S)	Nevoa ou Ceu 2(S)(V)	Formatos 1	Fwd. Knematics 3	Formatos 1			
	Radiosidade 3(S)	Agua ou Fogo 3(S)(V)		Morphin 3				
	BRDF 3(S)	Motion Blur 3		Facial 3(E)	8			3
6		Climáticos 3						
	15		5					
		15		9				

Sombras(G)	API Gráfica(G)	Sist. Op (Sf)	Linguagens (Sf)	Gerais(G)	Fisica (A)	Int. Artificial (A)	Som (A)	Suporte (Sp)
Shadow map 1	OpenGL *	Tipo *	C/C++	GUI 1(E)(S)(V)	Basico 1	Pathfinding 2(S)	Basico 1	Exemplos 1
Shadow volume 3 (E)(S)(V)	DirectX *	Quantidade *	Java, etc *	Entrada/saida 2(S)	Detecção Colisão 1(S)(V)	Tomada decisão 2	3D 2(E)(S)(V)	Fórum 2
				Scripting 2(E)(S)	Veiculo 3(S)(V)	Script 2	Streaming 2(S)(V)	Chat Onilne 2
	2	1	2	Possui algum tipo	Radgoll 3(S)(V)	Formação 3	Formatos *	Documentação 2
1				de editor 3(E)(S)	Terreno 3(S)(V)		2	
				est. De devem. *	Fluido 3(S)(V)	9		
				Custo *			7	5
					14			
				5				
				13				

Fonte: Jonas Goulart Pereira

Figura 15 – Funcionalidades geral Unity

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Animção(G)	Malhas(G)	Shaders(G)	Terreno(G)	Rede(A)
Geral 1	Lightmaps 1	Billboarding 1(E)	Basica 1	Keyframes 2(E)(V)	Leitura 1	Vertex Shader 2(S)	Basico 1	Estruturação 1
BSP 1(S)	Por-Vértice 1	Lens Flare 2(S)(V)	Multitextura 1(E)(S)(V)	Por esqueletos 2(E)(V)	Deformação 3(S)	Pixel Shader 2(S)	Deformavel 3(S)	Cliente/servidor 2 (V)
Portais 2(S)	Por-Pixel 2	Sist. Particulas 2(E)(S)(V)	Bump mapping 2(S)(V)	Blending 2(E)	Progressivas 3			Ponto a ponto 2 (V)
Octrees 2(S)	Anisotropia 2(S)	Profud. De Campo 2(S)(V)	Formatos *(V)	Inv. Knematics 3	Formatos *	4	4	Segurança 3
Pag paisagem 3(S)	Volumétrica 3(S)	Nevoa ou Ceu 2(S)(V)		Fwd. Knematics 3	Formatos 1			
	Radiosidade 3(S)	Agua ou Fogo 3(S)(V)	Formatos 2	Morphin 3				5
9	BRDF 3(S)	Motion Blur 3		Facial 3(E)	8			
		Climáticos 3	6					
	15			18				
		15						

Sombras(G)	API Gráfica(G)	Sist. Op (Sf)	Linguagens (Sf)	Gerais(G)	Fisica (A)	Int. Artificial (A)	Som (A)	Suporte (Sp)
Shadow map 1	OpenGL *	Tipo *	C/C++	GUI 1(E)(S)(V)	Basico 1	Pathfinding 2(S)	Basico 1	Exemplos 1
Shadow volume 3 (E)(S)(V)	DirectX *	Quantidade *	Java, etc *	Entrada/saida 2(S)	Detecção Colisão 1(S)(V)	Tomada decisão 2	3D 2(E)(S)(V)	Fórum 2
		3	1	Scripting 2(E)(S)	Veiculo 3(S)(V)	Script 2	Streaming 2(S)(V)	Chat Onilne 2
4	2			Possui algum tipo	Radgoll 3(S)(V)	Formação 3	Formatos *	Documentação 2
				de editor 3(E)(S)	Terreno 3(S)(V)		2	
				est. De devem. *	Fluido 3(S)(V)	9		7
				Custo *			7	
				3	14			
				11				

Fonte: Jonas Goulart Pereira

Figura 16 – Funcionalidades geral comparação

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Animção(G)	Malhas(G)	Shaders(G)	Terreno(G)	Rede(A)
Geral 1	Lightmaps 1	Billboarding 1(E)	Basica 1	Keyframes 2(E)(V)	Leitura 1	Vertex Shader 2(S)	Basico 1	Estruturação 1
BSP 1(S)	Por-Vértice 1	Lens Flare 2(S)(V)	Multitextura 1(E)(S)(V)	Por esqueletos 2(E)(V)	Deformação 3(S)	Pixel Shader 2(S)	Deformavel 3(S)	Cliente/servidor 2 (V)
Portais 2(S)	Por-Pixel 2	Sist. Particulas 2(E)(S)(V)	Bump mapping 2(S)(V)	Blending 2(E)	Progressivas 3			Ponto a ponto 2 (V)
Octrees 2(S)	Anisotropia 2(S)	Profud. De Campo 2(S)(V)	Formatos *(V)	Inv. Knematics 3	Formatos *	4	4	Segurança 3
Pag paisagem 3(S)	Volumétrica 3(S)	Nevoa ou Ceu 2(S)(V)	Formatos 1	Fwd. Knematics 3	Formatos 1			
	Radiosidade 3(S)	Agua ou Fogo 3(S)(V)	Formatos 2	Morphin 3				5
9	BRDF 3(S)	Motion Blur 3		Facial 3(E)	8			3
6		Climáticos 3	6					
	15		5	18				
		15		9				

Sombras(G)	API Gráfica(G)	Sist. Op (Sf)	Linguagens (Sf)	Gerais(G)	Fisica (A)	Int. Artificial (A)	Som (A)	Suporte (Sp)
Shadow map 1	OpenGL *	Tipo *	C/C++	GUI 1(E)(S)(V)	Basico 1	Pathfinding 2(S)	Basico 1	Exemplos 1
Shadow volume 3 (E)(S)(V)	DirectX *	Quantidade *	Java, etc *	Entrada/saida 2(S)	Detecção Colisão 1(S)(V)	Tomada decisão 2	3D 2(E)(S)(V)	Fórum 2
		3	1	Scripting 2(E)(S)	Veiculo 3(S)(V)	Script 2	Streaming 2(S)(V)	Chat Onilne 2
4	2	1	2	Possui algum tipo	Radgoll 3(S)(V)	Formação 3	Formatos *	Documentação 2
1				de editor 3(E)(S)	Terreno 3(S)(V)		2	
				est. De devem. *	Fluido 3(S)(V)	9		7
				Custo *			7	5
				3	14			
				5				
				11				
				13				

Fonte: Jonas Goulart Pereira

As figuras 17, 18 e 19 mostram as funcionalidades que se destacam no tipo de aplicação educacional.

Figura 17 - Funcionalidades tipo de aplicação educacional Panda3D

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Sombras(G)	Gerais(G)	Som (A)
Billboarding 1	Multitextura 1	Keyframes 2	Shadow volume 3	GUI 1	3D 2
Sist. Particulas 2		Por esqueletos 2		Scripting 2	
		Blending 2		Possui algum tipo	
		Facial 3		de editor 3	

Fonte: Jonas Goulart Pereira

Figura 18 - Funcionalidades tipo de aplicação educacional Unity

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Sombras(G)	Gerais(G)	Som (A)
Billboarding 1	Multitextura 1	Keyframes 2	Shadow volume 3	GUI 1	3D 2
Sist. Particulas 2		Por esqueletos 2		Scripting 2	
		Blending 2		Possui algum tipo	
		Facial 3		de editor 3	

Fonte: Jonas Goulart Pereira

Figura 19 - Funcionalidades tipo de aplicação educacional comparação

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Sombras(G)	Gerais(G)	Som (A)
Billboarding 1	Multitextura 1	Keyframes 2	Shadow volume 3	GUI 1	3D 2
Sist. Particulas 2		Por esqueletos 2		Scripting 2	
		Blending 2		Possui algum tipo	
		Facial 3		de editor 3	

Fonte: Jonas Goulart Pereira

As figuras 20, 21 e 22 mostra as funcionalidades que se destacam no tipo de aplicação simuladores.

Figura 20 - Funcionalidades tipo de aplicação simulação Panda3D

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Malhas(G)	Shaders(G)
BSP 1	Anisotropia 2	Lens Flare 2	Multitextura 1	Deformação 3	Vertex Shader 2
Portais 2	Volumétrica 3	Sist. Particulas 2	Bump mapping 2		Pixel Shader 2
Octrees 2	Radiosidade 3	Profud. De Campo 2			
Pag paisagem 3		Nevoa ou Ceu 2			
		Agua ou Fogo 3			
		Climáticos 3			
Terreno(G)	**Sombras(G)**	**Gerais(G)**	**Fisica (A)**	**Int. Artificial (A)**	**Som (A)**
Deformavel 3	Shadow volume 3	GUI 1	Detecção Colisão 1	Pathfinding 2	3D 2
		Entrada/saida 2	Veiculo 3		Streaming 2
		Scripting 2	Radgoll 3		
		Possui algum tipo	Terreno 3		
		de editor 3	Fluido 3		

Fonte: Jonas Goulart Pereira

Figura 21 - Funcionalidades tipo de aplicação simulação Unity

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Malhas(G)	Shaders(G)
BSP 1	Anisotropia 2	Lens Flare 2	Multitextura 1	Deformação 3	Vertex Shader 2
Portais 2	Volumétrica 3	Sist. Particulas 2	Bump mapping 2		Pixel Shader 2
Octrees 2	Radiosidade 3	Profud. De Campo 2			
Pag paisagem 3		Nevoa ou Ceu 2			
		Agua ou Fogo 3			
		Climáticos 3			
Terreno(G)	**Sombras(G)**	**Gerais(G)**	**Fisica (A)**	**Int. Artificial (A)**	**Som (A)**
Deformavel 3	Shadow volume 3	GUI 1	Detecção Colisão 1	Pathfinding 2	3D 2
		Entrada/saida 2	Veiculo 3		Streaming 2
		Scripting 2	Radgoll 3		
		Possui algum tipo	Terreno 3		
		de editor 3	Fluido 3		

Fonte: Jonas Goulart Pereira

Figura 22 - Funcionalidades tipo de aplicação simulação comparação

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Malhas(G)	Shaders(G)
BSP 1	Anisotropia 2	Lens Flare 2	Multitextura 1	Deformação 3	Vertex Shader 2
Portais 2	Volumétrica 3	Sist. Particulas 2	Bump mapping 2		Pixel Shader 2
Octrees 2	Radiosidade 3	Profud. De Campo 2			
Pag paisagem 3		Nevoa ou Ceu 2			
		Agua ou Fogo 3			
		Climáticos 3			
Terreno(G)	**Sombras(G)**	**Gerais(G)**	**Fisica (A)**	**Int. Artificial (A)**	**Som (A)**
Deformavel 3	Shadow volume 3	GUI 1	Detecção Colisão 1	Pathfinding 2	3D 2
		Entrada/saida 2	Veiculo 3		Streaming 2
		Scripting 2	Radgoll 3		
		Possui algum tipo	Terreno 3		
		de editor 3	Fluido 3		

Fonte: Jonas Goulart Pereira

As imagens 23, ,24 e 25 mostram as funcionalidades que se destacam no tipo de aplicação visualização.

Figura 23 - Funcionalidades tipo de aplicação visualização Panda3D

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Rede(A)	Sombras(G)	Gerais(G)
Lens Flare 2	Multitextura 1	Keyframes 2	Cliente/servidor 2	Shadow volume 3	GUI 1
Sist. Particulas 2	Bump mapping 2	Por esqueletos 2	Ponto a ponto 2		
Profud. De Campo 2	Formatos *	Facial 3			
Nevoa ou Ceu 2					
Agua ou Fogo 3					
Fisica (A)	**Som (A)**				
Detecção Colisão 1	3D 2				
Veiculo 3	Streaming 2				
Radgoll 3					
Terreno 3					
Fluido 3					

Fonte: Jonas Goulart Pereira

Figura 24 - Funcionalidades tipo de aplicação visualização Unity

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Rede(A)	Sombras(G)	Gerais(G)
Lens Flare 2	Multitextura 1	Keyframes 2	Cliente/servidor 2	Shadow volume 3	GUI 1
Sist. Particulas 2	Bump mapping 2	Por esqueletos 2	Ponto a ponto 2		
Profud. De Campo 2	Formatos *	Facial 3			
Nevoa ou Ceu 2					
Agua ou Fogo 3					
Fisica (A)	**Som (A)**				
Detecção Colisão 1	3D 2				
Veiculo 3	Streaming 2				
Radgoll 3					
Terreno 3					
Fluido 3					

Fonte: Jonas Goulart Pereira

Figura 25 - Funcionalidades tipo de aplicação visualização comparação

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Rede(A)	Sombras(G)	Gerais(G)
Lens Flare 2	Multitextura 1	Keyframes 2	Cliente/servidor 2	Shadow volume 3	GUI 1
Sist. Particulas 2	Bump mapping 2	Por esqueletos 2	Ponto a ponto 2		
Profud. De Campo 2	Formatos *	Facial 3			
Nevoa ou Ceu 2					
Agua ou Fogo 3					
Fisica (A)	**Som (A)**				
Detecção Colisão 1	3D 2				
Veiculo 3	Streaming 2				
Radgoll 3					
Terreno 3					
Fluido 3					

Fonte: Jonas Goulart Pereira

O Panda3D é motor de jogo Open-Source completo que utiliza linguagem Python e C++, possui ótimos gráficos e muitos efeitos visuais. A documentação é bem completa e possui uma grande contribuição da comunidade.

Com relação as métricas o motor Panda3D conseguiu 129 pontos dos 177 que poderia obter caso obtivesse pontuação máxima em todas as métricas, conseguindo assim 72,88% da pontuação total. Nos tipos de aplicações conseguiu 159 (129 + 30) pontos nas educacionais, onde obteve 30 dos 36 pontos que poderiam ser obtidos (83,33%). Nos simuladores 225 (129 + 96), com 91,42% dos 105 possíveis e 186 (129 + 57) nas de visualização, onde obteve 82,60% do máximo que é de 69 pontos.

O Unity3D possui um GUI muito interativo com o usuário, de modo a tornar o desenvolvimento muito prático, a tela de desenvolvimento permite colocar objetos, criar evento e regras para o jogo, com um feedback rápido.

Nas métricas em geral o Unity obteve 153 pontos dos 177 que poderia obter caso obtivesse pontuação máxima em todas as métricas, conseguindo assim 86,44% da pontuação total. Nos tipos de aplicações conseguiu 186 (153 + 33) pontos nas educacionais, onde obteve 33 dos 36 pontos que poderiam ser obtidos (91,66%). Nos simuladores 255 (153 + 102), com 97,14% dos 105 possíveis e 216 (153 + 63) nas de visualização, onde obteve 91,30% do máximo que é de 69 pontos.

A comparação dos dados obtidos mostrou que o motor de jogo Unity3D mostrou-se superior nas métricas em geral, e em todos os tipos de aplicação. Seguindo abaixo a lista:

a) geral: Unity (153, 86.44%), Panda3D (129, 72,88%);
b) educacional: Unity (33, 91.66%), Panda3D (30, 83.33%);
c) simulador: Unity (102, 97.14%) Panda3d (96, 91.42%);
d) visual: Unity (63, 91.30%), Panda3D (57, 82.60%).

Os recursos que o Panda3D possui no quesito geral são:

a) gerente de cena: geral, BSP, Portais, Octrees;
b) iluminação: *Lightmaps*, por-vertice, por-pixel, anisotropia, volumétrica,
c) radiosidade, BRDF;
d) efeitos especiais: *billboarding*, *lens flare*, sistema de partículas, profundidade, *motion blur*;
e) texturização: básica, Multitextura;
f) animação: *keyframes*, por esquelo, *blending*, *morphin*;

g) malhas: leitura, deformação, progressivas;
h) *shader*: *vertex shader*, *pixel shader*;
i) terreno: básico, deformável;
j) rede: estruturação, cliente/servidor;
k) sombra: *shadow map*;
l) API gráfica: OpenGL, DIrectX;
m) sistemas operacionais: Windows, Linux;
n) liguagem: C++, Python;
o) gerais: GUI, entrada/saída, *scripting*, editores, estado de desenvolvimento (estável);
p) física: básica, detecção de colisão, veículo, *radgoll*, terreno, fluido;
q) inteligência Artificial: *pathfinding*, tomada de decisão, script, formação;
r) som: básico, 3D, streaming;
s) suporte: exemplos, fórum, documentação.

Tipos de aplicação educacional:

a) efeitos especiais: *billboarding*, sistema de partículas;
b) texturização: multitextura;
c) animação: *keyframe*, por esqueleto, *blending*;
d) gerais: GUI, *scripting*, editores;
e) som: 3D;

Simulação:

a) gerente de cena: BSP, Potais, Octrees;
b) iluminação: anisotropia, volumétrica, radiosidade;
c) efeitos especiais: *lens flare*, sistema de partículas, profundidade de campo;
d) texturização: multitexturas;
e) malhas: deformação;
f) *shaders*: vertex *shader*, pixel *shader*;
g) terreno: deformação;
h) gerais: entreda/saída, *scripting*, editores;
i) som: 3D, streaming;

Visualização:

a) efeitos especiais: *lens flare*, sistema de partículas, profundidade de campo, *motion blur*;
b) texturização: multitextura;
c) animação: *keyframes*, por esqueletos;
d) rede: cliente/servidor;
e) gerais: GUI;
f) som: 3D, streaming;

Os recursos que o Unity possui no quesito geral:

a) gerente de cena: geral, BSP, Portais, Octrees, pag. paisagem;
b) iluminação: *Lightmaps*, *por-vertice*, por-pixel, anisotropia, volumétrica, radiosidade, BRDF;
c) efeitos especiais: *billboarding*, *lens flare*, sistema de partículas, profundidade, motion blur;
d) texturização: básica, Multitextura, *bump mapping*;
e) animação: *keyframes*, por esquelo, *blending*, *morphin*, *inv. Knematics*, *fwd knematicas*;
f) malhas: leitura, deformação, progressivas;
g) *shader*: vertex *shader*, pixel *shader*;
h) terreno: básico, deformável;
i) rede: estruturação, cliente/servidor, ponto a ponto;
j) sombra: *shadow map*, *shadow* volume;
k) API gráfica: OpenGL, DIrectX;
l) sistemas operacionais: BlackBerry 10, Windows Phone 8, Windows, OS X, GNU/LinuX, Android, iOS, Unity Web Player (incluindo Facebook), Adobe Flash, PlayStation 3, PlayStation 4, PlayStation Vita, Xbox 360, Xbox One, Wii U, e Wii;
m) linguagem: C++, C♯;
n) gerais: GUI, entrada/saída, *scripting*, editores, estado de desenvolvimento (estável);
o) física: básica, detecção de colisão, veiculo, *ragdoll*, terreno, fluido;
p) inteligência Artificial: *pathfinding*, tomada de decisão, script, formação;
q) som: básico, 3D, streaming;

r) suporte: exemplos, fórum, chat online, documentação.

Tipos de aplicação educacional:

a) efeitos especiais: *billboarding*, sistema de partículas;
b) texturização: multitextura;
c) animação: *keyframe*, por esqueleto, *blending*;
d) sombra: *shadow* volume;
e) gerais: GUI, *scripting*, editores;
f) som: 3D;

Simulação:

a) gerente de cena: BSP, Potais, Octrees, pag. paisagem;
b) iluminação: anisotropia, volumétrica, radiosidade;
c) efeitos especiais: *lens flare*, sistema de partículas, profundidade de campo;
d) texturização: multitexturas, *bump mapping*;
e) malhas: deformação;
f) shaders: vertex *shader*, pixel *shader*;
g) terreno: deformação;
h) sombra: *shadow* volume;
i) gerais: entreda/saída, *scripting*, editores;
j) som: 3D, streaming;

Visualização:

a) efeitos especiais: *lens flare*, sistema de partículas, profundidade de campo, motion blur;
b) texturização: multitextura, *bump mapping*;
c) animação: keyframes, por esqueletos;
d) rede: cliente/servidor, ponto a ponto;
e) sombra: *shadow* volume;
f) gerais: GUI;
g) som: 3D, streaming;

Porém a diferença entre os motores foi pequena, no quesito geral essa diferença se deu por Unity possuí uma grande portabilidade de plataformas, e por ter mais recursos gráficos. Nos tipo de aplicações o Unity ficou à frente sempre com alguma funcionalidade a mais, no tipo educacional a diferença

entre os pontos é quase inexistente, apenas por um recurso de animação, já no tipo de aplicação de simulação foi a texturização, e por final a aplicação para visualização a diferença se deu através das funcionalidades de rede.

CONCLUSÃO

Este trabalho aplicou métricas de avaliação para motor de jogo, afim de comparar motores open-source (Panda3D) e pago (Unity Pro), ambos presentes no mercado e em constante atualização, de modo a demonstrar para ao desenvolvedor independente, vantagens e desvantagens entre a utilização.

Ao decorrer de todo o trabalho todos os objetivos propostos foram alcançados. Pode se entender o conceito de jogos digitais, suas regras, sua limitação com relação ao mundo digital e seu desenvolvimento. Compreender os conceitos de motor de jogo, de sua arquitetura sobre a visão de alguns autores e componentes que agregam sua estrutura. Além de utilizar os motores o Panda3D e Unity, e entender o seu processo de desenvolvimento de jogos.

As métricas apresentadas nesse trabalho foram aplicadas com sucesso aos motores de jogo, assim obtendo resultados satisfatórios. Que por sua vez mostro a diferença de funcionalidades entre os motores avaliados Panda3D e Unity.

Os resultados mostraram que o Unity possui uma melhor pontuação em todos quesitos comparado ao Panda3D, mas essa diferença entre os dois motores foi bem pequena em todos os tipos de aplicações, e quase inexistente no tipo de aplicação educacional.

Os fatores que levaram a essa diferença de pontuação foram algumas funcionalidades gráficas avançadas, a grande portabilidade do Unity. Contudo ambos os motores tiveram ótimas pontuações em todos os tipos de aplicações e se mostraram ótimas escolhas para o desenvolvimento.

Com a utilização dos motores de jogo, e com a experiência adquirida, o Unity mostrou-se mais prático no desenvolvimento de um jogo digital do que o Panda3D, por possuir uma GUI muito dinâmica, de modo a poder desenvolver com um feedback muito rápido. Panda3D é basicamente scripts em Python e C++, por isso a necessidade de ter uma base de lógica e programação, e conhecimento na linguagem utilizada pelo motor, porém ele possui editores disponibilizados pela sua comunidade afim de facilitar o desenvolvimento do jogo.

Com relação a documentação ambos possuem um grande acervo, porem como a comunidade do Panda3D é menor comparada ao Unity, e no Brasil essa comunidade é menor ainda, existe pouquíssimas documentação sobre o Panda3D com a linguagem em português. Já o Unity possui um maior número de desenvolvedores brasileiros, que por consequência existem mais matéria disponível em português.

Para fins de custo de desenvolvimento no Unity pro, a versão paga do Unity utilizada para a desenvolvimento deste trabalho, acarreta em 75 dólares mensais ou 236 reais. Panda3D por ser um software open-source, não irá ter em nenhum custo para o desenvolvedor.

Houve uma dificuldade na aplicação das métricas, onde a soma de todas elas estabelecidas pelo artigo, não atingia a pontuação máxima que o motor de jogo poderia alcançar, demostrada também no artigo. Foi contatado o autor sobre o problema e assim constatando-se um erro de digitação do artigo. Para aplicação deste trabalho o problema foi corrigido, e depois as métricas foram aplicadas com êxito.

Com o conhecimento adquirido, pode se observar novos projetos, e trabalhos futuros que podem vir a ser desenvolvidos. Como exemplos: avaliação de outros motores e aplicação de novas métricas.

REFERÊNCIAS

ARRUDA, eucidio pimenta. **Fundamentos para o desenvolvimento de jogos digitais.** Porto Alegre: bookman, 2014.

BATTAIOLA, A.l. Jogos por computador: histórico, relevância tecnologia e mercadologia, tendências e técnicas de implementação. **Congresso Nacional da Sbc – Jornada de Atualização em Informação,** Curitiba, p.54-59, 12 set. 2000.

BITTENCOURT, João Ricardo; OSÓRIO, Fernando S.. **Motores para Criação de Jogos Digitais:** Gráficos, Áudio, Interação, Rede, Inteligência Artificial e Física. 2012. 36 f. Monografia (Especialização) - Curso de Curso de Engenharia da Computação, Unisinos, São Leopoldo, 2012.

BRANCO, marsal; MALFATTI, silvano; VINICIUS, marcus. **Jogos eletrônicos na prática.** 2. ed. novo hamburgo: feevale, 2013. 125 p.

BUSCHMANN, Frank et al. **Pattern-Oriented Software Architecture:** A System of Patterns. England: Wiley, 2009.

CAILLOIS, Roger. **Man, Play and Games.** Librairie Gallimard: Paris, 1958.

JUUL, Jesper. **A casual revolution:** reinventing video games and their player. Hong Kong: Printed And Bound, 2010.

CRAWFORD, Chris. **Chris Crawford on game design.** Printed: New Riders Publishing, 2003.

CHANDLER, Heather M. **Manual de Produção de Jogos Digitais.** 2. ed. Porto Alegre: Bookman, 2009.

CREIGHTON, Ryan Henson. **Unity 3D Game Development by Example Beginners Guide:** Lite. Birmingham: Packt Publishing Ltd, 2011. 104 p.

CREIGHTON, Ryan Henson. **Unity 3D Game Development by Example:** A Seat-of-Your-Pants Manual for Building Fun, Groovy Little Games Quickly. Birmingham: Packt Publishing Ltd, 2010. 364 p.

DOMINGUES, Rodrigo G. **Projeto de um framework para auxilio no desenvolvimento de aplicações com gráficos 3D e animação**. São Carlos, UFSCAR, 2003, 196 p.

DÜVEL, Oliver. **3D Game Engine Programming.** Portland: Premier Press, 2004. 860 p.

EBERLY, David H.. **3D Game Engine Design:** A Practical Approach to Real-Time Computer Graphics. San Antonio: Taylor & Francis, 2007. 1018 p.

EBERLY, David H.. **3D Game Engine Architecture:** Engineering Real-Time Applications with Wild Magic. Boca Raton: Taylor & Francis, 2005.

EBERLY, David H.. **3D Game Engine Architecture:** A Practical Approach to Real-Time Computer Graphics. Boca Raton: Taylor & Francis, 2007.

GREGORY, Jason. **Game Engine Architecture.** Boca Raton: Taylor & Francis, 2009.

GREGORY, Jason. **Game Engine Architecture:** Second Edition. Boca Raton: Taylor & Francis, 2014.

HARRISON, Lynn Thomas. **Introduction to 3D Game Engine Design Using DirectX 9 and C#.** New York City: Apress, 2013. 424 p.

HONORATO, RENATA. Para crescer, Brasil terá de apoiar os 'indie' games**, 2012. Disponível em: <http://veja.abril.com.br/noticia/vida-digital/para-crescer-brasil-tera-de-apoiar-os-indie-games>. Acesso em: 28 ago. 2014.**

HUIZINGA, Johan. **Homo Ludens.** Boston: Beacon, 1938.

LANG, Christoph. **Panda3D 1.7 Game Developer's Cookbook.** Birmingham: Packt Publishing Ltd, 2011. 336 p.

LAZZARI, Rafael. **Realismo visual: renderização de imagens em jogos computacionais.** 2011. 95 f. TCC (Graduação) - Curso de Ciência da Computação, Unesc, Criciuma, 2012.

LENGYEL, Eric. **Game Engine Gems 2.** Natick: A K Peters, 2011.

MADEIRA, André G. **Forge V8: Um Framework para o Desenvolvimento de Jogos de Computador e Aplicações Multimídia**. Recife, UFPE, 2001, 99 p.

MATHEWS, David Brian. **Panda3D 1.6 Game Engine Beginner's Guide.** Birmingham: Packt Publishing, 2011. 336 p.

MILLINGTON, Ian. **Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for your Game.** Boca Raton: Crc Press, 2010. 524 p.

NATAL, Daniel Milak. **Reuso, extensibilidade e simulação de física em motores para jogos digitais.** 2009. 83 f. TCC (Graduação) - Curso de Ciência da Computação, Unesc, Criciuma, 2010.

NORTON, Terry. **Learning C# by Developing Games with Unity 3D.** Birmingham: Packt Publishing Ltd, 2013. 292 p.

OKITA, Alex. **Learning C# Programming with Unity 3D.** Boca Raton: Crc Press, 2014. 690 p.

PASSOS, Erick Baptista et al. **Tutorial:** Desenvolvimento de Jogos com Unity 3D. Rio de Janeiro: Viii Brazilian Symposium On Games And Digital Entertainment, 2009.

PEREIRA, Daniel. **Desenvolvimento de um motor de jogos 3d, utilizando webgl.** 2012. 104 f. Monografia (Especialização) - Curso de Ciência da Computação, Furb, Blumenau, 2012.

PGT (núcleo de política e gestão tecnológiva). mapeamento da indústria brasileira e global de jogos digitais**.** são paulo: bndes, 2014.

PHAM, Andrew; PHAM, Phuong-van. **Scrum em Ação.** São Paulo: Novatec Editora, 2012. 288 p.

REZENDE, Denis Alcides. **Engenharia de Software e Sistemas de Informação.** Rio de Janeiro: Brasport, 2005. 316 p.

RIBEIRO, Ítalo Mendes da S.; SANTOS, Selan Rodrigues dos. Métricas para Avaliação de Motores de Jogos Tridimensionais. **Viii Brazillian Symposuim On Games And Digital Entertainment,** Rio de Janeiro, p.85-88, 10 ago. 2009.

RUSSELL, Jesse; COHN, Ronald. **Panda3d.** Philippines: Book On Demand, 2012.

SCHUYTEMA, Paul. **Game Design:** A Practical Approach. Charlottesville: Charles River Media, 2007.

SCHMITT, Heitor Augusto. **Mj3i-pa – um motor de jogos 3d para ios com personagens articulados.** 2012. 57 f. TCC (Graduação) - Curso de Ciência da Computação, Furb, Blumenau, 2012.

SUVAK, Janine. **Learn Unity3D Programming with UnityScript:** Unity's JavaScript for Beginners. New York City: Apress, 2014. 424 p.

THORN, Alan. **Game Engine Design and Implementation**. Woods Hole: Jones & Bartlett Publishers, 2011. 594 p.

TORI, Romero; KIRNER, Claudio; SISCOUTO, Robson. **Fundamentos e Tecnologia de Realidade Virtual e Aumentada.** Belém: Viii Sysmposium On Virtual Reality, 2006.

WATT, Alan H.; POLICARPO, Fabio. **3D Games:** Animation and Advanced Real-time Rendering. 2. ed. Boston: Addison-wesley Longman, 2003. 547 p.

APÊNDICE(S)

APÊNDICE A – Artigo

Avaliação das funcionalidades dos motores de jogos, Unity3d e Panda3

Jonas Goulart Pereira[1], Luciano Antunes[2]

[1]Ciência da Computação – Universidade do Extremo Sul Catarinense (UNESC)
Av. Universitária, 1105 – Bairro Universitário – Criciúma – SC - Brasil

jgpx@hotmail.com

Abstract. *Nowadays, the gaming development field of work have been increasing thanks to the fast advance in microprocessor technology. Today, any technologic device with a microprocessor can run a digital game, also these games don't target only the young male audience. Thus, gaming development became a great deal, wich independent developers also entered in this kind of business. The indie games have constantly increased it's audience, both by it's simple graphics and gripping stories, or by it's casual play-style focused and created by many these developers. However, to develop a game demand high cost that is funded by the it's own independent developer. As a result, this academic material makes a comparison between two game development engines, wich are tools used by digital game developers, where one has a paid license, Unity Pro, and the other has a free license, Panda3D. Evaluation metrics were used to show actual resources avaliable for each engine, for differente kinds of applications. Thus, indicating that both have almost the same functionalities except by some avaliable only in Unity. The Unity has shown a more practical engine for development and friendly user interface; on the other side, Panda3D obliges the developer to have a certain knowledge about the programming languages used by this engine. Both engines have a big and active community, with a great collection of documentation avaliable to new users.*

Resumo. *Atualmente a área de desenvolvimento de jogos digitais vem crescendo muito, e um dos motivos é o rápido avanço em microprocessadores. Hoje em dia, qualquer dispositivo com um microprocessador pode executar um jogo digital, além de que os jogos não tem mais como alvo apenas jovens do sexo masculino. Assim desenvolver jogos digitais virou um grande negócio, e desenvolvedores independe também entram nessa esquema. Os jogos indie tem conquistado um grande público, tanto pelos seus gráfico simples ou quem sabe pela suas histórias envolventes, ou também muitas vezes por um estilo de jogo mais casual que é criado por muitos desse desenvolvedores. Porém, desenvolver um jogo demanda custos, esses custos são muitas vez arcados pelo próprio desenvolvedor independente. Diante disto, este trabalho fez uma comparação entre dois motores de jogos, ferramenta utilizada pelos desenvolvedores de jogos digitais, entre um com uma licença paga, Unity Pro, e uma gratuito, Panda3D. Foram utilizado métricas demostrando funcionalidades existentes em cada motor, e para diferentes tipos de aplicações. Assim indicando que ambos possuem quase as mesma funcionalidade, excerto por algumas avançada onde apenas o Unity possui. O*

Unity se mostrou um motor mais prático para o desenvolvimento, com uma interface amigável, já o Panda3D possui a necessidade do desenvolvedor conhecer as linguagem utilizadas pelo motor. Além de que os dois possuem uma comunidade grande e ativa, com um grande acervo de documentos disponíveis para novos usuários.

1. Jogos Digitais

Jogos digitais engloba um conceito mais extenso que apenas o videogame. O termo esteve associado apenas aos jogos voltado aos consoles e fliperamas. Mas as Tecnologias digitais são com base na microinformática, que envolve jogos para computadores, consoles, fliperamas, smartphones, tablets e qualquer outro equipamento que venha a ser criada (ARRUDA, 2014). Desse modo esse termo se mostra mais abrangente em todo o contexto do formato de jogos, seja em vídeo ou outros.Jogo digital vêm em todas as formas e tamanhos, classificado em categorias ou gêneros cobrindo tudo, de Paciência à MMORPG (Massively Multiplayer On-line Role-Playing Games), e esses jogos são jogados em praticamente qualquer coisa com um microchip. Atualmente, você pode obter jogos para o seu PC, seu telefone celular, bem como uma série de diferentes jogos especializados em consoles, tanto de mão, quanto aqueles que se conectam a sua TV em casa. Estes consoles domésticos especializados tendem a representar a vanguarda da tecnologia de jogos, e o padrão destas plataformas sendo liberados em ciclos, que tem vindo a ser chamado de "gerações". As potências desta última geração são o Microsoft Xbox e o PlayStation da Sony (BITTENCOURT, 2012).

2. Motor de Jogo

O desenvolvimento e entretenimento de jogos digitais mostra um grande crescimento no mundo e agora no Brasil. E a criação e uma parte importante, tanto quanto uso de ferramentas para possibilitar um desenvolvimento mais prático e de maior qualidade nos jogos. Os motores de jogo são uma dessas ferramentas Desenvolvimento Rápido de Aplicação do Ingles, Rapid Application Development (RAD), por que permite acrescentar muitos recursos complexos aos jogos digitais. Como realismo, jogabilidade, e consequentemente um maior mercado (HARRISON, 2013, tradução nossa). Uma tarefa importante no desenvolvimento é escolha das ferramentas, conectada a definição do motor de jogo que é uma peça essencial para a criação do jogo. O motor de jogo permitirá ao desenvolvedor um grande reaproveitamento de código, e por assim um enorme ganho de tempo, ao utilizar-se das muitas funcionalidades, permitindo um desenvolvimento mais rápido de aplicações (WATT; POLICARPO, 2003, tradução nossa). A criação de jogos, seja um jogo casual para web 2D ou jogo mais complexo em 3D, pode ser acelerado de um jeito drástico, através do uso do motor de desenvolvimento de jogo. Existem diferentes motores para diferentes tipos de jogos, podemos citar o GameMaker que seu foco seria os jogos desenvolvidos em 2D, o RPG Maker focado em jogos de RPG, e ferramentas para jogo 3D como o OGRED, Unreal Engine, Unity, Panda3D entre outros (BITTENCOURT, 2012).

3. Métricas para motor de jogos

Para a comparação entre motores alguns elementos têm que ser destacados. Os principais fatores a serem avaliados são as suas funcionalidades, cinco categorias podem ser avaliadas: gráficas, acessórias, suporte, software e aplicações. As funcionalidades

gráficas são entre as atividades do motor para a renderização do ambiente. Nesta categoria entram os elementos como a geração de terreno, skybox, efeitos especiais como neblina, fogo e flare, texturas suportadas, geração de sombras, iluminação, etc. Física; conexão em rede; inteligência artificial são funcionalidades relacionadas acessórias. Entre as funcionalidades as relacionadas a suporte correspondem a documentação, suporte e a atividade da comunidade de desenvolvedores e usuários MILLINGTON, 2010, tradução nossa).

Outros fatores importantes de comparação são as linguagens suportadas e as plataformas de execução de um motor. A estrutura do motor é um elemento a ser considerado, que determina tanto a construção de um projeto ou aprendizagem. Essas funcionalidades compõem as métricas de software. Das métricas citadas, dependendo do tipo de aplicação algumas são mais ou menos importantes. Por exemplo, uma aplicação voltada ao simulador possui diferentes necessidades com relação a aplicação voltada ao entretenimento. Desse modo, dependendo do desenvolvimento da aplicação algumas métricas devem ser ressaltadas em relação a outras. As métricas de qualidade de software como portabilidade, eficiência e usabilidade estão relacionadas às métricas que serão apresentadas direta ou indiretamente, através das plataformas suportadas pelos motores (portabilidade), da documentação (usabilidade) e facilidade de criação dos elementos do ambiente (eficiência) (REBEIRO; SANTOS, 2009).

4. Análise comparativa de desempenho dos motores de jogos, Unity3d e Panda3d

O mercado de jogos digitais tem crescido, tanto quem sabe pelos seus gráficos simples ou pela suas histórias envolventes, jogos independentes são feitos com o próprio dinheiro e precisam de um tempo e de grande atenção para deixar um produto bom, o motor de jogo é a ferramenta para o desenvolvedor de jogos digitais. Então foram aplicados métrica entre os Panda3D open-source e Unity pro, para levantar possíveis desvantagens e vantagens em utilizar um motor pago ou gratuito/open source.

4.1. Metodologia

Existe uma variedade de motores de jogos disponíveis para diversos objetivos, como a criação de jogos, simulações, animações entre outros. Contudo, também existe uma grande necessidade de definição de mecanismos que permitam realizar comparações entre os motores, levando em consideração, por exemplo, parâmetros como qualidade de software, funcionalidades suportadas, facilidade de uso. Outro fator determinante que dever ser levado em consideração em uma eventual comparação de motores é a natureza da aplicação que se deseja desenvolver com o auxílio do motor.

Os motores de jogos foram comparados através de métricas, alguns fatores foram levados em consideração para ser realizada essa comparação entre esses motores, as funcionalidades são partes essenciais a serem consideradas, é comum ser avaliados em cinco diferentes fatores: acessórios, gráficas, software, suporte e aplicações. Correlação a funcionalidade gráfica compreendem-se os recursos de rederização do ambiente disponíveis no motor, ao suporte, no caso a atividade de comunicação entre desenvolvedor e usuários. Na funcionalidade de acessórios podemos destacar a simulação de física, que é de grande importância para o realismo e dinamismo no jogo; conexão de rede para partidas online através de redes entre computadores; inteligência artificial, que corresponde ao controle de NPCs. Correlação as métricas de qualidade de software podemos citar, portabilidade, eficiencia e usabilidade.

Parâmetros de métrica adiciona uma pontuação variando de zero à três, de acordo com o grau de dificuldade tecnológica para a existência do recurso no motor e a facilidade que ela introduz na criação de uma aplicação 3D. A categorização da pontuação é descrita a seguir:

Zero, inexistente: recurso não presente no motor.

Um, básica: recurso essencial, que deve estar presente em praticamente todo motor.

Dois, recomendada: recurso intermediário mas que aumenta a produtividade da aplicação e conduz a um melhor Efeito final para aplicação; presente na maioria dos motores.

Três, avançada: recurso existente em poucos motores, normalmente associado a efeitos e características específicas, que diferenciam o motor dos demais; muito útil no desenvolvimento.

Por exemplo, a presença de algum tipo de gerente de cena recebe pontuação 1, enquanto que se o mesmo motor apresentar uma técnica de otimização de cenário, como octree ou paginação de terreno, receberia nota 2. O cálculo total da pontuação de um motor é o somatório de todos os pontos obtidos nas diversas categorias da métricas.

As métricas que são incorporadas em um motor na forma de complemento (add-ons) também entram na contagem dos pontos. A figura 1 contém todas as métricas e suas respectivas pontuações, localizada ao lado de cada métrica.

As métricas onde a pontuação difere da forma estabelecida no início desta seção são marcadas com um "*" na Tabela e serão discutidas.

Gerente de cena (G)		Iluminação (G)		Efeitos Especiais (G)		Texturização (G)		Animação (G)		Malhas (G)	
Geral	1	*Lightmaps*	1	*Billboarding*	1	Básica	1	*Keyframes*	2	Leitura	1
BSP	1	Por-Vértice	1	*Lens flare*	2	Multitextura	1	Por esqueletos	2	Deformação	3
Portais	2	Por-Pixel	2	Sist. partículas	2	*Bump mapping*	2	*Blending*	2	Progressivas	3
Octrees	2	Anisotropia	2	Prof. de campo	2	Formatos	*	*Inv. knematics*	3	Formatos	*
Pag. paisagem	3	Volumétrica	3	Névoa ou Céu	2			*Fwd. knematics*	3		
		Radiosidade	3	Água ou Fogo	3			*Morphing*	3		
		BRDF	3	*Motion blur*	3			Facial	3		
				Climáticos	3						
Shaders (G)		**Terreno (G)**		**Sombras (G)**		**API Gráfica (G)**		**Sist. Op. (Sf)**		**Linguagens (Sf)**	
Vertex shader	2	Básico	1	*Shadow map*	1	OpenGL	*	Tipo	*	C/C++,	
Pixel shader	2	Deformável	3	*Shadow volume*	3	DirectX	*	Quantidade	*	Java, etc.	*
Gerais (G)		**Rede (A)**		**Física (A)**		**Int. Artificial (A)**		**Som (A)**		**Suporte (Sp)**	
GUI	1	Estruturação	1	Básico	1	*Pathfinding*	2	Básico	1	Exemplos	1
Entrada/saída	2	Cliente/servidor	2	Detecção colisão	1	Tomada decisão	2	3D	2	Fórum	2
Scripting	2	Ponto a ponto	2	Veículo	3	*Script*	2	*Streaming*	2	*Chat online*	2
Possui algum		Segurança	3	*Ragdoll*	3	Formação	3	Formatos	*	Documentação	2
tipo de editor	3			Terreno	3						
Estado desenv.	*			Fluído	3						
Custo	*										

Figura 1 – Métricas

5. Resultado Obtidos

As figuras 2, 3 e 4 monstram as fucionalidades e a comparação entre elas, destacados em amarelos são as fucionalidades que o Panda3D possuí, em azul o Unity e verde onde ambas possuem.

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Animção(G)	Malhas(G)	Shaders(G)	Terreno(G)	Rede(A)
Geral 1	Lightmaps 1	Billboarding 1(E)	Basica 1	Keyframes 2(E)(V)	Leitura 1	Vertex Shader 2(S)	Basico 1	Estruturação 1
BSP 1(S)	Por-Vértice 1	Lens Flare 2(S)(V)	Multitextura 1(E)(S)(V)	Por esqueletos 2(E)(V)	Deformação 3(S)	Pixel Shader 2(S)	Deformavel 3(S)	Cliente/servidor 2 (V)
Portais 2(S)	Por-Pixel 2	Sist. Particulas 2(E)(S)(V)	Bump mapping 2(S)(V)	Blending 2(E)	Progressivas 3			Ponto a ponto 2 (V)
Octrees 2(S)	Anisotropia 2(S)	Profud. De Campo 2(S)(V)	Formatos *(V)	Inv. Knematics 3	Formatos *	4	4	Segurança 3
Pag paisagem 3(S)	Volumétrica 3(S)	Nevoa ou Ceu 2(S)(V)	Formatos 1	Fwd. Knematics 3	Formatos 1			
	Radiosidade 3(S)	Agua ou Fogo 3(S)(V)		Morphin 3				
	BRDF 3(S)	Motion Blur 3		Facial 3(E)	8			3
6		Climáticos 3						
	15		5					
		15		9				

Sombras(G)	API Gráfica(G)	Sist. Op (Sf)	Linguagens (Sf)	Gerais(G)	Fisica (A)	Int. Artificial (A)	Som (A)	Suporte (Sp)
Shadow map 1	OpenGL *	Tipo *	C/C++	GUI 1(E)(S)(V)	Basico 1	Pathfinding 2(S)	Basico 1	Exemplos 1
Shadow volume 3 (E)(S)(V)	DirectX *	Quantidade *	Java, etc *	Entrada/saida 2(S)	Detecção Colisão 1(S)(V)	Tomada decisão 2	3D 2(E)(S)(V)	Fórum 2
				Scripting 2(E)(S)	Veiculo 3(S)(V)	Script 2	Streaming 2(S)(V)	Chat Onilne 2
	2	1	2	Possui algum tipo	Radgoll 3(S)(V)	Formação 3	Formatos *	Documentação 2
1				de editor 3(E)(S)	Terreno 3(S)(V)		2	
				est. De devem. *	Fluido 3(S)(V)	9		
				Custo *			7	5
					14			
				5				
				13				

Figura 2 – Funcionalidades geral Panda3D

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Animção(G)	Malhas(G)	Shaders(G)	Terreno(G)	Rede(A)
Geral 1	Lightmaps 1	Billboarding 1(E)	Basica 1	Keyframes 2(E)(V)	Leitura 1	Vertex Shader 2(S)	Basico 1	Estruturação 1
BSP 1(S)	Por-Vértice 1	Lens Flare 2(S)(V)	Multitextura 1(E)(S)(V)	Por esqueletos 2(E)(V)	Deformação 3(S)	Pixel Shader 2(S)	Deformavel 3(S)	Cliente/servidor 2 (V)
Portais 2(S)	Por-Pixel 2	Sist. Particulas 2(E)(S)(V)	Bump mapping 2(S)(V)	Blending 2(E)	Progressivas 3			Ponto a ponto 2 (V)
Octrees 2(S)	Anisotropia 2(S)	Profud. De Campo 2(S)(V)	Formatos *(V)	Inv. Knematics 3	Formatos *	4	4	Segurança 3
Pag paisagem 3(S)	Volumétrica 3(S)	Nevoa ou Ceu 2(S)(V)		Fwd. Knematics 3	Formatos 1			
	Radiosidade 3(S)	Agua ou Fogo 3(S)(V)	Formatos 2	Morphin 3				5
9	BRDF 3(S)	Motion Blur 3		Facial 3(E)	8			
		Climáticos 3	6					
	15			18				
		15						

Sombras(G)	API Gráfica(G)	Sist. Op (Sf)	Linguagens (Sf)	Gerais(G)	Fisica (A)	Int. Artificial (A)	Som (A)	Suporte (Sp)
Shadow map 1	OpenGL *	Tipo *	C/C++	GUI 1(E)(S)(V)	Basico 1	Pathfinding 2(S)	Basico 1	Exemplos 1
Shadow volume 3 (E)(S)(V)	DirectX *	Quantidade *	Java, etc *	Entrada/saida 2(S)	Detecção Colisão 1(S)(V)	Tomada decisão 2	3D 2(E)(S)(V)	Fórum 2
		3	1	Scripting 2(E)(S)	Veiculo 3(S)(V)	Script 2	Streaming 2(S)(V)	Chat Onilne 2
4	2			Possui algum tipo	Radgoll 3(S)(V)	Formação 3	Formatos *	Documentação 2
				de editor 3(E)(S)	Terreno 3(S)(V)		2	
				est. De devem. *	Fluido 3(S)(V)	9		7
				Custo *			7	
				3	14			
				11				

Figura 3 – Funcionalidades geral Unity

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Animção(G)	Malhas(G)	Shaders(G)	Terreno(G)	Rede(A)
Geral 1	Lightmaps 1	Billboarding 1(E)	Basica 1	Keyframes 2(E)(V)	Leitura 1	Vertex Shader 2(S)	Basico 1	Estruturação 1
BSP 1(S)	Por-Vértice 1	Lens Flare 2(S)(V)	Multitextura 1(E)(S)(V)	Por esqueletos 2(E)(V)	Deformação 3(S)	Pixel Shader 2(S)	Deformavel 3(S)	Cliente/servidor 2 (V)
Portais 2(S)	Por-Pixel 2	Sist. Particulas 2(E)(S)(V)	Bump mapping 2(S)(V)	Blending 2(E)	Progressivas 3			Ponto a ponto 2 (V)
Octrees 2(S)	Anisotropia 2(S)	Profud. De Campo 2(S)(V)	Formatos *(V)	Inv. Knematics 3	Formatos *	4	4	Segurança 3
Pag paisagem 3(S)	Volumétrica 3(S)	Nevoa ou Ceu 2(S)(V)	Formatos 1	Fwd. Knematics 3	Formatos 1			
	Radiosidade 3(S)	Agua ou Fogo 3(S)(V)	Formatos 2	Morphin 3				5
9	BRDF 3(S)	Motion Blur 3		Facial 3(E)	8			3
6		Climáticos 3	6					
	15		5	18				
		15		9				

Sombras(G)	API Gráfica(G)	Sist. Op (Sf)	Linguagens (Sf)	Gerais(G)	Fisica (A)	Int. Artificial (A)	Som (A)	Suporte (Sp)
Shadow map 1	OpenGL *	Tipo *	C/C++	GUI 1(E)(S)(V)	Basico 1	Pathfinding 2(S)	Basico 1	Exemplos 1
Shadow volume 3 (E)(S)(V)	DirectX *	Quantidade *	Java, etc *	Entrada/saida 2(S)	Detecção Colisão 1(S)(V)	Tomada decisão 2	3D 2(E)(S)(V)	Fórum 2
		3	1	Scripting 2(E)(S)	Veiculo 3(S)(V)	Script 2	Streaming 2(S)(V)	Chat Onilne 2
4	2	1	2	Possui algum tipo	Radgoll 3(S)(V)	Formação 3	Formatos *	Documentação 2
1				de editor 3(E)(S)	Terreno 3(S)(V)		2	
				est. De devem. *	Fluido 3(S)(V)	9		7
				Custo *			7	5
				3	14			
				5				
				11				
				13				

Figura 4 – Funcionalidades geral comparação

As figuras 5, 6 e 7 mostram as funcionalidades que se destacam no tipo de aplicação educacional.

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Sombras(G)	Gerais(G)	Som (A)
Billboarding 1	Multitextura 1	Keyframes 2	Shadow volume 3	GUI 1	3D 2
Sist. Particulas 2		Por esqueletos 2		Scripting 2	
		Blending 2		Possui algum tipo	
		Facial 3		de editor 3	

Figura 5 - Funcionalidades tipo de aplicação educacional Panda3D

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Sombras(G)	Gerais(G)	Som (A)
Billboarding 1	Multitextura 1	Keyframes 2	Shadow volume 3	GUI 1	3D 2
Sist. Particulas 2		Por esqueletos 2		Scripting 2	
		Blending 2		Possui algum tipo	
		Facial 3		de editor 3	

Figura 6 - Funcionalidades tipo de aplicação educacional Unity

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Sombras(G)	Gerais(G)	Som (A)
Billboarding 1	Multitextura 1	Keyframes 2	Shadow volume 3	GUI 1	3D 2
Sist. Particulas 2		Por esqueletos 2		Scripting 2	
		Blending 2		Possui algum tipo	
		Facial 3		de editor 3	

Figura 7 - Funcionalidades tipo de aplicação educacional comparação

As figuras 8, 9 e 10 mostra as funcionalidades que se destacam no tipo de aplicação simuladores.

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Malhas(G)	Shaders(G)
BSP 1	Anisotropia 2	Lens Flare 2	Multitextura 1	Deformação 3	Vertex Shader 2
Portais 2	Volumétrica 3	Sist. Particulas 2	Bump mapping 2		Pixel Shader 2
Octrees 2	Radiosidade 3	Profud. De Campo 2			
Pag paisagem 3		Nevoa ou Ceu 2			
		Agua ou Fogo 3			
		Climáticos 3			
Terreno(G)	**Sombras(G)**	**Gerais(G)**	**Fisica (A)**	**Int. Artificial (A)**	**Som (A)**
Deformavel 3	Shadow volume 3	GUI 1	Detecção Colisão 1	Pathfinding 2	3D 2
		Entrada/saida 2	Veiculo 3		Streaming 2
		Scripting 2	Radgoll 3		
		Possui algum tipo	Terreno 3		
		de editor 3	Fluido 3		

Figura 8 - Funcionalidades tipo de aplicação simulação Panda3D

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Malhas(G)	Shaders(G)
BSP 1	Anisotropia 2	Lens Flare 2	Multitextura 1	Deformação 3	Vertex Shader 2
Portais 2	Volumétrica 3	Sist. Particulas 2	Bump mapping 2		Pixel Shader 2
Octrees 2	Radiosidade 3	Profud. De Campo 2			
Pag paisagem 3		Nevoa ou Ceu 2			
		Agua ou Fogo 3			
		Climáticos 3			
Terreno(G)	**Sombras(G)**	**Gerais(G)**	**Fisica (A)**	**Int. Artificial (A)**	**Som (A)**
Deformavel 3	Shadow volume 3	GUI 1	Detecção Colisão 1	Pathfinding 2	3D 2
		Entrada/saida 2	Veiculo 3		Streaming 2
		Scripting 2	Radgoll 3		
		Possui algum tipo	Terreno 3		
		de editor 3	Fluido 3		

Figura 9 - Funcionalidades tipo de aplicação simulação Unity

Gerente de cena(G)	Iluminação(G)	Efeitos Especiais(G)	Textutização(G)	Malhas(G)	Shaders(G)
BSP 1	Anisotropia 2	Lens Flare 2	Multitextura 1	Deformação 3	Vertex Shader 2
Portais 2	Volumétrica 3	Sist. Particulas 2	Bump mapping 2		Pixel Shader 2
Octrees 2	Radiosidade 3	Profud. De Campo 2			
Pag paisagem 3		Nevoa ou Ceu 2			
		Agua ou Fogo 3			
		Climáticos 3			
Terreno(G)	**Sombras(G)**	**Gerais(G)**	**Fisica (A)**	**Int. Artificial (A)**	**Som (A)**
Deformavel 3	Shadow volume 3	GUI 1	Detecção Colisão 1	Pathfinding 2	3D 2
		Entrada/saida 2	Veiculo 3		Streaming 2
		Scripting 2	Radgoll 3		
		Possui algum tipo	Terreno 3		
		de editor 3	Fluido 3		

Figura 10 - Funcionalidades tipo de aplicação simulação comparação

As imagens 11, 12 e 13 mostram as funcionalidades que se destacam no tipo de aplicação visualização

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Rede(A)	Sombras(G)	Gerais(G)
Lens Flare 2	Multitextura 1	Keyframes 2	Cliente/servidor 2	Shadow volume 3	GUI 1
Sist. Particulas 2	Bump mapping 2	Por esqueletos 2	Ponto a ponto 2		
Profud. De Campo 2	Formatos *	Facial 3			
Nevoa ou Ceu 2					
Agua ou Fogo 3					
Fisica (A)	**Som (A)**				
Detecção Colisão 1	3D 2				
Veiculo 3	Streaming 2				
Radgoll 3					
Terreno 3					
Fluido 3					

Figura 11 - Funcionalidades tipo de aplicação visualização Panda3D

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Rede(A)	Sombras(G)	Gerais(G)
Lens Flare 2	Multitextura 1	Keyframes 2	Cliente/servidor 2	Shadow volume 3	GUI 1
Sist. Particulas 2	Bump mapping 2	Por esqueletos 2	Ponto a ponto 2		
Profud. De Campo 2	Formatos *	Facial 3			
Nevoa ou Ceu 2					
Agua ou Fogo 3					
Fisica (A)	**Som (A)**				
Detecção Colisão 1	3D 2				
Veiculo 3	Streaming 2				
Radgoll 3					
Terreno 3					
Fluido 3					

Figura 12 - Funcionalidades tipo de aplicação visualização Unity

Efeitos Especiais(G)	Textutização(G)	Animção(G)	Rede(A)	Sombras(G)	Gerais(G)
Lens Flare 2	Multitextura 1	Keyframes 2	Cliente/servidor 2	Shadow volume 3	GUI 1
Sist. Particulas 2	Bump mapping 2	Por esqueletos 2	Ponto a ponto 2		
Profud. De Campo 2	Formatos *	Facial 3			
Nevoa ou Ceu 2					
Agua ou Fogo 3					
Fisica (A)	**Som (A)**				
Detecção Colisão 1	3D 2				
Veiculo 3	Streaming 2				
Radgoll 3					
Terreno 3					
Fluido 3					

Figura 13 - Funcionalidades tipo de aplicação visualização comparação

O Panda3D é motor de jogo Open-Source completo que utiliza linguagem Python e C++, possui ótimos gráficos e muitos efeitos visuais. A documentação é bem completa e possui uma grande contribuição da comunidade.

Com relação as métricas o motor Panda3D conseguiu 129 pontos dos 177 que poderia obter caso obtivesse pontuação máxima em todas as métricas, conseguindo assim 72,88% da pontuação total. Nos tipos de aplicações conseguiu 159 (129 + 30) pontos nas educacionais, onde obteve 30 dos 36 pontos que poderiam ser obtidos (83,33%). Nos simuladores 225 (129 + 96), com 91,42% dos 105 possíveis e 186 (129 + 57) nas de visualização, onde obteve 82,60% do máximo que é de 69 pontos.

O Unity3D possui um GUI muito interativo com o usuário, de modo a tornar o desenvolvimento muito prático, a tela de desenvolvimento permite colocar objetos, criar evento e regras para o jogo, com um feedback rápido.

Nas métricas em geral o Unity obteve 153 pontos dos 177 que poderia obter caso obtivesse pontuação máxima em todas as métricas, conseguindo assim 86,44% da pontuação total. Nos tipos de aplicações conseguiu 186 (153 + 33) pontos nas educacionais, onde obteve 33 dos 36 pontos que poderiam ser obtidos (91,66%). Nos simuladores 255 (153 + 102), com 97,14% dos 105 possíveis e 216 (153 + 63) nas de visualização, onde obteve 91,30% do máximo que é de 69 pontos, como pode ser visto na figura 14.

Geral		
Unity3D	153	86,44%
Panda3D	129	72,88%
Total	177	100%
Educacional		
Unity3D	33	91,66%
Panda3D	30	83,33%
Total	36	100%
Simulação		
Unity3D	102	97,14%
Panda3D	96	91,42%
Total	105	100%
Visualização		
Unity3D	63	91,30%
Panda3D	57	82,60%
Total	69	100%

Figura – Resuldos das comparações

Os resultados mostraram que o Unity possui uma melhor pontuação em todos quesitos comparado ao Panda3D, mas essa diferença entre os dois motores foi bem pequena em todos os tipos de aplicações, e quase inexistente no tipo de aplicação educacional.

Os fatores que levaram a essa diferença de pontuação foram algumas funcionalidades gráficas avançadas, a grande portabilidade do Unity. Contudo ambos os motores tiveram ótimas pontuações em todos os tipos de aplicações e se mostraram ótimas escolhas para o desenvolvimento. Com relação a documentação ambos possuem um grande acervo, porem como a comunidade do Panda3D é menor comparada ao Unity, e no Brasil essa comunidade é menor ainda, existe pouquíssimas documentação sobre o Panda3D com a linguagem em português. Já o Unity possui um maior número de desenvolvedores brasileiros, que por consequência existem mais matéria disponível em português.

Para fins de custo de desenvolvimento no Unity pro, a versão paga do Unity utilizada para a desenvolvimento deste trabalho, acarreta em 75 dólares mensais ou 236 reais. Panda3D por ser um software open-source, não irá ter em nenhum custo para o desenvolvedor.

References

ARRUDA, eucidio pimenta. Fundamentos para o desenvolvimento de jogos digitais. Porto Alegre: bookman, 2014.

BITTENCOURT, João Ricardo; OSÓRIO, Fernando S.. Motores para Criação de Jogos Digitais: Gráficos, Áudio, Interação, Rede, Inteligência Artificial e Física. 2012. 36 f.

Monografia (Especialização) - Curso de Curso de Engenharia da Computação, Unisinos, São Leopoldo, 2012.Holton, M. and Alexander, S. (1995) “Soft Cellular Modeling: A Technique for the Simulation of Non-rigid Materials”, Computer Graphics: Developments in Virtual Environments, R. A. Earnshaw and J. A. Vince, England, Academic Press Ltd., p. 449-460.

HARRISON, Lynn Thomas. Introduction to 3D Game Engine Design Using DirectX 9 and C#. New York City: Apress, 2013. 424 p.

MILLINGTON, Ian. Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for your Game. Boca Raton: Crc Press, 2010. 524 p.

RIBEIRO, Ítalo Mendes da S.; SANTOS, Selan Rodrigues dos. Métricas para Avaliação de Motores de Jogos Tridimensionais. Viii Brazillian Symposuim On Games And Digital Entertainment, Rio de Janeiro, p.85-88, 10 ago. 2009.WATT, Alan H.; POLICARPO, Fabio. 3D Games: Animation and Advanced Real-time Rendering. 2. ed. Boston: Addison-wesley Longman, 2003. 547 p.

WATT, Alan H.; POLICARPO, Fabio. 3D Games: Animation and Advanced Real-time Rendering. 2. ed. Boston: Addison-wesley Longman, 2003. 547 p.