UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS APLICADAS A EDUCAÇÃO

DEPARTAMENTO DE CIÊNCIAS EXATAS

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ANÁLISE DE GAME ENGINES PARA PLATAFORMAS MÓVEIS

RENNAN ARAÚJO BARBOSA

Orientador: Prof. Dr. RAONI KULESZA

RIO TINTO - PB

2013

RENNAN ARAÚJO BARBOSA

ANÁLISE DE GAME ENGINES PARA PLATAFORMAS MÓVEIS

Monografia apresentada para obtenção do título de Bacharel à banca examinadora no Curso de Bacharelado em Sistemas de Informação do Centro de Ciências Aplicadas e Educação (CCAE), Campus IV da Universidade Federal da Paraíba.
Orientador: Prof. Dr. Raoni Kulesza.

RIO TINTO - PB
2013

B238a Barbosa,Rennan Araújo.
Análise de *game engines* para plataformas móveis / Rennan Araújo Barbosa. – Rio Tinto: [s.n.], 2013.
52f.: il. –
Orientador: Raoni Kulesza.
Monografia (Graduação) – UFPB/CCAE.

1. Tecnologia da Computação. 2. Jogos digitais – Dispositivos móveis. 3. *Game engines*. I. Título.

UFPB/BS-CCAE CDU: 004(043.2)

RENNAN ARAÚJO BARBOSA

ANÁLISE DE GAME ENGINES PARA PLATAFORMAS MÓVEIS

Trabalho de Conclusão de Curso submetido ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal da Paraíba, Campus IV, como parte dos requisitos necessários para obtenção do grau de BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Assinatura do autor:_____

APROVADO POR:

_____

Orientador: Prof. Dr. Raoni Kulesza
Universidade Federal da Paraíba – Campus IV

_____

Prof. Dr. Alexandre Scaico
Universidade Federal da Paraíba – Campus IV

_____

Prof. MSc. Carlos Eduardo Silveira Silva
Universidade Federal da Paraíba – Campus IV

RIO TINTO - PB
2013

Aos amigos, colegas e professores, agradeço pelo companheirismo e apoio durante este importante passo da minha carreira.

AGRADECIMENTOS

O orientador desta monografia, pelo seu exemplo de dinamismo e trabalho que é a maior lição que um professor pode dar a seu aluno, gostaria de agradecê-lo por ter acreditado na minha caminhada.

A família e amigos, pelo apoio sempre que necessário durante a vida acadêmica.

A todos os amigos, funcionários, professores e colegas que fazem parte do DCE da Universidade Federal da Paraíba que de uma forma ou de outra, contribuíram para a conclusão deste trabalho.

RESUMO

O avanço da tecnologia móvel proporciona poderosas plataformas para o desenvolvimento de jogos. O notável crescimento no uso de dispositivos móveis proporcionou o surgimento de um grande mercado de jogos digitais para este tipo de aparelho. Por consequência, segundo a *PriceWaterHouseCoopers* até 2016 este mercado será 36% maior que o de jogos para PC e consoles, indicando o potencial desta indústria. Tendo estes fatores em vista, este trabalho apresenta os jogos digitais e como são feitos, a importância do uso de *game engines* neste processo e a analise de três *engines* para desenvolvimento móvel. Com o objetivo de auxiliar os desenvolvedores no importante processo de escolha da *engine*, é proposto um processo comparativo que elege uma das *game engines* comparadas como a que mais facilita o desenvolvimento.

Palavras chave: plataformas móveis, *engine*, *game engine*, jogos, análise.

ABSTRACT

Mobile technology improvement provides powerful platforms for game development. The remarkable growth in the use of mobile devices afforded the emergence of a huge game market for this kind of gadget. Therefore, according to PriceWaterHouseCoopers until 2016 this market will be 36% bigger than the market of PCs and consoles, showing the potential of this industry. Having this factors in mind, this work presents digital games and how they are made, the importance of the use of game engines in this process and the review of three game engines for mobile development. Aiming to assist developers in the importante process of choosing an engine, it is suggested a comparative method that elects one of the three analysed game engines as the one that most facilitates development.

Keywords: mobile platforms, engine, game engine, games, comparative analysis.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

WYSIWYG	*What you see is what you get* (O que você vê é o que você tem)
DJCTQ	Departamento de Justiça, Classificação, Títulos e Qualificação
PEGI	*Pan European Game Information* (Informação Pan-Européia sobre Jogos)
ESBR	*Entertainment Software Rating Board* (Quadro de classificação de software de entretenimento)
API	*Application Programming Interface* (Interface de Programação de Aplicativos)

SUMÁRIO

1 INTRODUÇÃO

Há vários anos os jogos estão presentes na vida das pessoas, sejam os de tabuleiro, os de cartas e, com o advento dos computadores, os de consoles e dispositivos móveis.

O crescimento deste mercado denota a importância que tem tomado e explica o interesse em desenvolver jogos de maneira organizada e mais rápida para atender as necessidades e a demanda cada vez maior dos consumidores.

1.1 PROBLEMATIZAÇÃO

O desenvolvimento de jogos apresenta vários desafios e um deles é a implementação de funções que tentam representar a realidade, como: (1) gravidade; (2) colisões; (3) iluminação; (4) áudio; e (5) animação, entre outras.

Desenvolver esses aspectos em uma linguagem de programação é um trabalho árduo, as *game engines* ou motores de jogos vêm com a intenção de facilitar este problema, pois generalizam rotinas que se fazem necessárias na maioria dos jogos, e assim, promovem o reuso tornando possível que a equipe se foque em outros elementos que compõem o *game* em si.

Dentro de uma variedade de *engines* que podem ser escolhidas o desenvolvedor precisa pesquisar e analisar qual delas será a mais adequada para o desenvolvimento de seu jogo, logo questões devem ser respondidas: Qual *engine* é apropriada? Como escolhê-la?

Dada a importância da *engine* para o desenvolvimento de um game e a variedade de opções existentes, este trabalho irá analisar três *game engines* que agilizam o desenvolvimento de *games* para plataformas móveis.

1.2 JUSTIFICATIVA

Segundo a *PriceWaterHouseCoopers*[1], o mercado de jogos irá crescer para $83 bilhões de dólares em 2016, *smartphones* modernos com tecnologias melhores de hardware, *tablets* e *iPods* com sistemas operacionais como *Android* e *iOS* contribuem para o desenvolvimento de jogos em plataformas móveis.

[1] *Video games segment insights from the Entertainment & Media Outlook: PwC*. Disponível em: http://www.pwc.com/gx/en/global-entertainment-media-outlook/segment-insights/video-games.jhtml. Acesso em: 26/04/2013

A mesma pesquisa da *PriceWaterHouseCoopers* estima que os jogos móveis e online irão superar as vendas de *games* para PC e consoles em 2013 e que em 2016 será 36% maior, estes fatores reforçam a importância que o mercado tem tomado.

Neste mercado em crescimento o auxílio das *engines* mostra-se fundamental. A produtividade conferida ao desenvolvimento possibilita a chegada do jogo ao mercado de maneira mais rápida, atendendo a demanda, logo devido a complexidade do desenvolvimento deste tipo de software, o uso de *game engines* é peça chave (FURTADO, 2012).

1.3 OBJETIVOS

1.3.1 Objetivo Geral

- Analisar a utilização de *game engines* com o propósito de auxiliar os desenvolvedores a escolhê-la.

1.3.2 Objetivos Específicos

- Propor um processo para análise de *game engines*;
- Comparar 3 *game engines* na área de desenvolvimento de jogos digitais para dispositivos móveis.

1.4 ESTRUTURA DO TRABALHO

O capítulo 2, "Jogos Digitais", define jogo e descreve maneiras de classificá-lo, posteriormente aborda o planejamento destes jogos e os processos de desenvolvimento que podem ser utilizados.

Ainda no capítulo 2 são descritas as principais soluções para aumentar a produtividade do desenvolvimento de jogos: (1) as APIs multimídia; (2) as ferramentas sem programação; e (3) as *game engines*. Concluindo o segundo capítulo é descrito o modelo arquitetural que servirá de base para o processo comparativo, explicado no capítulo seguinte.

O capítulo 3, "Análise comparativa", propõe e justifica um processo para a análise comparativa das *engines* escolhidas, o coloca em prática analisando três *game engines* e finalmente resume a análise em uma tabela comparativa.

E no último capítulo, "Conclusão", são descritas as considerações finais do trabalho, bem como possíveis trabalhos futuros.

2 JOGOS DIGITAIS

Neste capítulo são descritas definições de jogos e maneiras de classificá-los, aborda também como pode ser realizado o planejamento de um jogo digital, e cita metodologias de desenvolvimento de jogos, após demonstrar a importância que o uso de processos de desenvolvimento tiveram na evolução da indústria por meio de um breve histórico.

2.1 AS DEFINIÇÕES DE JOGO

Desde os jogos de cartas e tabuleiro até os modernos jogos digitais, a indústria de *games* cresceu devido a evolução tecnológica do *hardware* e a criatividade dos desenvolvedores dando origem a inúmeros jogos com variadas classificações.

Assim como existem vários tipos de jogos, igualmente existem várias definições para jogo na literatura, algumas delas listadas na Tabela 2.1.

Tabela 2.1: Definições de jogos

"Um sistema formal, fechado, que subjetivamente representa um subconjunto da realidade." (CRAWFORD, 1982, p.7).
O dicionário mini Aurélio (2000, p.408) define jogo como: "Atividade física ou mental fundada em sistemas de regras que definem a perda ou o ganho."
É uma atividade que se processa dentro de certos limites temporais e espaciais, segundo uma determinada ordem e um dado número de regras livremente aceitas, e fora da esfera da necessidade ou da utilidade material. (HUIZINGA, 1990, p.147.).
"Um jogo é sistema no qual os jogadores se envolvem em um conflito artificial, definido por regras, que implica em um resultado quantificável." (SALEN; ZIMMERMAN, p. 95, 2012.).

2.2 CLASSIFICAÇÃO DOS JOGOS

Em meio a várias definições Parlett (1991) é peculiar sobre o assunto, o autor diz que a palavra jogo é usada para definir tantas atividades, que não é interessante insistir em uma definição proposta. Sendo assim, além de definições formais é possível classificar jogos quanto as suas características, seu gênero, a faixa etária recomendada e no âmbito de *games* digitais quanto aos gráficos.

Quanto a classificação dos jogos Crawford (1982) diz que os *games* podem ser classificados como: (1) de tabuleiro; (2) cartas; (3) atléticos; (4) infantis; e (5) de computador este último engloba várias outras plataformas como os *arcades* (fliperamas), videogames e plataformas móveis, por exemplo, no geral jogos digitais.

Dentro do escopo de jogos digitais, estes podem ter inúmeras classificações de acordo com o gênero, Azevedo et al. (2005) enumera doze deles e, em destaque cinco : (1) *adventure*; (2) estratégia; (3) esporte; (3) luta; (4) *role playing game;* e (5) educacional. Apesar disso, a criatividade dos *designers* traz novos gêneros, dessa forma podem surgir novas classificações a qualquer momento.

Jogos podem ser classificados, também, quanto a faixa etária sugerida, no Brasil não há um grupo especializado na classificação de *games*, entretanto o Departamento de Justiça, Classificação, Títulos e Qualificação (DJCTQ) trata o assunto seguindo diretivas de um órgão dos Estados Unidos e um da Europa, ESBR e a PEGI, respectivamente. Essas duas instituições são especializadas e fazem a separação em grupos de acordo com a idade que julgam sugerida dado o conteúdo do *game*.

Ainda é possível classificar jogos digitais quanto aos gráficos que são usados para desenvolvê-lo, abaixo alguns tipos de gráficos que jogos podem ser feitos:

1. 2D: jogos que usam duas dimensões para desenhar seus gráficos, através dos planos horizontal e vertical figuras são coloridas na tela.

Figura 1: Chrono de Chrono Trigger
Fonte: Wikipédia (2013)

2. 3D: jogos que possuem uma representação tridimensional do ambiente e dos objetos dentro dele, eles podem ser:

- Primeira pessoa: jogos onde o jogador não vê o personagem que controla por completo, é dado o ponto de vista do personagem ou qualquer que seja o objeto controlado pelo jogador como visto na figura 2.

Figura 2: Imagem do jogo N.O.V.A
Fonte: Gameloft (2013)

- Terceira pessoa: jogos onde o jogador consegue ver o personagem ou objeto que controla.

Figura 3: Imagem do jogo *Infinity Blade* 2
Fonte: *Infinity Blade Game* (2013)

3. 2.5D: jogos tridimensionais que usam apenas um plano para apresentação, ou seja, a perspectiva do jogador (câmera) está fixa como a de um *game* 2D.

Figura 4: Imagem do jogo Megaman X8
Fonte: Mobygames (2013)

4. Isométricos: são jogos 2D, mas que usam técnicas de desenho para simular um efeito 3D, como na figura 5.

Figura 5: Imagem do jogo *Final Fantasy Tactics Advance*
Fonte*: Gamefaqs* (2013)

5. Baseado em texto: jogos em que apenas texto é usado como meio de interação com o jogador.

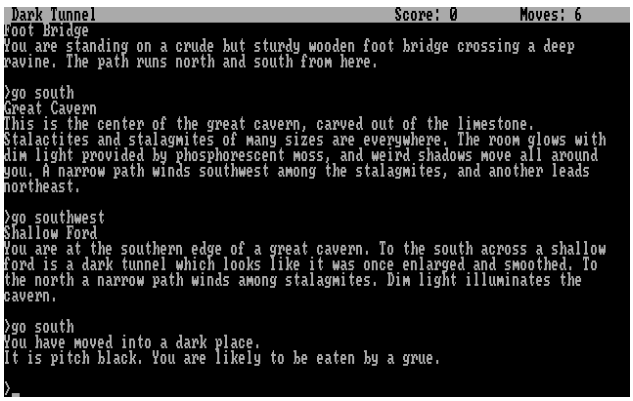

Figura 6: Imagem do jogo Zork
Fonte: *MobyGames* (2013)

2.3 PLANEJAMENTO DE JOGOS

Nesta seção é dado um resumo de como é feito o planejamento de um jogo, com base em Lavor (2009) que destaca partes importantes deste processo.

Planejamento é uma etapa importante de qualquer projeto, produzir uma ideia mal formada pode resultar em retrabalho e gastos que se mal administrados podem resultar no fim do projeto, então é importante investir tempo no planejamento do *game* para garantir um gerenciamento mais eficiente dos problemas (PERUCIA et al. 2005).

No planejamento de um jogo procura-se definir as suas características para poder criar o *Game Design Document* este documento possui todas as informações do projeto, como os requisitos do jogo e pesquisas de mercado.

Ele é um ferramenta importante para o *game designer* que pode tomar decisões mais rápidas com as informações registradas no documento (AZEVEDO et al., 2005).

As seguintes características devem ser definidas durante o planejamento de um jogo:

- Público alvo: A idade de quem irá jogar;
- Plataforma: É o hardware onde o jogo irá funcionar, esta é uma decisão importante, visto que, dependendo da plataforma escolhida podem haver mais gastos. Plataformas como: *Xbox 360, Playstation 3, Nintendo wii,* exigem a compra de *development kits* proprietários, para PCs e celulares existem várias *engines* e ferramentas, inclusive gratuitas que auxiliam o desenvolvimento;
- Game Design: Parte do planejamento onde são discutidas ideias para o jogo, como serão as fases, as personagens, os desafios do jogo, suas regras, registrando tudo no *Game Design Document*;
- Cronograma e orçamento: Planejam-se os gastos e prazos, para uma gerencia de tempo, recursos e acompanhamento do andamento do projeto.

O desenvolvimento de um jogo contempla diferentes áreas como: Artes Gráficas, programação, *game design*, gerência de projeto, sonorização, entre outras (CURTI, 2006) logo a equipe deve ser montada com profissionais destas áreas.

Tendo o planejamento do jogo a equipe montada irá desenvolver o projeto nas diferentes disciplinas envolvidas: modelar e animar os personagens de acordo com as características propostas, criar os sons e músicas que darão emoção ao *game* e programar o jogo. A programação é o que dá poder ao jogador e vai diferir o jogo de uma animação, tanto

o planejamento quanto o desenvolvimento fazem parte de um conjunto de fases das quais a maioria dos projetos de jogos partilham configurando um ciclo de vida no projeto dos jogos.

2.3.1 Ciclo de vida do projeto de jogos

Durante o planejamento e o desenvolvimento do jogo existem fases pelas quais eventualmente o projeto irá passar, Sloper (2002) cita cinco principais que são detalhadas abaixo:

- Concepção: É realizado um estudo de viabilidade do jogo, tendo em vista as limitações técnicas, operacionais e econômicas;
- Pré-produção: Neste momento são criados artefatos importantes no processo: o *Game Desing Document* e o *Technical Design Document*. O primeiro trata de características conceituais do jogo como: (1) design de personagens, (2) cenários e (3) sons, o segundo trata dos detalhes técnicos de como o conceitual será transformado em jogo, as ferramentas utilizadas no processo, a *engine*, por exemplo, são descritas aqui;
- Produção: Seguindo o definido no artefato *Game Design Document* o jogo é desenvolvido por meio da produção e união de código-fonte, arte, som e música;
- Pós-produção: São feitos testes para identificar possíveis defeitos e aqui começa o trabalho de divulgação do jogo para o seu lançamento;
- Pós-lançamento: Neste momento é dado suporte aos compradores do jogo e também é feito o monitoramento do sucesso do jogo verificando a possibilidade da produção de uma sequencia para o jogo ou conteúdo extra.

2.4 PROCESSOS DE DESENVOLVIMENTO DE JOGOS DIGITAIS

O desenvolvimento de jogos sofreu mudanças significativas desde o início. Nesta seção é feito um resumo dos primórdios da produção de games e um histórico dos processos de desenvolvimento de games.

2.4.1 Princípio do desenvolvimento de jogos digitais

Os primeiros jogos digitais foram programados em *Assembly* no começo dos anos 60, construídos com foco em desempenho já que os computadores tinham limitações grandes de *hardware*. A princípio feitos para computadores analógicos e na ausência de uma indústria os primeiros jogos foram produzidos por cientistas e entusiastas nas universidades.

No início dos anos 70 a indústria tomou forma e agora times trabalhavam na produção dos jogos para fliperamas, ainda focados em aproveitar o máximo do *hardware.* Mesmo dez anos depois, nos anos 80, jogos eram feitos para rodar em computadores com processadores de 4Mhz e 64Kb de memória, por exemplo. (ROLLINGS; MORRIS, 2000).

De acordo com Furtado (2012), devido a tais limitações era complicado rodar jogos compilados em C já que ficariam muito pesados nas máquinas e aponta complicações que *Assembly* provoca: a linguagem torna o *debug* um verdadeiro desafio e complica uso de conceitos importantes da engenharia de software como modularidade e reuso, caracterizando um cenário onde havia falta de organização e processos.

No decorrer do tempo o padrão estético e técnico dos jogos crescia assim como a exigência do consumidor mostrando que o uso de uma linguagem de alto nível e de engenharia de software era importante no processo de desenvolvimento do jogo, dadas as complicações enfrentadas com *Assembly* (FURTADO; SANTOS, 2002).

Em 1993 foi criado *Doom*, o primeiro jogo completamente feito em C, mostrando que é possível usar linguagem de alto nível na produção de jogos nos *PCs* da época. Por conseguinte o surgimento de novos compiladores, máquinas mais poderosas e o uso da engenharia de software fizeram com que soluções de mais alto nível fossem desenvolvidas com o intuito de melhorar o processo de criação do jogo (FURTADO, 2012, p.12).

2.4.2 Processos de desenvolvimento de jogos

Como visto na seção anterior o desenvolvimento de jogos, nos primórdios, não contava com metodologias de desenvolvimento, tinha uma abordagem *ad hoc* dadas as dificuldades impostas pelo uso de *Assembly,* não havia um processo nem papéis bem definidos e times pequenos produziam os jogos com foco em desempenho. Porém, com o crescimento na demanda de jogos mais modernos o que funcionou bem entre 1950 e 1960, utilizando processos *ad hoc,* não funcionaria tão bem nos anos 70, surgia então processo cascata (ARAUJO, 2006) (FLYNT, 2005).

2.4.2.1 *Game Waterfall Process*

O processo cascata da engenharia de *software* foi adotado pelos desenvolvedores de *games* nos anos 70, com cinco fases definidas que ocorrem sequencialmente. O *Game Waterfall Process* é uma adaptação do processo *Waterfall,* usado na produção de *software* geral para o âmbito dos games (FLYNT, 2005). Como visto na figura 7, em azul o processo original e em vermelho o adaptado para o ciclo de vida do projeto de jogos.

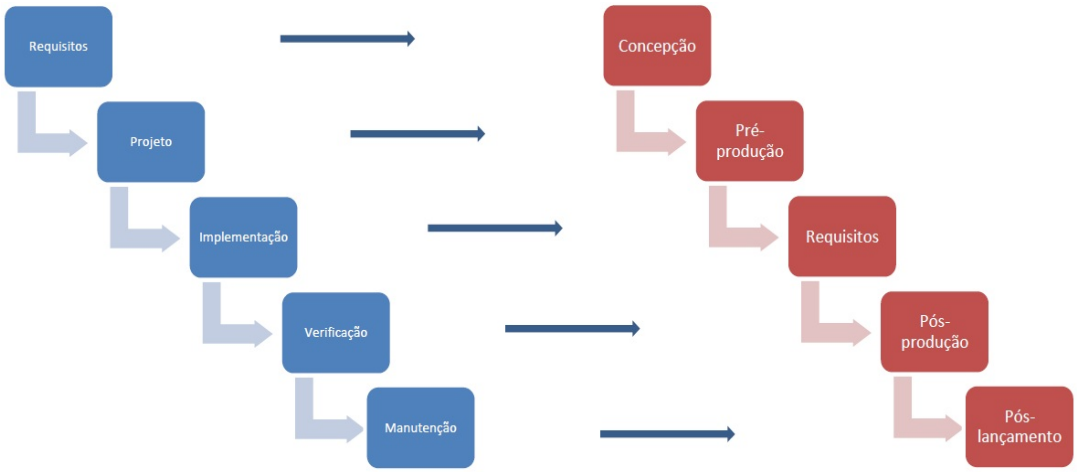

Figura 7: Processos Waterfall e Game Waterfall
Baseado em: (BARROS, 2007)

2.4.2.2 Processos ágeis, iterativos e incrementais

Nos anos 90 a indústria de jogos começou a adotar processos iterativos e incrementais como: (1) XP (*Extreme Programing*); (2) RUP (*Rational Unified Process*); e (3) *Scrum*, da mesma maneira que foram descritos para software. Porém com o tempo adaptações destes processos surgiram o GUP (*Game Unified Process*) e o XGD (*Extreme Game Development*), descritos abaixo:

2.4.2.2.1 XGD

É uma metodologia ágil de desenvolvimento de jogos criada na empresa de jogos francesa Titus, esta possui as mesas características do XP só que algumas práticas foram adaptadas para membros de outras especialidades que não a programação, por exemplo: (1) artistas; (2) engenheiros de som; e (3) modeladores 3D (BARROS, 2007).

Como o XP o XGD baseia-se em cinco valores: (1) comunicação; (2) simplicidade; (3) *feedback*; (4) coragem; e (5) respeito. O XGD também manteve as práticas do XP, algumas delas descritas na tabela 2:

Tabela 2.2: Práticas do XGP

Prática	Descrição
Whole Team	A equipe deve ser coesa, trabalhar de forma multidisciplinar e manter comunicação sempre
Stand-up Meetings	Reuniões rápidas feitas de pé para manter o time ciente do andamento do projeto e das tarefas.
User Stories	São descrições simples das funcionalidades do jogo.
Collective Ownership	Toda a equipe é responsável pelo código do jogo e qualquer membro pode alterá-lo.

Fonte: Wikipedia (2013), BARROS (2007)

2.4.2.2.2 GUP

O GUP (*Game Unified Process*) tinha conceitos de XP e RUP, nele a equipe poderia variar o estilo do processo de acordo com a necessidade sendo mais voltado ao RUP quando necessitava de formalidades e documentação e mais ao XP quando menos formal (FURTADO, 2012, p.22).

2.5 DAS APIS AS *GAME ENGINES*

A seção 2.4.1 "Princípio do desenvolvimento de jogos digitais", diz que a evolução do *hardware* e a adoção de engenharia de software foi acompanhada da criação de soluções de mais alto nível para melhorar o processo de desenvolvimento de jogos digitais.

Esta seção é responsável por descrever tais soluções e como elas contribuíram para o surgimento das *game engines* utilizadas nos dias de hoje.

2.5.1 *APIs* Multimídia

As *APIs* Multimídia são bibliotecas que permitem acesso facilitado ao *hardware* do dispositivo como entrada de dados, placas de vídeo e de som, desta maneira elas proporcionam uma nova camada de abstração para o desenvolvedor, que sem a *API* teria de programar de acordo com o hardware, com esta nova camada (figura 8) a preocupação com detalhes de baixo nível do dispositivo não existe, proporcionando flexibilidade.

Figura 8: API multimídia é uma nova camada de abstração.
Baseado em: Furtado (2012)

Como as APIs vão tratar o *hardware* elas têm a liberdade de personalizar os algoritmos de acordo com o dispositivo para melhorar ainda mais o desempenho, sem que seja necessária a intervenção do programador do jogo, segundo Furtado (2012) a grande maioria das APIs multimídia suportam funções de gráficos, sons e entrada, e jogos digitais modernos estão direta ou indiretamente utilizando os benefícios destes algoritmos otimizados.

As seguintes funções estão presentes em quase todas as APIs multimídia: (1) alterar o modo de apresentação do *display*, (2) desenhar e ler *pixels* na tela, (3) rolar a tela, (4) ler entrada do usuário, (5) tocar música e efeitos sonoros (FURTADO, 2012).

APIs multimídia famosas usadas na atualidade são o DirectX e o OpenGL, esta última tem um versão chamada OpenGL ES feita para potencializar o uso do hardware de dispositivos móveis.

Mesmo sendo úteis na abstração da interação com o hardware as APIs Multimídia não são feitas especificamente para jogos e sim para aplicações multimídia no geral, logo toda a programação do jogo tem que ser feita. Por exemplo, transição entre fases e inteligência artificial ainda são implementadas pelo desenvolvedor, logo, elas ainda não são a solução ideal.

2.5.2 Ferramentas Sem Programação

Com a intenção de fazer os jogos sem a necessidade de programação surgiram as *Click-n-play tools* (Ferramentas de *click-n-play*), com elas a programação é feita de forma visual utilizando uma interface gráfica para que a produção de jogos possa ser feita sem que o desenvolvedor precise programar. Essas ferramentas traduzem os comandos editados na

interface gráfica para um API Multimídia permitindo funções como: (1) controle de *sprites*[2]; (2) menus; e (3) sons do jogo, estas ferramentas podem ser focadas em um gênero específico de jogo ou não como o *Game Maker* [3](figura 9).

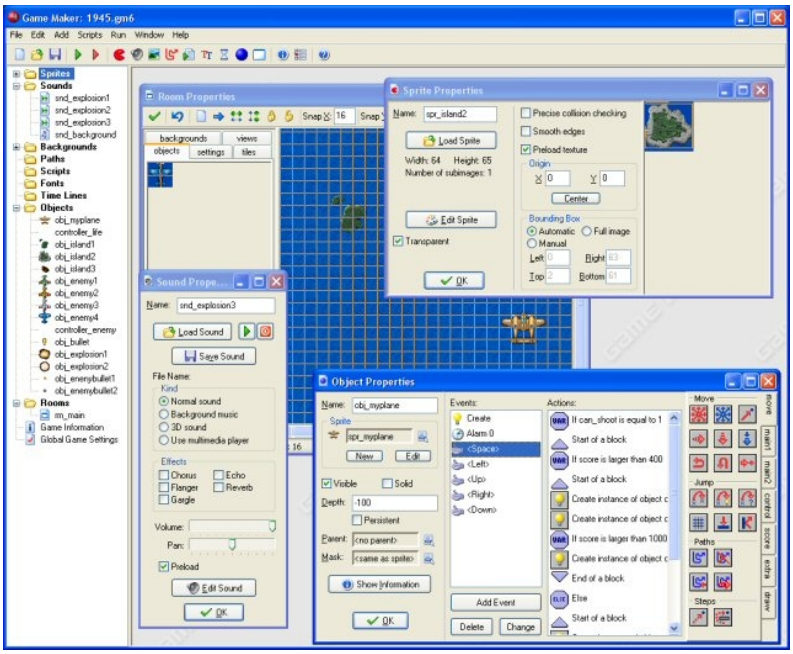

Figura 9: Imagem do *Game Maker*
Fonte: Romero Games[4]

O uso destas ferramentas é interessante principalmente pelo fato de não ser necessário conhecimento de linguagens de programação, proporcionando a liberdade de criar jogos só com cliques, porém, nem todos os jogos podem ser criados por estas ferramentas, já que os recursos oferecidos por elas são limitantes dificultando iniciativas mais criativas.

Na tentativa de aumentar a liberdade de criação, algumas destas ferramentas colocaram suporte a linguagens para dar mais flexibilidade, exigindo conceitos básicos de programação.

As versões mais recentes do *Game Maker* possuem a GML (*Game Maker Language*). Todavia, Furtado (2012) aponta que o fato de exigir estas habilidades fere o princípio de não programação proposto pelas *Click-n-play Tools,* e que ao aprender a programar os usuários eventualmente irão buscar outras alternativas como as *game engines* que foram desenvolvidas para programadores.

[2] Conjunto de imagens que quando exibidas em sequencia formam uma animação.

[3] Game Maker Studio. Disponível em: http://www.yoyogames.com/gamemaker/studio Acesso em: 02/05/2013.

[4] Romero Games *Game Maker*. Disponível em: http://romerogames.blogspot.com.br/2011/10/game-maker.html Acesso em: 07/05/2013.

2.5.3 Game Engines

A *engine* é um software que abstrai a implementação de rotinas comuns aos jogos, como renderização e física, para que assim a equipe de desenvolvimento possa focar nos detalhes que fazem o jogo único (WARD, 2008), dessa forma uma *engine* será o software que interpretará a entrada (*touchscreen*, controle, teclado) de acordo com a lógica do jogo e apresentará uma saída.

Tendo funções variadas as *engines* podem inclusive apresentar soluções que tratam o envio mensagens para servidores (CLUA; BITTENCOURT, 2005). São ferramentas que buscam auxiliar o programador de jogos, de modo que ele possa direcionar o esforço que seria gasto com a programação de funções que já estão prontas na *engine* para as funções específicas do seu jogo.

As *engines* são construídas em uma camada acima das APIs, dando um nível ainda maior de abstração para os desenvolvedores como visto na figura 10. Focadas em prestar suporte ao desenvolvimento de jogos complexos por programadores são bem mais flexíveis e poderosas que as *Click-n-play Tools*, algumas delas proporcionam inclusive interface gráfica integrada para desenvolver cenários e modelagem 3D (FURTADO, 2012).

Figura 10: A *Game Engine* é uma nova camada de abstração
Baseado em: Furtado (2012)

Justamente devido a maior abstração, encapsulamento e reuso que as *engines* proporcionam elas se tornaram o estado da arte no desenvolvimento de muitos jogos digitais. Estas características deram um nível de produtividade que ainda não havia sido visto na indústria (FURTADO, 2012), permitindo que jogos tivessem um tempo mais curto para a chegada no consumidor final. Isto ressalta a importância da escolha de uma *engine* apropriada, processo pelo qual o projeto de jogo é passível, destacando a necessidade de uma análise das *engines* disponíveis e o que elas proporcionam para o projeto do jogo.

Existem várias no mercado e cada uma delas terá soluções para diferentes necessidades, há *engines* com soluções de áudio, algumas com soluções de inteligência artificial, enquanto outras não terão, há também as que são orientadas para plataformas como as *engines* para dispositivos móveis, já que estes tipos de aparelho apresentam necessidades específicas (por exemplo, forma específica de entrada, processamento) e plataformas variadas (iOS *e Android*).

Abaixo alguns fatores que podem ser levados em consideração para ajudar nesta escolha (CLUA; BITTENCOURT, 2005):

- Orçamento: A variação e a complexidade podem variar muito, é importante investigá-los para garantir o custo-benefício da adoção da *engine*;
- Gênero do *game*: Alguns motores podem favorecer o desenvolvimento de um gênero específico, ou podem criar diferentes tipos de jogos;
- Tempo: O uso de *engines* com um nível de abstração muito alto pode criar jogos muito parecidos, porém o desenvolvimento será mais rápido, já motores menos especializados permitem mais liberdade ao implementar, podendo assim criar experiências mais exclusivas, ao custo de mais tempo na codificação;
- Plataforma: A *engine* precisa suportar a plataforma escolhida;
- Documentação oferecida: A documentação da *engine* ajuda o desenvolvedor a compreender melhor os recursos e a maneira correta de usá-los;
- Ferramentas disponíveis: A *engine* escolhida precisa compreender o conteúdo desenvolvido pela equipe em outras ferramentas.

2.6 ARQUITERUTA DAS *GAME ENGINES*

Devido as capacidade de sistemas de software como as *game engines* elas podem se tornar bastante complexas com diversas camadas e subsistemas.

A arquitetura apresentada é baseada no trabalho de Gregory (2009) que busca generalizar os componentes que as *engines* podem dispor, descrevendo suas reponsabilidades. Cada camada representa um conjunto de componentes que utiliza as funcionalidades providas pela camada inferior.

A figura 10 resume as camadas e componentes que as *engines* podem apresentar:

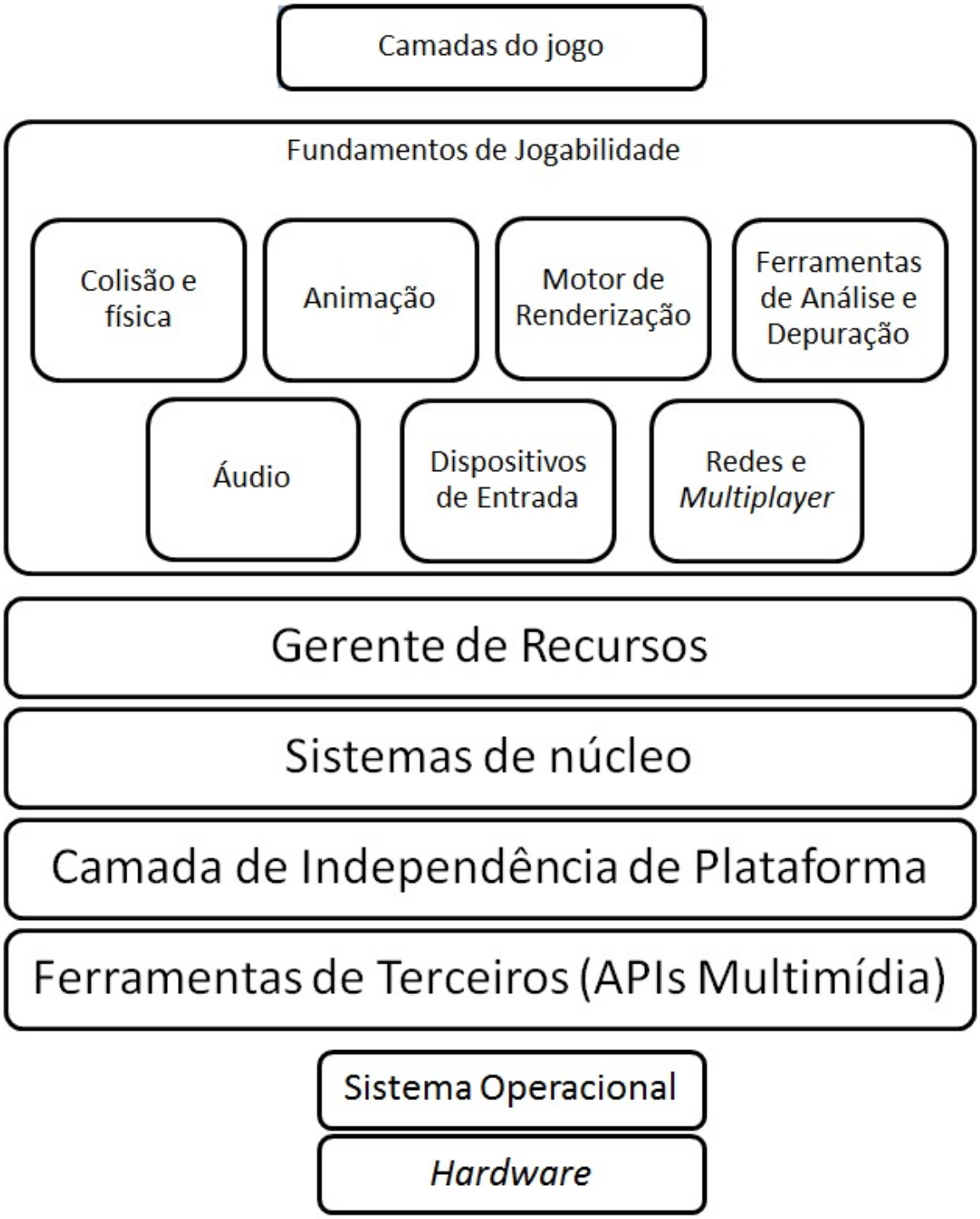

Figura 11: Arquitetura de uma *game engine*
Baseado em: Gregory (2009)

A seguir são descritos cada módulo dessa arquitetura:

- **Soluções de terceiros:** As *engines* também podem usufruir de *APIs* e *engines* de terceiros, adicionando estes componentes a sua arquitetura. Soluções de física e colisões, gráficos, animação e áudio podem ser utilizadas, certas empresas se especializam nestes campos disponibilizando *APIs* e *engines* especializadas. *APIs* de gráficos famosas são *DirectX* e *OpenGL ES* que acessam e abstraem o *hardware* como visto na seção 2.5.1 "*APIs* Multimídia", já para física temos as *engines* de física e colisão *PhysX*[5] e *Havoc*[6] são populares e poderosas, a primeira está disponível gratuitamente.

- **Camada de Independência de Plataforma:** Responsável por generalizar chamadas ao sistema operacional e ao *hardware*, vários jogos são feitos para funcionar em múltiplas plataformas dada, então, a necessidade de *engines* com a capacidade de produzir jogos multiplataforma. Inserindo uma camada para independência as acima não se preocupam com a plataforma na qual o *game* irá operar delegando a responsabilidade da compatibilidade para esta camada. Ela fica acima das camadas de soluções para terceiros, sistema operacional e *hardware*, justamente para tratar as diferenças entre as plataformas.

- **Sistemas de núcleo:** Proporciona um conjunto de utilidades de sistemas, como: (1) estruturas de dados e algoritmos; (2) Gerenciamento de memória; (3) Assertivas; e (4) Bibliotecas de matemática, por exemplo.

- **Gerente de Recursos:** Responsável por gerenciar os ativos da *engine*, esta camada organiza os recursos que serão usados através de uma interface pela qual será disponibilizado o acesso aos diversos tipo de ativos que o jogo terá, por exemplo: (1) arquivos de textura; (2) de gráficos; e (3) fontes tipográficas. Todas as *engines* devem disponibilizar este acesso de alguma maneira, se ele vai ser organizado ou *adhoc* dependerá do criador dela.

[5] Physx, Disponível em: http://www.geforce.com/hardware/technology/physx Acesso em: 26/05/2013
[6] *Havoc Physics,* Disponível em: http://www.havok.com/products/physics Acesso em: 26/05/2013

- **Motor de renderização:** Responsável por desenhar na tela é uma camada complexa em jogos 3D, abaixo os componentes desta camada:

    - **Renderizador de baixo nível:** Com o uso das APIs de gráfico, como DirectX e Open GL, o renderizador será responsável por desenhar as primitivas gráficas dados os pontos geométricos da melhor maneira possível, sejam eles texturas, iluminação, texto, e até campos de visão da câmera no caso de jogos 3D.

    - **Otimização de Gráficos:** Formas de otimizar os gráficos para poupar *hard-ware* estará aqui, caso a *engine* consiga implementá-las automaticamente. O renderizador de baixo nível desenha tudo que for passado para ele e sem se preocupar com o que será visível ou não, no caso de jogos 3D, em determinadas situações, a posição da câmera pode deixar objetos fora do campo de visão dado ao jogador, desta maneira este componente pode melhorar o desempenho do jogo, sabendo o que está sendo visualizado e poupar recursos, não desenhando o que esta fora do campo de visão, otimizando o jogo. A possibilidade de verificar e alertar a presença de muitas imagens ou partículas, alterando a resolução para um melhor desempenho é outro exemplo.

    - **Efeitos Visuais:** *Engines* podem possuir uma área dedicada a tratar efeitos visuais, detalhes como partículas na tela (fogo, fumaça, respingos de água) e sombras dinâmicas.

    - **Interface gráfica com o usuário:** Responsável por desenhar os menus do jogo, onde o jogador escolhe, por exemplo, o nome e a aparência do personagem, a fase que quer jogar, áreas da tela dedicadas a exibir a vida, o mapa, a quantidade de balas e inúmeras outras interfaces gráficas que os produtores acharem necessárias para a gerência do jogo. Outra responsabilidade é a exibição de vídeos e animações pré-renderizadas na tela, um recurso interessante bastante utilizado em jogos modernos.

- **Ferramentas de análise e depuração:** Pode ser feita por uma ferramenta externa ou estar dentro da *engine*, é responsável por prover ao desenvolvedor ferramentas que

possibilitem análise de dados da execução do jogo, com estes dados a equipe pode descobrir detalhes o quanto o jogo consome em recursos de o hardware, um exemplo de ferramenta de análise é a capacidade de exibir dados da execução do jogo em tempo real como memória utilizada pela *engine* impressos na tela do jogo, ou colocar estes dados em um arquivo externo, outra é a possibilidade do programador colocar frases em locais específicos do código para depurar erros como um *logger*.

- **Colisão e Física:** Esta também pode ser provida por ferramentas de terceiros e normalmente é, pelo fato destas ferramentas serem especializadas, esta camada trata da colisão e física dos objetos do jogo, geralmente quando um objeto do jogo colide com o outro uma reação física é disparada por isso estas duas andam juntas. Sendo assim, esta camada é responsável por algoritmos que calculam: (1) força; (2) gravidade; (3) velocidade; enfim, a simulação da física real para o mundo virtual. Devido a importância de uma simulação realista e da complexidade em criá-las as empresas tem escolhido por não desenvolvê-las e utilizar ferramentas de terceiros.

- **Animação:** Esta camada irá ter a responsabilidade de resolver o problema de simulação de movimentos dos personagens. Em um jogo 2D os elementos são animados através de *sprites*, estes são figuras repetidas que quando postas em sequencia simulam movimentos, a troca de uma figura por outra na tela, dá a ideia de movimentação.

Figura 12: Exemplo de *Sprite*
Fonte: Yoyogames (2013)

Já para jogos 3D esta área pode ficar extremamente complexa, o processo de animação de personagens tridimensionais podem ficar o quão complexo a tecnologia, limitações de *hardware* e o projeto do jogo permitirem, um exemplo deste processo é a captura de movimentos de atores reais refletidos em esqueletos virtuais, é possível transpor os movimentos do corpo real para as modelagens 3D.

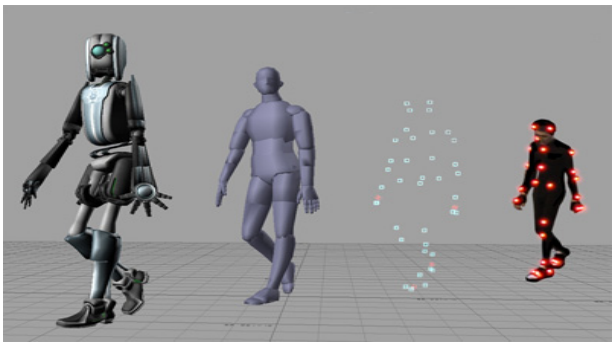

Figura 13: Captura de movimentos
Fonte: Wikipedia (2013)

- **Dispositivos de entrada:** São os periféricos pelos quais o jogador irá interagir com o jogo como: (1) controle; (2) teclado e mouse; (3) *touchscreen*; entre outros dependendo da plataforma. Esta camada deve disponibilizar um meio de compreender os dados que vem do periférico e passando para o desenvolvedor que vai traduzir em uma ação, e em alguns casos enviar respostas para o *hardware* do periférico como vibrar o controle do jogador, por exemplo.

- **Áudio:** Assim como os gráficos é parte importante nos jogos digitais, as *engines* devem prover suporte a áudio, esta camada trata das peculiaridades de dados de áudio do jogo, como músicas e efeitos sonoros, tratando também do acesso ao *hardware* do áudio. É possível usar *software* de terceiros, como a API *Open Audio Library* (*OpenAL*) parente da gráfica OpenGL.

- **Redes e *Multiplayer*:** Jogos podem proporcionar experiência *multiplayer*, significa que o jogo poderá ter mais de uma pessoa jogando em um único mundo virtual.
  Esta experiência pode ocorrer em uma única tela e quando há mais de um dispositivo de entrada conectado a plataforma, de duas maneiras: uma única câmera que mantêm todas as personagens simultaneamente tela ou quando se divide a tela em partes dando uma para cada jogador que terá sua própria visão do jogo.
  Ainda pode ocorrer com o auxílio da rede, conectando cada um dos jogadores através de suas plataformas para criar a experiência *multiplayer online*.
  Dada a importância desta características em alguns gêneros jogos as *engines* podem ter uma camada para facilitar a implementação desta funcionalidade.

Pensando nisso algumas *engines* consideram o jogo com um único jogador um caso especial do *multiplayer* onde há um só jogador, segundo Gregory (2009) transformar uma experiência *multiplayer* em único jogador é normalmente trivial, porém o oposto pode dar muito trabalho.

- **Camadas do jogo:** Seja para o caso da empresa estar criando a própria *engine* ou mesmo para inserir uma nova camada de abstração na arquitetura, podem existir camadas com soluções específicas para o jogo que está sendo feito já que, normalmente as *engines* se especializam em um gênero de jogo.
  Sendo assim, é possível que exista uma nova camada com funções básicas do jogo ou "camada de fundamentos de jogo" (GREGORY 2009) nesta camada são programadas regras que afetarão o jogo como um todo, como tratamento de eventos, física de corpos rígidos (pedras, cadeiras, garrafas, etc.), inteligência artificial (IA), enfim, o que puder ser generalizado no escopo do jogo. É possível usar ferramentas de terceiros para IA, porém reações complexas específicas do jogo devem ser codificadas.

Finalmente, o jogo, a camada que a equipe de desenvolvimento cria, onde todo o conhecimento específico do jogo, do *Game Design Document*, toma vida. Esta camada está acima de todas as outras, ela usará o poder vindo dos componentes para criar o jogo planejado.

2.7 CONSIDERAÇÕES SOBRE O CAPÍTULO

Dada a evolução tecnológica jogos agora podem habitar o mundo virtual através de diferentes plataformas, desenvolver jogos não é um processo trivial, envolve planejamento, ferramentas de suporte e diferentes tipos de profissionais: desenhistas, escritores e programadores, entre outros.

Dentre os desafios enfrentados por cada um destes profissionais, um é importante para esta pesquisa: unir os recursos criados por eles e de fato criar o jogo, este é o papel da programação com a *game engine*.

A programação dá vida a arte criada pelos *designers*, permitindo que o jogador interaja com o jogo, diferindo um jogo de uma animação. A complexidade deste processo, ou seja,

interpretar a entrada do usuário e responder com uma ação na tela, pode ser regulada com uso de uma *game engine*.

Durante o planejamento do jogo são feitas decisões sobre o público alvo, o gênero, o roteiro e também sobre que ferramentas serão utilizadas em sua elaboração, como a *engine* empregada. Devido a existência de vários tipos de *engine* com níveis diferentes de complexidade e propósitos esta escolha pode se tornar complicada, e dada a importância desta ferramenta para o projeto esta escolha deve ser feita com cuidado.

O uso de *game engines* é considerado hoje o estado da arte no desenvolvimento de jogos (FURTADO, 2012), configurando assim a necessidade desta ferramenta no desenvolvimento de jogos, para que este chegue no mercado em tempo. Funcionalidades como renderização de gráficos, tocar música e sons, reconhecer comandos de entrada do usuário, inteligência artificial, colisões e física, podem ser implementados pela *engine,* estando a disposição dos programadores do jogo, aumentando a produtividade na hora de escrever o código do jogo.

Nem sempre as *engines* irão implementar todas as funcionalidades, sendo este mais um agravante na hora da escolha. Na seção 2.6 foi apresentado um modelo de arquitetura criado com base no modelo proposto por Gregory (2009). Esta arquitetura será usada no processo comparativo para a análise das *game engine* escolhidas, já que apresenta as possíveis funções que uma *engine* apresenta para auxiliar o desenvolvedor.

3 ANÁLISE COMPARATIVA

Neste capítulo será abordada a análise comparativa das *game engines* para desenvolvimento de jogos móveis. Devido ao tempo disponível para realização deste trabalho, será feita a análise de três *engines*, de acordo com o processo comparativo proposto. Posteriormente será montada uma tabela comparativa entre as *engines* analisadas, para finalmente eleger dentre elas a que facilita mais o desenvolvimento de jogos.

3.1 PROCESSO DE ANÁLISE COMPARATIVA

A análise comparativa das *engines* será feita com base no modelo arquitetural proposto descrito na seção 2.6. Será investigada a presença dos componentes desta arquitetura na *engine* analisada.

Para verificar a presença dos componentes será feita uma pesquisa na documentação disponível das *engines* e caso a documentação não seja satisfatória, na interface gráfica e código da mesma.

Quanto mais destes componentes a *game engine* tiver, mais soluções de produtividade estarão disponíveis para o desenvolvedor, logo, a *engine* que mais se adequar ao modelo arquitetural, será eleita como a que mais facilita o desenvolvimento.

Caso a *engine* possua o componente, este será descrito como ela o aborda, dessa maneira é possível analisar de que maneiras ela pode ajudar o desenvolvedor, a medida que a presença de mais componentes representa mais abstração, identificando quais soluções a *engine* provê.

Na pré-produção, onde o processo de escolha da *engine* ocorre, o desenvolvedor irá buscar o que ela têm para oferecer e se as soluções condizem com o tipo de jogo que será feito. Tomando como base a arquitetura proposta é possível detectar mais objetivamente qual das *engines* candidatas se adequam mais ao futuro jogo, comparando as possíveis candidatas. Desta forma, o processo pode ser aplicado para comparar quaisquer conjunto de *engines* que o desenvolvedor deseje.

3.2 ANDENGINE

A *AndEngine* é uma *engine* em biblioteca livre[7] para desenvolvimento de jogos 2D, ou seja, ela é um conjunto de classes já implementadas para ajudar o desenvolvedor do jogo (ROGERS, 2012). O projeto pode ser feito com ajuda do ambiente de desenvolvimento Eclipse e com o plug-in ADT[8], que permite a criação e manutenção de projetos *Android*. Com um projeto criado no ambiente basta adicionar a biblioteca ao diretório, é importante notar que a *AndEngine* faz jogos exclusivamente para *Android*.

A figura 14, resume os componentes encontrados na *engine* de acordo com processo proposto:

Figura 14: Componentes da *AndEngine*

3.2.1 Descrevendo os Componentes

Nesta seção será descrita a abordagem da *AndEngine* para os componentes arquiteturais presentes e a justificativa para os não presentes.

[7] O código da AndEngine é aberto, aceitando sugestões e contribuições de quem estiver interessado.
[8] *Android Development Tools*

3.2.1.1 Presentes

- Motor de renderização: Usa a API Multimídia *OpenGL* como renderizador de baixo nível, o *OpenGL* desenha gráficos independente do hardware através das placas de vídeo dos dispositivos que possuem implementação compatível, na *AndEngine* ela está encapsulada, o programador codifica as classes que irão desenhar na tela, e estas que fazem as chamadas a API multimídia.
- Animação: A classe *AnimatedSprite* trata a animação, passando uma imagem em sequencia como a figura 12, indicando a posição dos quadros (cada imagem da personagem da figura), é possível mapear e usando métodos da classe animar o Sprite.
- Dispositivos de Entrada: O próprio SDK do *Android* disponibiliza métodos de capturar eventos de toque na tela, que podem ser usados, porém a *AndEngine* pode criar um simulador de controle na tela do smartphone. Colocando botões translúcidos na tela parecidos com os de um controle de videogame.
- Áudio: Possui duas classes: *Music* e *sound*, para representar músicas e efeitos sonoros, o responsável por tocar o som é o *Android* que recebe e carrega o áudio enviado pela *engine* para ser tocado pelo *hardware* do dispositivo.
- Colisão e Física: Esta função vem em uma biblioteca separada, *AndEngine Physics Box2D Extension*, estendendo as funções da solução de terceiros *Box2D* para o usar no contexto da *AndEngine*. A *Box2D* é uma *engine* feita exclusivamente para física e colisões, que proporciona detecção de colisão e simulação de gravidade.
- Ferramentas de análise e depuração: A classe *FPSLogger* mostra a situação dos *FPS* do jogo no *LogCat* que é uma interface para mostrar mensagens de *logging* no plug-in da IDE.
- Sistemas de núcleo: Como um jogo feito na *AndEngine* é feito exclusivamente para *Android* o qual roda aplicações *java*, todas as classes com os algoritmos que existem no SDK do *java* podem ser usadas.

3.2.1.2 Não Presentes

- Camada de independência de plataforma: A *AndEngine* só faz jogos para *Android.*

- Gerente de Recursos: Apesar do Eclipse providenciar visualização dos recursos, este é feito apenas através da reprodução da estrutura de diretórios do projeto. Como de alguma forma as *engines* devem dispor um gerente de recursos, este será analisado de acordo com a presença de um gerente mais robusto, com filtros de pesquisa, pré-visualização, no sentido de proporcionar uma busca melhor que a do próprio sistema operacional, ou seja, quando a *engine* dispor de uma solução específica para o tratamento dos recursos do projeto. O que não ocorre na *AndEngine*.
- Redes e Multijogador e Inteligência Artificial: Este recurso tem que ser implementado pelo programador. Podendo usar ajuda dos recursos de núcleo.
- Otimização: Na *AndEngine* otimizar o código é uma tarefa do programador, não há alguma ferramenta para isto, o programador deve usar o *FPSLogger* e verificar o desempenho do jogo, e então buscar uma maneira de otimizar cenas pesadas do jogo, removendo excesso de *sprites* de uma cena que perde desempenho, por exemplo.

3.3 Cocos2D

A *Cocos2D*[9] é uma *engine* em biblioteca, como a *AndEngine*. É possível desenvolver jogos para *iOS* e *MacOS* com ela. O ambiente de desenvolvimento é o *Xcode,* ao instalar o pacote da *Cocos* basta adicionar o *template* na lista do *Xcode* e iniciar o projeto.

A figura 15, resume os componentes encontrados na *engine* de acordo com o processo proposto:

[9] *Cocos2D for iPhone*, Disponível em: http://www.cocos2d-iphone.org/wiki/doku.php/prog_guide:index Acesso em: 23/09/2013.

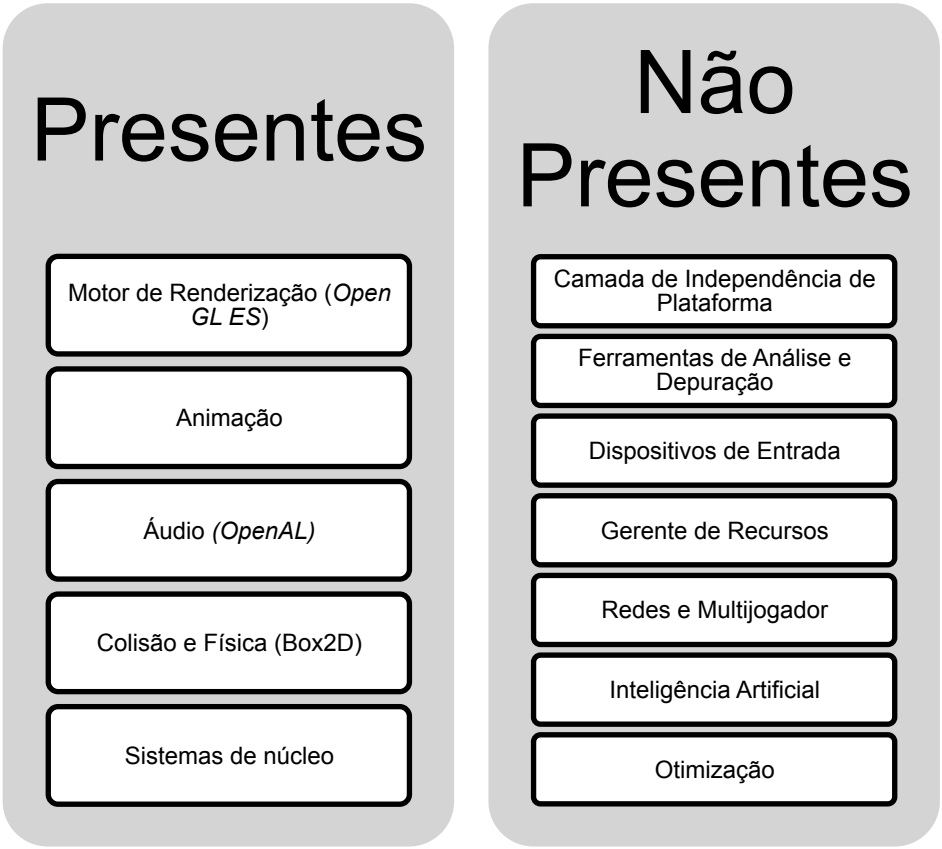

Figura 15: Componentes da *Cocos2D*

3.3.1 Descrevendo os Componentes

Nesta seção será descrita a abordagem da *Cocos2D* para os componentes arquiteturais presentes e a justificativa para os não presentes.

3.3.1.1 Presentes

- Motor de renderização: Usa a API Multimídia *OpenGL ES* como renderizador de baixo nível, o *ES* representa que esta é uma versão específica para hardware móvel, feita para aproveitar o hardware com economia de energia.
- Animação: O suporte a animação é dado pela classe *CCAnimation* que é responsável por reconhecer os recursos animáveis importados e animá-los. A *Cocos* reconhece *sprites* feitos com o *Photoshop* e com *Flash CS4* além do modo de interpretação de quadros, como visto na *AndEngine*.
- Áudio: Possui uma *engine* dedicada ao tratamento de áudio a *CocosDenshion*, esta usa *OpenAL* por baixo para tocar o som.

- Colisão e Física: Assim como a *AndEngine* a *Cocos* usa a *Box2D* para tratar a física.
- Sistemas de núcleo: Classes de *Objective-C* podem ser usadas para apoiar o desenvolvimento, já que esta é a linguagem padrão usada pela *Cocos2D*.

3.3.1.2 Não Presentes

- Camada de independência de plataforma: A *Cocos2D* só faz jogos para *iOS*.
- Ferramentas de análise e depuração: Não foram encontradas ferramentas específicas para análise.
- Dispositivos de Entrada: A captura de eventos de toque é feita automaticamente pela *engine* porém não é possível simular um controle, fica a cargo do desenvolvedor criar o controle e adicioná-lo a tela. É importante notar que a captura de toque é algo inerente ao *SDK* do *smartphone* seja *iOS* ou *Android*. A presença deste componente (dispositivos de entrada) se dá ao notar que a *engine* se esforça em facilitar a captura de eventos de toque que diferem dos já implementados pelos *kits*.
- Gerente de Recursos: Apresenta o mesmo problema descrito no gerente de recursos da *AndEngine* na seção 3.2.1.2.
- Redes e Multijogador e Inteligência Artificial: Este recurso tem que ser implementado pelo programador.
- Otimização: Não há otimização automática, é necessário teste manual para verificar gargalos no jogo e otimizá-los por conta própria.

3.4 Unity

A *Unity*[10] é um ecossistema para o desenvolvimento de jogos para múltiplas plataformas: (1)*Windows*; (2)*MacOS*, (3)*Linux*, (4)*Android*, (5)*iOS*, e (6)Navegadores web. Possui uma versão Pro, com mais recursos, porém a versão analisada neste trabalho será a gratuita. Empresas ou entidades que tiveram menos de US$100,000 de receita podem criar e publicar jogos feitos com a versão gratuita da *Unity*.

A *Unity* possui um conjunto de ferramentas para o desenvolvimento do jogo, o desenvolvedor deve usar o ecossistema *Unity* para criar ou importar os objetos do jogo, como:

[10] *Unity – Game Engine*, *Tools and multiplataform*, Disponível em: http://unity3d.com, Acesso em: 05/07/2013.

modelos gráficos 3D, figuras, áudio, e programar estes objetos através de C#, *Javascript* ou *Boo*, que permitem a chamada de funções escritas nas linguagens da plataforma fim.

Dotada de um editor *WYSIWYG* a *engine* procura mostrar na tela como o jogo será enquanto ele está sendo criado na tela do ambiente de desenvolvimento. Cada componente possui uma interface gráfica de onde pode ser controlado.

3.4.1 Descrevendo os Componentes

A *Unity* apresenta todos os componentes do modelo arquitetural proposto, nesta seção será descrita sua abordagem para estes componentes.

- Motor de renderização: O renderizador de baixo nível da *Unity* para dispositivos móveis é o *OpenGL ES,* uma API multimídia específica para hardware de plataformas móveis. A *Unity* renderiza jogos 3D e possui componentes para tratar: câmera, objetos 3D e texturas, efeitos visuais: (1) partículas, (2) iluminação e (3) sombras, otimização gráfica e interface gráfica 2D para *GUI*.
- Animação: Uma das visões da *Unity* é a de animação onde o desenvolvedor pode criar animações através de *scripts*, usar animações pré-carregadas para animação de humanoides através de mapeamento de esqueletos, criar, salvar e importar animações de esqueletos.
- Dispositivos de Entrada: Reconhece as entradas de toque do *Android* e *iOS,* com simulação de controle.
- Áudio: Através dos componentes *Audio Listener* que pode ser adicionado a câmera para captar o som tocado em uma determinada posição no ambiente 3D e *Audio Source* que pode ser adicionado a um objeto do jogo para que ele toque um som dado um evento.
- Colisão e Física: Usa a *engine PhysiX*. Para detectar colisão utiliza-se os *Colliders* que possuem formas padrão como esferas, cubos e *Meshs,* este último é o nome que a *Unity* dá para formas específicas de objetos que o desenvolvedor personaliza, as três servem para mapear a forma do objeto e criar uma área que detecta colisão.
- Ferramentas de análise e depuração: Dentro das visões do ambiente de desenvolvimento *Unity* existem *Managers* que são responsáveis por alterar variáveis dos recursos, por exemplo no *Physics Manager* é possível alterar a variável que controla a velocidade da gravidade. Ela possui também o *Unity Profiler* que analisa o

jogo como um todo e verifica onde estão os gargalos, ou seja, os locais onde se exige mais do *hardware*.

- Sistemas de núcleo: Os algoritmos e estruturas de dados disponíveis em C#, *Javascript* e *Boo* e específicas para as plataformas que a *Unity* pode criar jogos.
- Camada de independência de plataforma: A *Unity* pode gerar jogos para diferentes plataformas: (1)*Windows*; (2)*MacOS*, (3)*Linux*, (4)*Android*, (5)*iOS*, e (6)Navegadores web, logo este componente está presente.
- Gerente de Recursos: A *Unity* separa os recursos em um navegador de diretórios, além disso disponibiliza busca e pré-visualização de todos tipos de recursos, sejam eles objetos 3D, imagens, sons ou código, separando e filtrando resultados, na visão de Projetos.
- Redes e Multijogador: A *Unity* possui métodos para tratar conexão com servidores, chamadas de procedimentos remotos, clientes do jogo, sincronia dos objetos no cliente com o servidor e compartilhamento de dados. E para multijogador, basta adicionar uma nova câmera na cena, colocando a posição corretamente dividindo a tela.
- Inteligência Artificial: Através do *Navmesh* é possível padronizar comportamento dos objetos 3D, traçando rotas de movimentação pelo cenário. Ao designar um objeto com o padrão obstáculo outros agentes (objetos animados) irão evitar este objeto, por exemplo.

3.5 Tabela Comparativa

A tabela lista os componentes presentes nas *engines* analisadas, com o objetivo de comparar as funcionalidades presentes de acordo com o processo proposto, além de reunir as informações da análise para rápida compreensão de forma resumida.

As células em verde indicam a presença do componente, as em branco a ausência.

Tabela 3.1: Tabela comparativa das *engines*.

	Cocos2D	*AndEngine*	*Unity*
Motor de Renderização			
Animação			
Áudio			
Colisão e Física			
Sistemas de núcleo			
Dispositivos de Entrada			
Análise e Depuração			

Independência de Plataforma			
Gerente de Recursos			
Redes e Multijogador			
IA			
Otimização			

A *Unity* mostra completa adequação à arquitetura proposta, sendo assim, dentro do processo de análise proposto e dentre as comparadas, demonstra-se como a *engine* que mais facilita o desenvolvimento de jogos. Desta forma, possui todos os atrativos descritos na figura 11, proporcionando maior produtividade para os desenvolvedores.

4 CONCLUSÃO

4.1 CONCLUSÃO

O desenvolvimento deste trabalho teve por objetivo apresentar uma maneira de analisar *game engines* para auxiliar o desenvolvedor na escolha desta ferramenta tão importante na criação de jogos digitais.

Ao descrever o processo de criação de jogos ficou claro a importância das *engines* no processo de desenvolvimento dos jogos, e ao mostrar a evolução das ferramentas de apoio ao desenvolvimento foi possível verificar que as *game engines* são o estado da arte no desenvolvimento de jogos digitais.

Para analisar estas ferramentas foi proposto um processo que verifica a presença de componentes que dão suporte ao desenvolvedor, estes componentes foram retirados do modelo arquitetural proposto, que busca generalizar os possíveis componentes de uma *engine.*

Foi notado durante o desenvolvimento deste trabalho que o processo é melhor aproveitado caso as *engines* tenham uma boa documentação, possibilitando uma rápida compreensão de suas funcionalidades. Caso seja necessário analisar código para desvendar as funções a análise pode tomar tempo, o que pode causar um atraso no ciclo de vida do desenvolvimento, prejudicando o tempo de mercado.

Das três *engines* escolhidas uma delas destacou-se já que possui todos os componentes propostos, a *Unity*, dentro do processo proposto é eleita como a que tem potencial para dar mais suporte ao desenvolvedor. Ainda assim, a versão gratuita da *Unity* tem limitação quanto a receita anual da empresa que não pode ser superior a US$100,000, sendo superior é necessário comprar a licença de US$1,500[11].

Porém, não apenas a eleição de uma *engine* é contribuição para o objetivo geral deste trabalho, há também importância no processo comparativo proposto, que por si só tem potencial para auxiliar os desenvolvedores. Através deste desenvolvedores podem elencar possíveis *engines* e construir a tabela comparativa para escolher uma entre elas, demonstrando sua replicabilidade.

[11] Valor vigente na data de publicação deste trabalho.

4.2 SUGESTÕES DE TRABALHOS FUTUROS

Futuramente é possível expandir a análise para um número maior de *engines,* incluindo *aquelas* que desenvolvem outros jogos digitais que não os para plataformas móveis.

Outra contribuição futura seria a possibilidade de inserir outros componentes arquiteturais a serem analisados, ao verificar a presença de possíveis componentes novos não listados. Seria também interessante eleger peso para os componentes, dando importância maior a algum componente, como uma forma de dar mais precisão ao resultado da análise.

Finalmente, vejo como possível a criação de um métrica a fim de medir o quão compatível a *engine* é ao modelo arquitetural proposto.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAUJO, A. ***AGP: Agile Game Process***, Trabalho de Conclusão de Curso (Ciências da Computação) - Universidade Federal de Pernambuco, 2006.

AZEVEDO, Eduardo. **Desenvolvimento de Jogos 3D e Aplicações em Realidade Virtual.** Rio de Janeiro: Elsevier, 2005. 319 p.

BARROS, R. **Análise de Metodologias de Desenvolvimento de Software aplicadas ao Desenvolvimento de Jogos Eletrônicos**, Trabalho de Conclusão de Curso (Ciências da Computação) - Universidade Federal de Pernambuco, 2007.

BARROS, Rui. **Jogos 3D em tempo real para iPhone / iPad baseados em sensores.** Dissertação de Mestrado (Engenharia Informática e Computação) – FEUP, Faculdade de Engenharia da Universidade do Porto, 2001.

CAPTURA DE MOVIMENTOS. ***Activemaker2.png*** Disponível em: < http://upload.wikimedia.org/wikipedia/commons/6/6d/Activemarker2.PNG> Acesso em: 24 set. 2013.

CHRONO TRIGGER. **Screenshot Chrono Trigger – Super Nintendo.** 2013. Disponível em: <*http://pt.wikipedia.org/wiki/Ficheiro:ChronoTrigger_battle.jpg*>. Acesso em: 05 mai. 2013.

CLUA, E. W. G.; BITTENCOURT J. R. **Desenvolvimento de Jogos 3D: Concepção, Design e Programação.** Disponível em: <http://www.unisinos.br/_diversos/congresso/sbc20-05/_dados/anais/pdf/arq0286.pdf>. Acesso em: 02 mai. 2013.

CRAWFORD, C. ***The Art of Computer Game Design*.** 1982. Disponível em: <http://www-rohan.sdsu.edu/~stewart/cs583/ACGD_ArtComputerGameDesign_ChrisCrawford_1982.pdf> . Acesso em: 02 mai. 2013.

CURTI, M. M. **Conceitos e Tecnologias no Desenvolvimento de Jogos Eletrônicos.** Trabalho de Conclusão do Curso (Sistemas de Informação) – UNIFEV, Centro Universitário de Votuporanga, Votuporanga, 2006. 84p.

EXEMPLO DE SPRITE. ***Various sprites sizes vs masks*.** Disponível em: < http://gmc.yoyogames.com/index.php?showtopic=555226> Acesso em: 24 set. 2013.

FINAL FANTASY TACTICS ADVANCE. ***Screenshot Final Fantasy Tactics Advance – Gameboy Advance*.** 2013. Disponível em: < http://img.gamefaqs.net/screens/9/0/e/gfs_4490-8_2_8.jpg>. Acesso em: 05 mai. 2013.

FLYNT, J. ***Software Engineering for Game Developers***. Course Technology 2005.

FURTADO, A. ***Domain-Specific Game Development*.** Recife: O Autor, 2012. 233 p.

FURTADO, A.; SANTOS A. **FunGEn:** ***A Game Engine for Haskell***, 1st *Brazilian Workshop in Games and Digital Entertainment* (Wjogos2002), 2002.

GREGORY, Jason. ***Game Engine Architecture.*** Wellesley, Massachusetts: A K Peters, 2009.

HUIZINGA, Johan. **Homo Ludens: o jogo como elemento cultural.** São Paulo: Perspectiva, 1971. 243 p.

INFINITY BLADE 2. ***Screenshot Infinity blade 2 – iOS.*** 2013. Disponível em: < http://-infinitybladegame.com/infinityblade2/>. Acesso em: 05 mai. 2013.

MADEIRA, C. **FORGE V8: Um *framework* para o desenvolvimento de jogos de computador e aplicações *multimedia***, Disponível em: <http://charles.madeira.free.fr/publications/MadeiraMasterThesis2001.pdf> Acesso em: 07 mai. 2013.

MEGAMAN X8. ***Screenshot Megaman* X8 – *Playstation* 2.** 2013. Disponível em: < http://www.mobygames.com/game/windows/mega-man-x8/screenshots/gameShotId,292468/ >. Acesso em: 05 mai. 2013.

N.O.V.A. ***Screenshot* N.O.V.A – *Android.*** 2013. Disponível em: <*http://br.gameloft.com/jogos-android/nova/*>. Acesso em: 05 mai. 2013.

PARLETT, David. ***The Oxford history of board games.*** *Oxford University Press*, 1999. 386 p.

PERUCIA, A. S. et al. **Desenvolvimento de Jogos Eletrônicos: Teoria e Prática.** São Paulo: Novatec, 2005. 320 p.

ROGERS, R. ***Learning Android Game Programming.*** United States, Indiana: Pearson, 2012. 429 p.

ROLLINGS, A.; MORRIS, D. ***Game Architecture and Design***. *The Coriolis Group*, 2000. 7-42 p.

SALEN, Katie. **Regras do Jogo: fundamentos do design de jogos: principais conceitos: volume 1** / Katie Salen e Eric Zimmerman; [tradução Edson Furmankiewicz]. São Paulo: Blucher, 2012. Título Original: Rules of play: game design fundamentals.

SLOPER, Tom. ***Following Up After the Game is Released: It's not Over when it's Over***. Game Design Perspectives. 2002.

TAKANO, Rafael. **Motor de jogos 3D para iphone OS.** Trabalho de Conclusão de Curso (Ciências da Computação) – Universidade Regional de Blumenau, Blumenau 2009.

WARD, Jeff. ***What is a game engine?*** [S.l.], 2008. Disponível em: <http://www.gamecareerguide.com/features/529/what_is_a_game_.php?page=1>. Acesso em: 02 mai. 2013.

ZORK. **Screenshot Zork – PC.** 2013. Disponível em: <http://www.mobygames.com/game/dos/zork-ii-the-wizard-of-frobozz/screenshots/gameShotId,398361/>. Acesso em: 05 mai. 2013.