FUNDAÇÃO UNIVERSIDADE DO TOCANTINS
SISTEMAS DE INFORMAÇÃO

ATILA ALVES PACHECO

ESTUDO COMPARATIVO ENTRE OS MOTORES *UNITY* E *UNREAL* PARA O DESENVOLVIMENTO DE JOGOS 2D DESTINADOS A PLATAFORMA ANDROID

PALMAS – TO
2017

ATILA ALVES PACHECO

ESTUDO COMPARATIVO ENTRE OS MOTORES *UNITY* E *UNREAL* PARA O DESENVOLVIMENTO DE JOGOS 2D DESTINADOS A PLATAFORMA ANDROID

Trabalho apresentado ao Curso de Sistemas de Informação da Fundação Universidade do Tocantins - UNITINS como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação.

PALMAS – TO

2017

ATILA ALVES PACHECO

ESTUDO COMPARATIVO ENTRE OS MOTORES *UNITY* E *UNREAL* PARA O DESENVOLVIMENTO DE JOGOS 2D DESTINADOS A PLATAFORMA ANDROID

Trabalho apresentado ao Curso de Sistemas de Informação da Fundação Universidade do Tocantins - UNITINS como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Me. Silvano Maneck Malfatti

COMISSÃO EXAMINADORA

_____

Prof. Silvano Maneck Malfatti.

_____

Prof. Arlenes Delabary Spada.

_____

Prof. Frederico Pires Pinto.

DEDICATÓRIA

Para a minha família e amigos**.**

AGRADECIMENTOS

Quero agradecer primeiramente a Deus, que me concedeu a oportunidade de concluir este trabalho através de força e superação das dificuldades. Agradeço também ao meu orientador que concedeu parte do seu tempo e conhecimento. Agradeço também a todos os professores da Unitins porque fizeram parte da minha formação como profissional. Agradeço aos familiares, amigos e todos os que fizeram parte deste sonho, seja direta e indiretamente.

EPÍGRAFE

"É no brincar, e somente no brincar, que o indivíduo, criança ou adulto pode ser criativo e utilizar sua personalidade integral: e é somente sendo criativo que o indivíduo descobre o eu" **(WINNICOTT, 1971)**

RESUMO

O processo de desenvolvimento de jogo é complexo, envolve programadores, roteiristas, gerentes de projetos, artistas gráficos entre outros profissionais. Para projetos com uma equipe limitada é necessário o uso de ferramentas, bibliotecas ou APIs que facilitem o trabalho permitindo o foco na aplicação. Um Motor de jogo permite ao desenvolvedor criar aplicações aproveitando-se de códigos já existentes, reduzindo o número de módulos desenvolvidos pelos programadores. Nesse contexto, este trabalho tem como objetivo comparar dois dos principais motores de jogos no desenvolvimento de aplicações para dispositivos móveis com Android e demonstrar os impactos gerados pela escolha de um motor específico na aplicação final. A partir do desenvolvimento do trabalho foi possível demonstrar os impactos que cada motor gera no desenvolvimento de jogos 2D para smartphones com Android.

PALAVRAS CHAVE: Jogo, APIs, Motor de jogo, Android, Impactos.

ABSTRACT

The game development process is complex, involves, programmers, scriptwriters, project managers, graphic artists and other professionals. For projects with a limited team it’s necessary to use tools, libraries or API’s that facilitate the work allowing the focus on the application. A game engine allows to the developer create applications using codes that already exists, reducing the number of modules developed by the programmers. In this contexto this wor proposes analyze and compare two of the main game engines in the developement of applications for mobiles with Android and demonstrate the impacts generated by choose of a specific engine in the final aplication. Through the development of the work it was possible to demonstrate the impacts that each engine generates in the development of 2D games for Android smartphones.

KEYWORDS: Game, APIs, Game Engine, Android, Impacts.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O mercado de jogos tem ganhado cada vez mais importância econômica. Atualmente, apresenta um forte crescimento na quantidade e na qualidade dos aplicativos. Nesse contexto, o Brasil se tornou o quarto maior mercado consumidor na área em 2012, representando 35 milhões de usuários que movimentaram cerca de 5,3 bilhões de reais no ano de 2012. Além disso, se comparado ao ano anterior o mercado cresceu 32% (SEBRAE, 2014).

Em destaque está a indústria de jogos para dispositivos móveis. Segundo uma pesquisa da empresa Superdata (http://www.superdataresearch.com/blog/latin-american-digital-games-market-hits-4-5b/), os jogos para dispositivos móveis representaram 43% do mercado total para jogos na América Latina em 2014. Tal resultado poderia ser atribuído à grande quantidade de *smartphones* em utilização nesses países. Como a demanda pelo desenvolvimento de jogos é alta, empresas tem se empenhado no desenvolvimento de ferramentas que possibilitem criar projetos de qualidade em pouco tempo, esta ferramenta é denominada de motor de jogo.

Um Motor de jogo é utilizado para facilitar o desenvolvimento de jogos, isto é, um motor de jogo consiste em uma API de alto nível para as complicadas APIs gráficas. Com diversas ferramentas e funcionalidades permite ao desenvolvedor criar mais rapidamente seu projeto sem se preocupar em desenvolver todos os módulos (GREGORY, 2009). Utilizar esse tipo de ferramenta possibilita ao desenvolvedor à utilização de módulos para simular física básica, por exemplo, a aplicação da força da gravidade em um determinado objeto e até a renderização dos gráficos. Nesse contexto, a utilização de um motor de jogo define o quanto deverá ser criado ante ao que se poderá aproveitar pelo mesmo.

Com a necessidade de se acelerar o desenvolvimento de aplicações para dispositivos móveis que necessitam de boa qualidade gráfica, a *Epic Games* lançou em março de 2014 a *Unreal Engine* 4, um motor de jogos (*game engine*), que na sua quarta versão começou a trabalhar com consoles e sistemas embarcados como smartphone e tablet. Com o grande crescimento da indústria de jogos para *smarthphones* a *Unreal* acrescentou no seu motor de jogo uma implementação do *OpenglES*, uma API (*Application Programming Interface*) - termo utilizado para classificar um conjunto de normas ou instruções de funções específicas que permitem a criação de aplicações,

dando condições ao desenvolvedor de utilizar os recursos da GPU (*Graphics Processing Unit*) para aprimorar as qualidades gráficas dos jogos e melhorar o desempenho.

A Unity ou Unity 3D é uma ferramenta para o desenvolvimento de jogos e tem ganhado grande espaço no desenvolvimento de jogos para diversas plataformas, entre elas os sistemas embarcados (MORIBE, 2012). A Unity 3D em sua versão gratuita permite ao desenvolvedor criar aplicações para Windows, Mac, Android e Web, utilizando das linguagens de programação C# ou JavaScript. Assim, surge a pergunta: qual ferramenta possibilita ao desenvolvedor trabalhar com maior produtividade quando o projeto não possui uma equipe grande e os recursos são limitados?

O Android é um sistema operacional móvel baseado em Linux desenvolvido pela Android INC, que posteriormente foi comprado pela Google e atualmente segue mantido por um consórcio de empresas. De acordo com dados da *Net Applications*, o sistema operacional Android tornou-se o líder de uso na área móvel ultrapassando o até então líder iOS (NETMARKETSHARE, 2014). Neste contexto surge a necessidade de um estudo que compare as duas ferramentas mais populares atualmente para o desenvolvimento de jogos para dispositivos móveis que utilizam o sistema operacional Android.

Com a evolução dos dispositivos móveis e *softwares* especializados para o desenvolvimento de aplicações voltadas aos sistemas embarcados, a escolha de uma ferramenta que possibilite uma maior produtividade e desempenho se torna essencial. Para o desenvolvimento de aplicações do tipo jogos, a escolha de um determinado motor de jogo implicará no quanto o desenvolvedor terá de implementar e o quanto poderá ser aproveitado de códigos já existentes.

O processo de desenvolvimento de jogos requer uma equipe multidisciplinar qualificada e, preferencialmente experiente. Essas atividades necessitam de artistas/designers, engenheiros de software, programadores, pessoas com conhecimentos específicos nas áreas de física, álgebra linear e geométrica além de softwares especializados para transformar uma ideia no produto final. Os motores de jogos permitem às equipes ou desenvolvedores e programadores trabalharem com todos esses conceitos e atividades de maneira mais simplificada, permitindo então, o foco no desenvolvimento do produto em si. Quais os impactos no projeto e no desempenho do jogo um motor pode se diferenciar do outro? Qual das duas *engines*, *Unity* ou *Unreal*, promove mais produtividade no desenvolvimento de uma determinada aplicação do tipo

“jogos” para o sistema operacional Android? Estas são as perguntas que este trabalho responderá.

1.1 OBJETIVO GERAL

Este trabalho apresenta um estudo comparativo de motores de jogos através da implementação de um jogo utilizando os motores *Unreal Engine 4* e *Unity3D*. O presente trabalho tem como objetivo comparar dois motores no desenvolvimento de jogos 2D para Android e demonstrar os impactos gerados no projeto e na aplicação.

1.2 OBJETIVOS ESPECIFICOS

- Estudar a documentação disponibilizada por cada empresa desenvolvedora do motor de jogo.
- Desenvolver um jogo utilizando os motores da *Unity* e *Unreal*.
- Comparar a quantidade de recursos, custo e produtividade.
- Mensurar parâmetros de renderização (fps), tamanho do aplicativo e consumo de bateria.

2 REFERENCIAL TEÓRICO

2.1 O QUE É UM JOGO?

Segundo Koster (2013) o jogo é uma experiência interativa que oferece ao jogador padrões cada vez mais desafiadores para que ele aprenda e eventualmente ganhe maestria. Para Huizinga (2003) conceitualiza o jogo como uma atividade lúdica com regras que não se restringe ao ser humano. Ainda segundo Huizinga (2003) o jogo humano em sua forma mais simples consiste em uma atividade com regras que são respeitadas pelos participantes e que proporciona imenso prazer e divertimento.

Segundo Huizinga (2003) o jogo surgiu antes que a cultura, pois pressupõe em suas definições menos rigorosas sempre uma sociedade humana, entretanto, é possível reconhecer características de jogos praticados por animais. O autor exemplifica essa noção citando os cachorros, que através de brincadeiras convidam-se para um ritual de atitudes e gestos, respeitando algumas regras, sendo então possível reconhecer elementos do jogo humano.

Nesse contexto, define-se o jogo como uma atividade voluntária maior do que um fenômeno fisiológico ou reflexo psicológico, ultrapassando assim, os limites da realidade física (HUIZINGA, 2003).

Frente a isso, é importante que sejam identificados os elementos fundamentais de um jogo, pois consistem na parte mínima que deve ser aplicada no seu desenvolvimento sendo o tema a ser considera a seguir.

2.1.1 ELEMENTOS FUNDAMENTAIS DOS JOGOS

Para que uma determinada atividade seja classificada como jogo é necessário que ela siga alguns elementos fundamentais (CRAWFORD, 1982). Esse autor define quatro elementos fundamentais presentes em todos os jogos: representação, interação, conflito e segurança, definidos a seguir (1982, p.7-12):

- **Representação:** o jogo é um sistema formal que representa subjetivamente um subconjunto da realidade. Sendo completo, autossuficiente e independente de referências externas. Os jogos apresentam um conjunto de regras explícitas, representadas em um sistema com agentes que interagem

entre si muitas vezes de forma complexa.

- **Interação:** O elemento interativo torna os jogos uma representação da realidade, diferente dos filmes, o espectador se torna agente de mudanças e consequentemente pode alterar a realidade.
- **Conflito:** o conflito surge naturalmente a partir das interações. O jogador procura cumprir um objetivo, obstáculos são impostos para impedi-lo de facilmente atingir essa meta.
- **Segurança:** o jogo é uma forma segura de experimentar a realidade. Conflito implica em perigo, perigo por sua vez implica em risco de dano e o dano é indesejável pelo jogador. Em jogos de computador, por exemplo, a situação de conflitos e perigos são psicológicos o que permite desassociar as consequências das ações. O risco pode ser associado à penalidade ou ausência de recompensas.

Na visão de Crawford (1982), esses conceitos caracterizam todos os tipos de jogos eletrônicos. Os elementos como os cenários, gráficos, *hardware* e motor de jogo são características específicas que delimitam o formato do jogo eletrônico.

2.1.2 JOGOS ELETRÔNICOS

Segundo Mendes (2006) os jogos eletrônicos são *softwares* que dependem de um hardware para funcionar, havendo uma relação consistente, linear e recíproca entre eles. Ainda segundo o autor quanto maior a necessidade de transmitir uma sensação de "realidade" se faz necessário um *hardware* com componentes mais potentes.

Os jogos eletrônicos são artefatos de fascínio econômico, tecnológico e social (MENDES, 2006). Isso se deve a vários fatores. A demanda por jogos com gráficos cada vez mais realistas contribuiu para a evolução dos consoles. Nesse contexto a evolução dos jogos eletrônicos é dada através da evolução dos hardwares. A indústria dos jogos eletrônicos movimenta bilhões de dólares no mundo. Os lançamentos de jogos alcançaram números expressivos de público possibilitando a comparação com grandes lançamentos da indústria cinematográfica.

2.2 POR QUE AS PESSOAS JOGAM?

Os jogos são mecanismos de educação, entretenimento e motivação (CRAWFORD, 1982). Jogos educacionais computadorizados são softwares que apresentam conteúdos e atividades práticas com objetivos educacionais baseados no lazer e na diversão (FALKEMBACH, 2006). Até os jogos criados com o intuito de entreter propiciam um desenvolvimento de diversas áreas como: área cognitiva, afetiva, linguística, motora e criativa. A motivação está ligada à estímulos que fazem com que o indivíduo jogue, características que tornam os jogos atrativos.

Segundo Prensky (2005) as características intrínsecas dos jogos eletrônicos que motivam os seres humanos a jogarem são:

1. Os jogos são uma forma de diversão.
2. Os jogos possuem regras.
3. Os jogos possuem metas.
4. Os jogos são interativos.
5. Os jogos, em geral, podem informar detalhes sobre o desempenho do jogador. Além da capacidade de adaptação do jogo as habilidades do jogador.
6. Os jogos são adaptativos.
7. Os jogos possuem conflitos, desafios, competições e oposições.
8. Os jogos possuem resolução de problemas.
9. Os jogos possibilitam a interação entre os jogadores.
10. Os jogos possuem história.

Cada característica mencionada pelo autor representa uma emoção diferente para o jogador, assim, cada fator contribui para a imersão do ser humano na realidade criada pelos desenvolvedores. As regras determinam o comportamento do jogador, metas remetem a recompensas, resoluções de problemas estimulam a criatividade, interação entre jogadores permite a formação de grupos sociais e a história transmite a emoção dos personagens.

2.3 MOTORES DE JOGOS

Um jogo eletrônico pode ser desenvolvido do zero ou aproveitando-se de códigos existentes para implementar partes essenciais como detecção de colisão,

renderização, tratamento de áudio entre outros. Nesse contexto um motor de jogo pode ser definido como uma série de módulos e interfaces que permitem ao desenvolvedor manter o foco na jogabilidade do produto, reduzindo o trabalho do conteúdo técnico (GOLD, 2004).

O termo motor de jogo (*game engine)* surgiu em por volta de 1990 em referência aos jogos de tiro em primeira pessoa conhecidos como *FPS* (*first-person shooter*), termo utilizado para definir os jogos em que a visão do jogador simula a visão do personagem dentro do jogo, tornando-se conhecido através do Doom, uma *engine* da idSoftware (GREGORY, 2009). Com a popularização do termo o motor de jogo se transformou em uma das principais ferramentas para o desenvolvimento de jogos. A evolução dos dispositivos móveis também atraiu os desenvolvedores para a produção de jogos para sistemas embarcados, visando este novo mercado alguns motores de jogo começaram a trabalhar com *smartphones* e tablet, como por exemplo, os motores *Unity* e *Unreal*, que na próxima seção será apresentado sua arquitetura básica.

2.3.1 ARQUITETURA BÁSICA PARA UM MOTOR DE JOGOS

Um motor de jogo é constituído de duas partes principais: um conjunto de ferramentas de programação (APIs) e um ambiente de tempo de execução (*runtime*) (GREGORY, 2009). A API nada mais é do que um código escrito por terceiros que implementa funções básicas requeridas por um jogo. O *runtime* proporciona os serviços de software para os processos implementados no código do jogo via chamada de métodos específicos da API associada. Portanto, um motor de jogo se apresenta como um extenso pacote de código onde as funcionalidades comuns à maioria das aplicações desse gênero estão codificadas em uma determinada linguagem de programação. A Figura 1 demonstra os principais componentes de tempo de execução que pode ser encontrado em todo motor de jogo 3D.

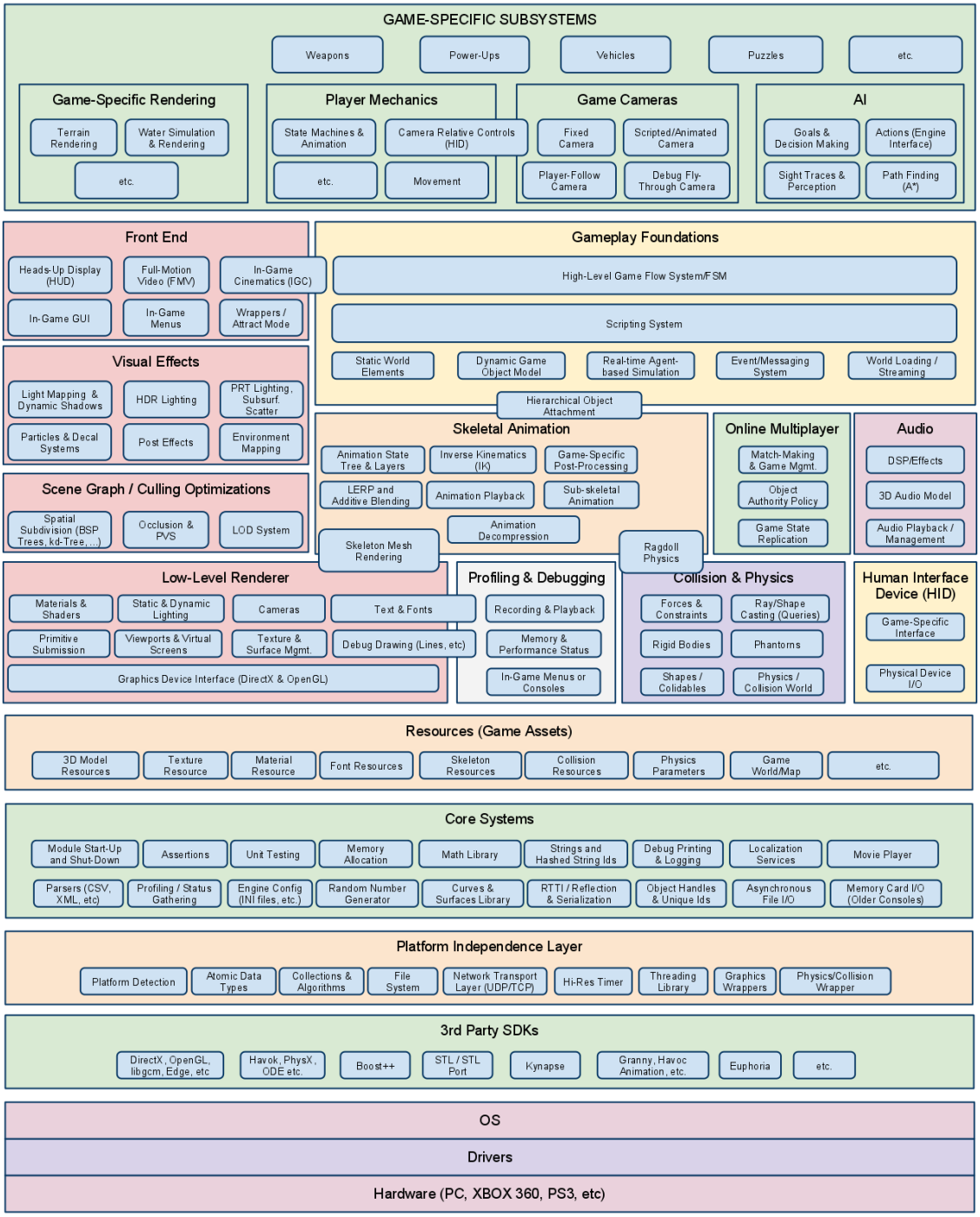

Figura 1 - arquitetura do motor de jogo em tempo de execução.

Fonte: http://www.bennychen.cn/wp-content/uploads/2011/03/RuntimeGameEngineArchitecture.png

Como demonstra a Figura 1 os motores de jogo são construídos em camadas. Como na maioria dos *softwares* as camadas superiores dependem de camadas inferiores, porém, camadas inferiores não dependem das camadas superiores (GREGORY, 2009).

Na camada de hardware se encontra os dispositivos disponíveis para que o motor de jogo crie aplicações. Para este trabalho o hardware selecionado são os dispositivos móveis com Android.

A camada *3rd Party SDKs* contém as APIs e ferramentas que possuem acesso aos recursos da GPU, bibliotecas que tem a finalidade de aumentar o desempenho para a renderização dos gráficos 3D. Ainda segundo Gregory (2009) essa camada possui ferramentas que trabalham com a estrutura de dados da aplicação, animação de personagens, movimentação de personagens, tratamento de colisão e física.

Grandes empresas como, por exemplo, a Blizzard e Ubisoft trabalham com várias plataformas, a camada *Platform Independece Layer* garante um comportamento consistente em todas as plataformas de hardware (GREGORY, 2009). O jogo HearthStone da Blizzard, por exemplo, permite com que os jogadores utilizem tanto o computador quanto os smartphones para jogar com outras pessoas.

A camada de *Core Systems* contém componentes de verificação de erros nas linhas de código, sistema para o gerenciamento de memória, bibliotecas matemáticas e um conjunto de ferramentas para o gerenciamento da estrutura de dados personalizada, caso o desenvolvedor tenha a necessidade de modificar a estrutura para buscar alto desempenho, ou mesmo, caso não confie totalmente em pacotes prontos para tratar a estrutura de dados.

A camada *Resource (Game Assets)* fornece uma interface unificada para o acesso aos recursos do jogo, por exemplo, as texturas utilizadas no projeto, modelos 3D, mapas, personagens e etc. Todos os recursos como trilha sonora, componentes gráficos, por exemplo, personagens, mapas e imagens são acessadas através dessa interface.

O motor de renderização é um dos componentes mais complexos de qualquer motor de jogo (GREGORY, 2009). Na camada *Low-level Renderer* o motor trabalha com um conjunto de primitivas geométricas o quão rápido possível para montar todas as partes de uma cena.

O processador de baixo nível desenha toda geometria que lhe é submetida, normalmente é necessário um componente para limitar o número de primitivas apresentadas ao processador, por exemplo, um componente que limita o campo a ser renderizado processando somente o que for visualizado pela câmera, assim, melhora o desempenho e evita a sobrecarga do processador. Este componente é representado pela camada *Scene Graph/Culling Optimizations*. Muitos jogos utilizam de recursos como

draw distance que limita o campo de visão do jogador, para reduzir o processamento, é renderizado apenas uma parte do mapa e o restante preenchido com uma neblina.

Os motores de jogos modernos possuem uma ampla gama de efeitos visuais como:

- **Sistema de partículas**: efeitos que simulam fumaça, fogo, água.
- **Sistema de decalque**: efeitos para os passos, buracos de projetis.
- **Sistema de mapeamento**: efeitos de luz sobre determinado objeto ou superfície.
- **Sistema de sombreamento**: torna as sombras dinâmicas.
- **Efeitos após tela cheia**: aplicado após as cenas 3D serem carregadas no *buffer* fora do campo visual.

Estes efeitos estão na camada *Visual Efects*, *game engine* possui um sistema de gerenciamento de efeitos, verificando a necessidade de processamento de partículas especializadas, decalques entre outros efeitos visuais.

A camada de *Front End* possibilita a utilização de gráficos em 2D sobrepostos as cenas em 3D, por exemplo, os menus. Para os jogos de RPG, por exemplo, é representado pela manipulação do inventário e configurações de batalha. Alguns jogos utilizam a sobreposição sem interferir na renderização da cena em 3D, por exemplo, o jogo DarkSouls permite alterações na cena enquanto o menu está sobrepondo a cena.

Os jogos são sistemas de tempo real, para não causar problemas de desempenho *game engineers* traçam um perfil de desempenho de seus jogos, a fim de aperfeiçoar o desempenho. A camada de *Profiling & Debbuging* inclui estas ferramentas e recursos de depuração do jogo, além de ter a capacidade de gravar e reproduzir a jogabilidade para testes.

A detecção de colisão é um elemento crucial para qualquer jogo. Na camada *Collision and Physics* além da detecção de colisão ela possui um sistema para tornar os jogos mais realistas. Física e colisão estão fortemente ligados, por exemplo, no jogo de futebol quando o jogador chuta uma bola a colisão detecta que algo a atingiu e aplica uma série de transformações para projetar a bola em uma determinada direção, com uma velocidade e trajetória. Alguns jogos utilizam o termo *Hitbox* que delimita uma área de colisão para determinado objeto, assim, de acordo com a *Hitbox* determina-se se houve colisão com outros objetos ou não.

Sem esses dois componentes os objetos se penetrariam impossibilitando qualquer interação. Não existe jogo sem interação, a camada *Collision and Physics* facilita o trabalho do desenvolvedor, uma vez que, ela tem várias funcionalidades prontas como, por exemplo, aplicar a força da gravidade em determinado objeto.

Todo jogo tem personagens, humanos ou não, todo personagem precisa de um sistema de animação. Gregory (2009) afirma que existem cinco tipos básicos de animação usados em jogos:

- **Sprite**: clássica forma de animação, que utiliza de um conjunto de imagens para transmitir a sensação da movimentação de um personagem.
- **Animação hierárquica de um corpo rígido**: corpos rígidos não sofrem transformações durante a colisão.
- **Animação esquelética**: consiste em texturas carregadas sobre um conjunto de ossos interligados. Utilizada geralmente para dar animação a seres humanos, personagens orgânicos.
- **Animação por vértice**: técnica em que o artista cria diversas cópias do modelo original, e cada cópia é um quadro. A movimentação do personagem é dada através da movimentação dos vértices.
- **Animação de transformação**: quando uma imagem se sobrepõe a outra causando uma transformação.

A animação esquelética permite um detalhamento através de uma malha 3D sobre um conjunto de ossos. Apesar da animação por vértice e por transformação serem bastante utilizadas em alguns motores de jogo, a animação esquelética é a mais utilizada nos jogos atuais. Neste contexto a animação esquelética possui uma camada específica *Skeletal Animation* demonstrada na Figura 1.

A camada *Human Interface Devices* (HID) é responsável pelo controle de entrada dos jogadores, normalmente os jogadores utilizam de dispositivos para interagir com o jogo. Os dispositivos mais utilizados são: teclado e mouse, controle, volante e outros dispositivos específicos como o acelerômetro dos *smarthphones*.

O áudio é um componente extremamente importante dos motores de jogo. A camada de *Audio* é tão importante quanto à dos gráficos. Cada jogo exige uma grande quantidade de desenvolvimento de software personalizado, atenção nos detalhes, para produzir um áudio de alta qualidade para o produto final.

Segundo Gregory (2009) alguns jogos permitem que vários jogadores humanos possam jogar em um mesmo mundo virtual, esses jogos são intitulados de *multiplayer* e o autor define quatro tipos básicos:

- ***Single-screen multiplayer***: duas ou mais pessoas que jogam através de um único dispositivo, fliperama, computador ou console. Nesse tipo de jogo vários personagens habitam em um único mundo virtual e uma única câmera mantém todos os personagens no mesmo quadro.
- ***Split-screen multiplayer****:* duas ou mais pessoas que jogam através de um único dispositivo, fliperama, computador ou console. Nesse tipo de jogo vários personagens habitam em um único mundo virtual, porém, a câmera é individual dividindo a tela em seções para que cada jogador possa ver seu personagem.
- ***Networked multiplayer***: vários dispositivos conectados em rede, com cada maquina hospedando um ou mais jogadores, por exemplo, o jogo *Castle Crashers* permite que um ou mais jogadores utilizem de uma mesma máquina para jogar com outros jogadores em rede.
- ***Massively multiplayer online games (MMOG)***: milhares de usuários podem estar jogando simultaneamente dentro de um gigante, persistente, mundo virtual hospedado por um conjunto de servidores centrais.

O termo *gameplay* ou jogabilidade refere-se a ação que ocorre no jogo, as regras que governam o mundo virtual em que o jogo acontece, as habilidades do personagem do jogador e outros objetos no mundo, como metas e objetivos do jogador (GREGORY, 2009). Cabe ao desenvolvedor do jogo determinar a jogabilidade na linguagem nativa do motor e/ou por scripts de alto nível.

A camada *Gameplay Foudations* preenche a lacuna entre jogabilidade e os motores de baixo nível, essa camada fornece um conjunto de núcleos para facilitar a forma que a lógica específica do jogo é implementada (GREGORY, 2009). Ainda segundo Gregory (2009) a camada *Gameplay Foundations* introduz a noção de mundo do jogo, contendo elementos estáticos e dinâmicos. Esses elementos podem ser desde corpos rígidos que pertencem ao cenário como pedras até personagens. Os tipos de objetos mais comuns são: veículos, armas, câmeras, personagens não jogáveis (NPC, *non-player characters*), personagens, objetos rígidos, elementos estáticos como prédios

e estradas.

A camada *Gameplay Foundations* possui um sistema de eventos que possibilita a comunicação entre objetos do jogo. Isso ocorre de várias formas, por exemplo, o objeto pode mandar uma mensagem para chamar uma função do objeto receptor, esse tipo de comunicação é conhecido como orientação a eventos. A orientação a eventos é muito utilizada no desenvolvimento de jogos, consiste basicamente em um receptor que espera alguma recomendação, esse receptor quando necessário executa uma determinada função a fim de responder a alguma solicitação de outro objeto (GREGORY, 2009).

Segundo Gregory (2009) os motores de jogo utilizam da linguagem de script a fim de facilitar o desenvolvimento focando nas regras de jogo e no conteúdo tornando o desenvolvimento mais rápido. Entretanto, existem motores que permitem ao desenvolvedor utilizar de uma linguagem diferente, por exemplo, a *Unreal Egine* permite a utilização de script como linguagem de programação ou c++.

A inteligência artificial presente nos jogos Gregory (2009) afirma que "a inteligência artificial (AI) não era considerada uma parte do motor de jogo, e sim como algo específico tratado fora do motor". Algumas empresas de desenvolvimento de jogos têm reconhecido padrões que surgem em quase todos os sistemas de AI de seus jogos. Com base nesses padrões remeteram ao motor de jogo uma área específica para a inteligência artificial. A Kynogon desenvolveu um motor de AI que atua como uma camada de base sobre a qual a inteligência artificial pode ser facilmente desenvolvida. Os recursos disponibilizados pela Kynogon são:

- ***Free-roaming***: permite que personagens não jogáveis (NPC) se movimentem pela superfície sem medo de colisão com algum objeto estático.
- **Informação de colisão**: simplifica as informações de colisão com bordas.
- **Conhecimento de região**: o inimigo pode ter conhecimento das entradas e saídas de determinada região, assim pode ser capaz de ver ou criar emboscadas.
- **Reconhecimento de caminhos:** motor para reconhecimento de caminhos baseados em um determinado algoritmo.
- **Linha de visão (LOS, line of sight)**: possui a funcionalidade de acessar o sistema de colisão e o modelo de mundo para atribuir determinadas

percepções.

- **Reconhecimento de entidades**: permite que os personagens e npcs se esquivem de obstáculos em movimento.

Esses recursos possibilitam a redução do número de linhas de código referentes à inteligência artificial dos personagens do jogo.

Segundo Gregory (2009) no topo da camada *Gameplay Foundation* atuam os programadores e artistas que cooperam entre si para implementar as características do jogo em si. Todo jogo contém um conjunto de subsistemas funcionando simultaneamente, a jogabilidade varia muito de acordo com a forma específica em que o jogo é desenvolvido.

A camada *Game-Specific Subsystems* inclui sistemas que vão além do controle da mecânica do personagem de cada jogador, como sistema de controle das câmeras, inteligência artificial, sistema de armas, veículos, desafios entre outros demonstrados na Figura 1. Gregory (2009) afirma que o jogo está situado entre os subsistemas e a camada *Gameplay Foundation*.

Todas essas camadas servem como base dos motores de jogos mais atuais, inclusive, dos motores utilizados neste trabalho, *Unity3D* e *Unreal Egine* 4 respectivamente. Na próxima seção será realizado um comparativo entre os motores citados.

2.3.2 COMPARAÇÃO ENTRE MOTORES DE JOGOS

Segundo Macedo e Rodrigues (2015) a variação existente entre os motores de jogos mais atuais está relacionado aos seguintes critérios:

- **Portabilidade:** capacidade do motor de jogo para criar aplicações para diferentes dispositivos, por exemplo, mobile, desktop e consoles.
- **Linguagem de programação:** delimita a linguagem de programação que o motor reconhece e tem suporte.
- **Curva de aprendizado:** implica na dificuldade ou tempo levado para se familiarizar com o motor de jogo.
- **Custo mensal ou licenciamento:** motores de jogos com versões pagas delimitam os recursos disponibilizados em suas versões gratuitas, assim, o

desenvolvedor paga uma mensalidade para ter acesso a todos os recursos. Outra forma das empresas lucrarem com o motor de jogo implica em reter uma parte do lucro gerado pela aplicação final.

Todos estes critérios somados ao nível de maturidade, suporte a APIs de computação gráfica e a documentação dos motores de jogos servem de base para a comparação inicial entre a *Unreal Engine* 4 e a *Unity3D*. A Tabela 1 apresenta um comparativo entre os motores de jogo de acordo com os critérios mencionados.

Tabela 1 - Tabela comparativa: *Unity3D* e *Unreal Engine 4*

Motor de jogo	**Unity 3D**	**Unreal Engine 4**
Portabilidade	Mobile, desktop e consoles	Mobile, desktop e consoles
Linguagem de programação	C#, Javascript e Boo	C/C++ e Blueprint
Curva de aprendizagem	Baixa	Média
Custo mensal/ Modelo de negócio	Versão gratuita e versão com mensalidade de U$75	Gratuito e *OpenSource* com 5% de royalties
Nível de maturidade	Alto	Alto
Suporte a APIs gráficas	DirectX 11+, OpenGL 3.2 até o 4.5, OpenGL ES 3	DirectX 10+, OpenGL 3.3 até o 4.5, OpenGL ES 3
Documentação	Completa e disponível	Completa e disponível

Fonte: http://www.seer.ufrgs.br/index.php/rita/article/view/RITA-VOL22-NR2-181/35626/

Como é possível perceber, a Tabela 1 demonstra as características dos motores selecionados para o estudo de caso, cada motor tem sua linguagem de programação e um modelo de negócio específico.

3 METODOLOGIA

Este capítulo descreve os métodos utilizados no processo de definição e construção da aplicação como estudo de caso.

3.1 MODELAGEM DA APLICAÇÃO

A aplicação desenvolvida para o presente trabalho é uma versão do jogo Frogger do Atari de 1981 denominada Jumper. A escolha de um jogo 2D possibilita a realização de testes em diversos smartphones diferentes com alta e baixa capacidade de processamento gráfico, também possibilita a avaliação do motor quanto ao suporte no desenvolvimento de jogos 2D. A Figura 2 demonstra o jogo Frogger do Atari.

Figura 2 - O jogo Frogger

Fonte: http://stamfordresearch.com/google-playing-atari-how/

A Figura 2 demonstra a tela de jogo do Frogger, o jogo consiste em levar o sapo para o outro lado do lago e deixa-lo nos cinco espaços identificados da parte superior, para isso, é necessário desviar dos automóveis na rua e evitar cair no rio antes que acabe o tempo. Por ser um jogo de Atari o personagem era controlado por um *joystick* (dispositivo de entrada), como o jogo desenvolvido tem como alvo smartphones com Android a movimentação do personagem é realizada através do *touch*. Para pontuar no Frogger o jogador tem que mover o sapo em direção dos espaços e se possível coletar elementos que aparecem aleatoriamente, a pontuação no Jumper é atribuída através do tempo que o jogador leva para chegar do outro lado do rio.

Para o motor da *Unity* adotou-se a linguagem de programação Javascript no desenvolvimento da aplicação e para o motor da *Unreal* a linguagem utilizada é denominada de Blueprint.

Para o desenvolvimento do jogo utilizou-se como base a documentação disponibilizada por cada empresa desenvolvedora do motor de jogo. A primeira tarefa realizada trata-se da coleta de requisitos da aplicação, após a coleta houve a definição do *layout* da aplicação com a delimitação de cada componente utilizado. Posteriormente foram definidos os *game assets*, recursos do jogo, por exemplo, textura, personagens, animações, cenário e música. Após a escolha dos recursos foi iniciado o desenvolvimento da aplicação utilizando da linguagem de programação suportada pelo motor.

3.2 TESTES

Após o desenvolvimento da aplicação com os dois motores de jogos foram realizados testes para comparar a aplicação gerada por cada motor. O primeiro teste realizado é o de FPS (*Frames Per Second)* que verifica o número de imagens transmitidas por segundo. O segundo teste verificou o consumo de bateria por tempo de uso, para cada aplicação foram determinados intervalos fixos para a coleta de informações comparando o consumo de cada aplicação. A próxima seção detalha o desenvolvimento do jogo Jumper com os motores da *Unity3D* e *Unreal Engine 4*.

4 DESENVOLVIMENTO DO JOGO JUMPER COM UNITY E UNREAL

Este capítulo descreve o processo que culminou no aplicativo denominado Jumper, uma versão do jogo Frogger desenvolvido com os motores Unity e Unreal Engine 4.

4.1 REQUISITOS DA APLICAÇÃO

Descrição das funcionalidades presentes na aplicação, com seus níveis de prioridade que são: Essencial “requisito imprescindível, sem o qual a aplicação não funciona”, Importante “requisito sem o qual a aplicação entra em funcionamento, de

forma não satisfatória" e desejável "que pode ser entregue a qualquer momento sem oferecer prejuízo ao a aplicação".

4.1.1 REQUISITOS FUNCIONAIS "RF"

RF01 – Retornar ao menu principal: A aplicação deve permitir que o usuário retorne a tela inicial em qualquer momento.

Prioridade: Importante;

RF04 – **Pausar:** O jogo poderá ser pausado no meio da partida. Ao ser pausado o jogador poderá voltar para o jogo, reiniciar a partida e ir para o menu inicial.

Prioridade: Importante;

RF02 – **Movimentar Personagem**: o personagem possuirá movimentações para a direita, esquerda, cima e baixo através do touch deslizando o dedo sobre a tela;

Prioridade: Essencial;

RF03 – Salvar pontuação: A aplicação gravará às 4 maiores pontuações. Essas pontuações serão gravadas no próprio dispositivo, sem a possibilidade de compartilhar.

Prioridade: Importante;

4.1.2 - REQUISITOS NÃO FUNCIONAIS "RNF"

Esta seção apresenta a descrição do conjunto de requisitos que compõem a aplicação, descrevendo atributos de qualidade e restrições que a aplicação deve possuir.

RNF01 – Interface responsiva

A aplicação deve ser responsiva, assim, permitir que a aplicação possa ser executada em diferentes *smartphones*.

Prioridade: Importante;

RNF02 - Usabilidade

A aplicação deve possuir uma usabilidade simples e intuitiva de fácil navegação para facilitar o uso por parte dos usuários.

Prioridade: Essencial;

RNF03 - Linguagem de Programação

A aplicação deverá ser desenvolvida utilizando das linguagens JavaScript para a *Unity* e Blueprint para a *Unreal*;

Prioridade: Essencial;

RNF04 - Persistência

A persistência da aplicação na *Unity* deverá ser realizada através do PlayerPrefs. A persistência na *Unreal* deverá ser realizada através do *GameInstance*.

Prioridade: Importante;

RNF05 - Instabilidade

O jogo deve renderizar os objetos com frames acima de 30 quadros por segundo.

Prioridade: Essencial;

RNF06 - Apresentação da Interface Gráfica

A aplicação utilizará exclusivamente a língua portuguesa para todo e qualquer texto apresentado.

Prioridade: Importante;

RNF07 - Sistema operacional Android;

A aplicação poderá ser executada nos smartphones com sistema operacional Android na versão 4.4 ou superior;

Prioridade: Essencial;

4.2 PROTÓTIPO DE TELAS

Após a coleta de requisitos foram criados os protótipos de cada tela do aplicativo com o intuito de definir o espaço e o local de cada elemento na tela visando facilitar a

utilização do aplicativo. Para a prototipação utilizou-se a ferramenta Pencil, um *software* livre para prototipação de telas. A Figura 3 demonstra a prototipação da tela inicial e tela de informação dos desenvolvedores.

Figura 3 - Protótipo da tela inicial e da tela com informações dos desenvolvedores.

Fonte: Autor da obra, 2017.

A tela de informações dos desenvolvedores é acessada através do botão de informações localizado a esquerda do botão jogar na tela inicial. A tela de desenvolvedores possui um painel com informações como nome e e-mail, A tela inicial possui também um botão para exibir os recordes e o botão para jogar. A Figura 4 demonstra a prototipação da tela de instruções e de pontuação.

Figura 4 - Protótipo das telas de instruções e pontuação

Fonte: Autor da obra, 2017.

A tela de instruções é acessada através do botão jogar, nessa tela planejou-se dar informações sobre como vencer o jogo, só é liberado o botão para jogar após o jogador confirmar a leitura das instruções. Para à tela de recordes as pontuações foram organizadas dentro de um painel com espaço para guardar as 4 maiores.

Por fim para a tela do jogo planejou-se seguir a mesma configuração do Frogger, uma parte de uma rodovia e um lago paralelo à rodovia. A Figura 5 apresenta as telas do jogo e do fim de jogo.

Figura 5 - Tela do jogo e tela de fim do jogo.

Fonte: Autor da obra, 2017.

Após clicar no botão jogar na tela de instruções a aplicação exibe a tela inicial do jogo, o fim de jogo é apresentado assim que as vidas acabam. As vidas foram posicionadas no canto superior direito e o tempo pode ser visualizado no canto superior esquerdo.

4.3 IMPLEMENTAÇÃO COM A UNITY

Após a prototipação das telas iniciou o processo de desenvolvimento do aplicativo com a Unity. Para este trabalho utilizou-se da versão 5.5.0f3 da Unity, a

primeira tarefa no desenvolvimento do aplicativo foi realizar a configuração do projeto, para isso, nas configurações foi determinado as dimensões, neste caso um jogo 2D, assim, o motor elimina do desenvolvedor a preocupação com a câmera. Após determinar as dimensões do jogo também é necessário informar para qual plataforma o jogo será desenvolvido, nesse caso foi determinado uma aplicação para Android. A Figura 6 demonstra a interface da Unity e o menu principal.

Figura 6 - Interface da *Unity* e tela inicial do aplicativo Jumper.

Fonte: Autor da obra, 2017.

Como demonstra a Figura 6, a interface da Unity é bem simples e intuitiva, pode-se alterar a posição e quantidade de menus pelo desenvolvedor para adequar ao seu modo. Possui basicamente uma área para demonstrar o jogo e as transformações que ocorrem no jogo denominada de *Game*, uma área para editar os elementos apresentados na tela denominada de *Scene*, além dos painéis com as pastas do projeto, hierarquia dos objetos posicionados na tela e as propriedades dos elementos selecionados demonstrado no menu *Inspector*.

Após a configuração do motor iniciou-se o processo de criação das cenas para cada menu seguindo o *Layout* definido na prototipação. Apesar de Unity tirar do desenvolvedor a preocupação com a câmera para jogos 2D ainda é necessário trabalhar no plano 3D para ajustar botões e imagens para que no *smartphone* não sejam sobrepostos. Mesmo preenchendo o campo *Order in Layer* que tem como função

organizar em camadas o que será exibido na tela foi necessário sair do modo 2D e alterar a posição em Z de alguns componentes.

Para a implementação da lógica e do modo de jogo foram criados scripts para controlar o personagem, tratamento da colisão e pontuação. A movimentação do personagem é dada através do *touchscreen* sem a utilização de botões, para movimentar o personagem o jogador tem que deslizar o dedo para a direção que deseja movimentar. A Figura 7 demonstra o script criado para movimentar o personagem.

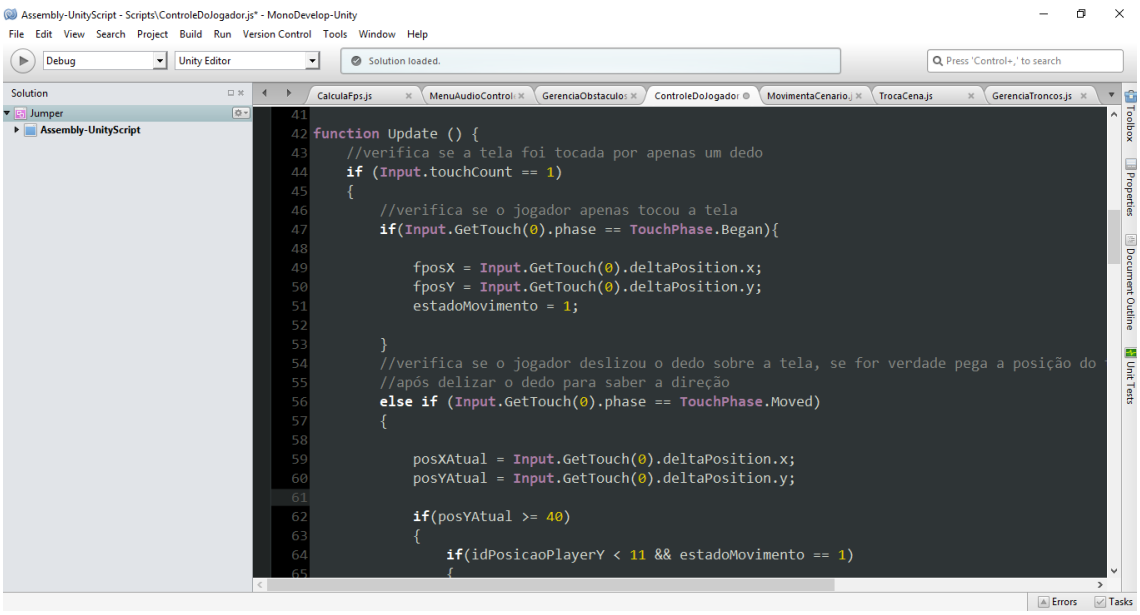

Figura 7 - Código em Javascript para criar a movimentação do personagem.

Fonte: Autor da obra, 2017.

A Figura 7 demonstra uma parte do código da movimentação do personagem, foi implementada utilizando de um conceito conhecido como máquina de estados, para cada ação determinou-se um estado a fim de evitar vários comandos ao mesmo tempo.

Quando o *touch* é apenas pressionado coleta-se a posição do toque na tela e armazena em duas variáveis fposX referente a posição no eixo x e fposY referente a posição no eixo y respectivamente. O comando *TouchPhase* descreve a fase de um toque na tela, quando o dedo apenas toca a tela o *TouchPhase* está como *Began*, se o jogador movimenta o dedo na tela o *TouchPhase* muda para *Moved*. Assim, quando o *TouchPhase* está como *Moved* coleta-se as informações do eixo X e Y comparando com para determinar a direção do movimento e assim movimentar o personagem.

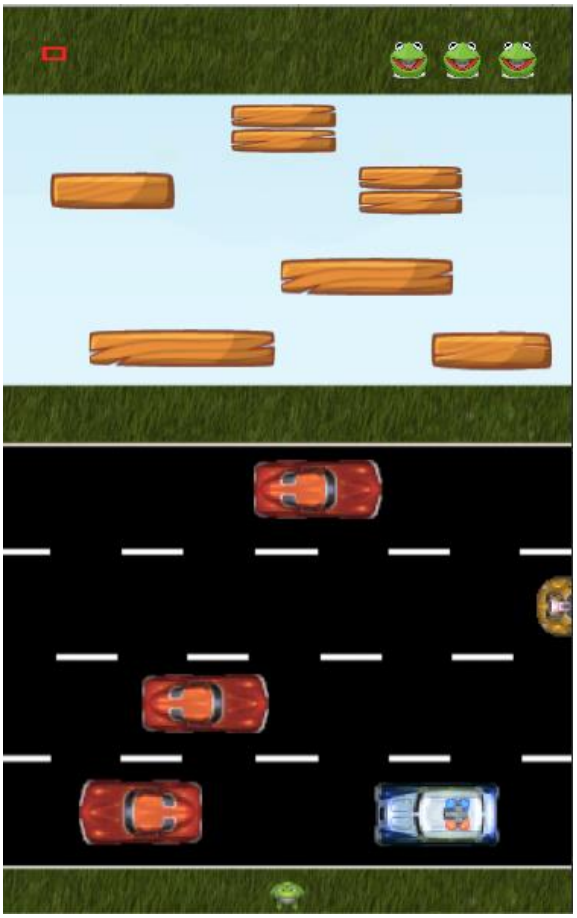

Figura 8 - Cena de jogo.

Fonte: Autor da obra, 2017.

A Figura 8 demonstra a cena de jogo criado pelo motor da *Unity*, é importante ressaltar que em alguns *smartphones* o plano de fundo ficou menor que o tamanho da tela, apresentando uma área azul, para resolver este problema foi necessário aumentar a imagem de fundo para cobrir uma área maior que a tela.

Diferente do Frogger a pontuação é atribuída através do tempo, quanto mais tempo o jogador leva para atravessar o mapa menor a sua pontuação, se o tempo chegar a 0 o jogador perde uma vida e o tempo volta a sua numeração inicial. Os três sapos no canto superior direito representam o número de sapos para atravessar o mapa, uma vez que o jogador leve um sapo para o outro lado também é retirada uma vida. Para vencer o jogo o jogador deve atravessar os três sapos no menor tempo possível.

Por fim, o arquivo APK (*Android Package*) gerado pela *Unity* tem o tamanho de 34,4 *megabyte*, o tamanho do arquivo é apresentado sem marcar as opções de compressão na configuração do projeto.

4.4 IMPLEMENTAÇÃO COM A UNREAL ENGINE 4

Para este trabalho utilizou-se a versão 4.17.2 da *Unreal Engine*, a implementação no motor teve como primeira etapa a configuração do projeto, definiu-se o *hardware* ao qual a aplicação foi desenvolvida, a *Unreal* classifica o *hardware* em dois tipos sendo eles Mobile/Tablet ou Desktop/Console, para este trabalho foi selecionado a opção Mobile/Tablet. A Figura 9 demonstra a interface da *Unreal Engine*, o *layout* da interface pode ser alterado pelo desenvolvedor, pode-se ocultar os menus e muda-los de posição.

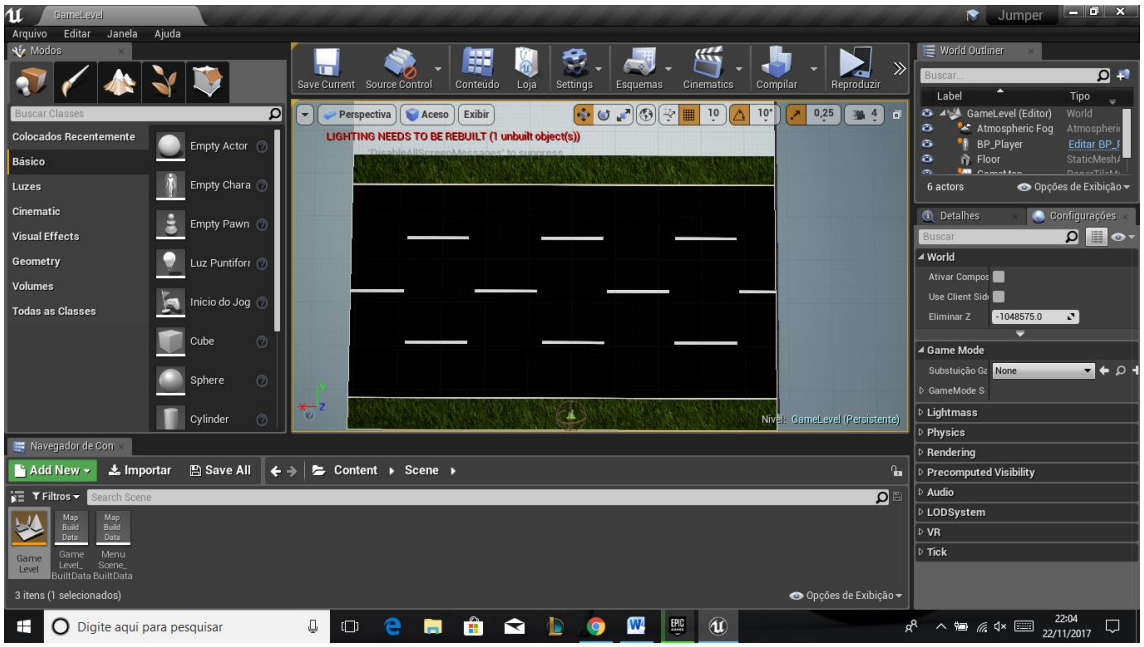

Figura 9 - Interface da *Unreal Engine 4*.

Fonte: Autor da Obra, 2017.

Após as definir as configurações do projeto iniciou-se o processo de criação dos menus. Para criar os menus foi utilizado o *UserWidget*, com esse recurso não é necessário criar uma cena para cada menu, pode-se criar a cena do jogo e vários *UserWidget* para utilizar como menu. A Figura 10 apresenta o menu inicial.

Figura 10 - Interface de criação do *UserWidget*.

Fonte: Autor da obra, 2017.

O blueprint do menu pode ser criado através da aba grafo ao lado da aba pré-selecionada denominada *Designer*. Para que o menu seja exposto é necessário criar um blueprint informando qual menu será exibido na tela. A movimentação do personagem foi implementada seguindo o mesmo padrão utilizado na *Unity*. Para movimentar o personagem o jogador deliza o dedo sobre a tela para a direção que deseja movimentar. A Figura 11 apresenta o blueprint criado para a movimentação do personagem.

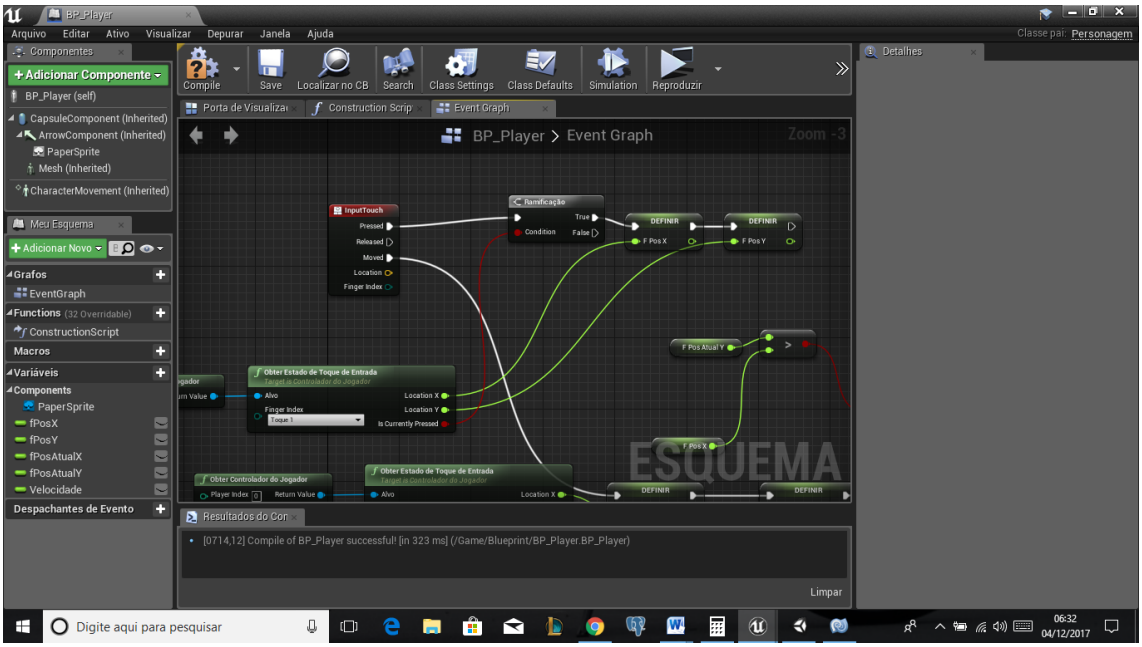

Figura 11 - Blueprint utilizado para a movimentação do personagem.

Fonte: Autor da obra, 2017.

A linguagem de programação da *Unreal* é considerada de alto nível e totalmente visual, em comparação com o Javascript da *Unity* não exige um grande conhecimento prévio de programação, elimina preocupações com indentação e permite ao desenvolvedor a identificação de erros lógicos de forma rápida através do debug visual.

A *Unreal*, apesar de ser uma ferramenta com diversas funcionalidades, para o desenvolvimento 2D, apresenta alguns problemas, entre eles estão problemas com o editor que opera preferencialmente no plano 3D, excesso de configurações manuais além de ser uma ferramenta pesada exigindo bastante do *hardware*. Destaca-se no motor a linguagem de programação, que torna o desenvolvimento mais intuitivo.

Por fim, o arquivo APK gerado pela *Unreal* tem o tamanho de 62,3 *megabytes*, o tamanho do arquivo é apresentado sem marcar as opções de compressão na configuração do projeto.

5 RESULTADOS

Este capitulo descreve os resultados obtidos através dos testes realizados com a aplicação gerada pelos motores da *Unity* e *Unreal* e um comparativo da curva de aprendizagem obtida em cada motor. Para os testes realizados foram utilizados *hardwares* específicos demonstrados na Tabela 2.

Tabela 2 - *Hardwares* utilizados

HARDWARE	**FABRICANTE**	**MODELO/VERSÃO**	**S.O.**	**RAM**	**GPU/**
NOTEBOOK	ASUS	250T	WINDOWS 10	8 GB	NVIDIA 940M
SMARTPHONE	SONY	Xperia Z3	ANDROID 6.0	3 GB	ADRENO 330

Fonte: Autor da obra, 2017.

Como demonstra a Tabela 2 para a viabilidade do trabalho utilizou-se como *hardware* componentes com grande poder de processamento gráfico. A versão do Android definida para o projeto é denominada de *KitKat* com número 4.4, atribuir uma versão antiga do Android permite a instalação do aplicativo em *smartphones* com essa versão ou superior.

5.1 TESTE DE QUADROS POR SEGUNDO

Para realizar o teste de fps foi implementado uma função para retornar o número de quadros por segundo. A Figura 12 apresenta o resultado do teste de fps.

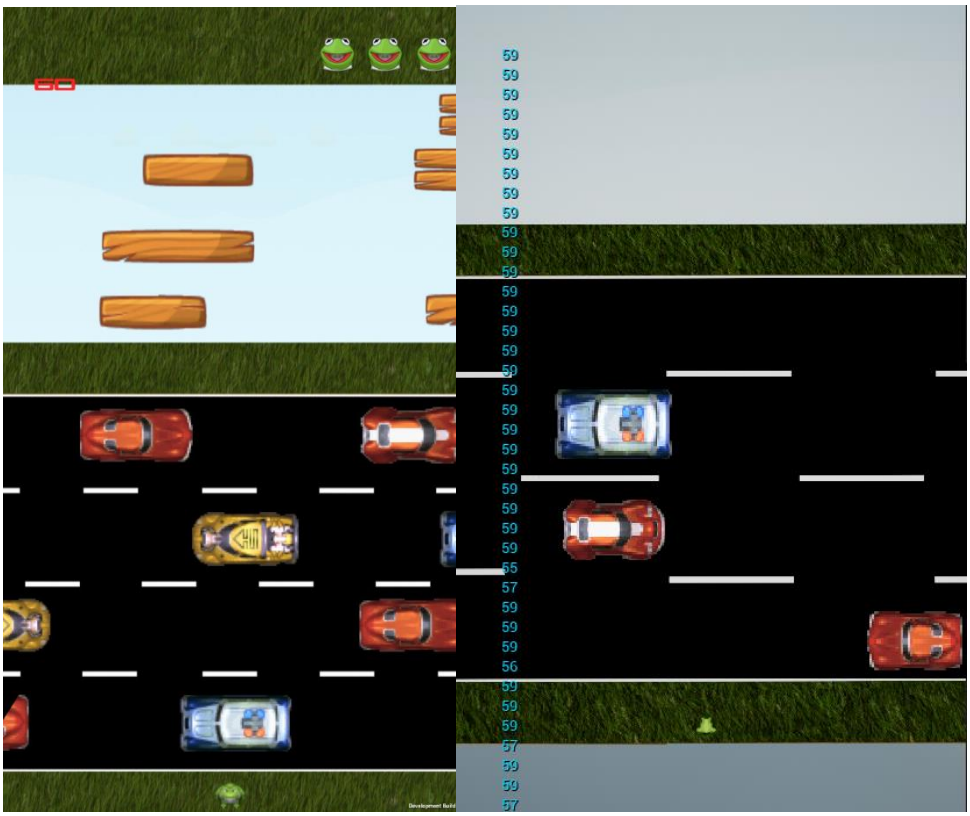

Figura 12 - Teste de FPS.

Fonte: Autor da obra, 2017.

A Figura 12 apresenta o fps obtido pela aplicação gerada através dos motores da *Unity* e *Unreal*. A aplicação da *Unity*, localizada a esquerda da imagem, apresentou uma variação de 60 a 62 quadros por segundo. Jogos que apresentam o fps superior a 60 tem maior qualidade, entretanto, só é possível perceber uma grande diferença caso o fps seja inferior a 30, nesse caso pode-se perceber falhas e travamentos. Como é possível perceber a aplicação da *Unreal* apresentou uma variação de 55 a 59 quadros por segundo. Apesar da aplicação gerada pela *Unreal* apresentar uma maior variação no fps do jogo a jogabilidade não foi prejudicada mantendo sempre o fps acima de 55.

5.2 TESTE DE CONSUMO DE BATERIA

Para o teste de consumo de bateria definiu-se o seguinte ambiente: somente o jogo aberto, sem aplicativos em segundo plano, wifi desabilitado, bluetooth desabilitado, brilho da tela em 10% e carga do aparelho em 100%. Os testes foram realizados em intervalos de 30 minutos, sempre partindo da bateria 100% e coletando

dados após 30 minutos jogando. A Figura 13 apresenta o resultado dos testes de consumo de bateria.

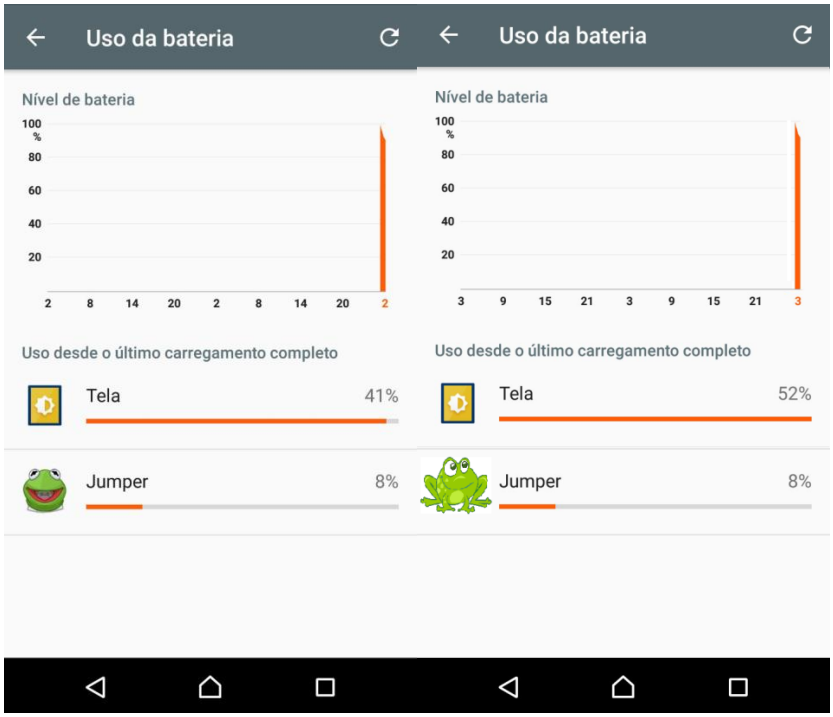

Figura 13 - Consumo de bateria por aplicativo.

Fonte: Autor da obra, 2017.

Em todos os testes realizados o resultado foi de 8% no consumo de bateria após 30 minutos de uso, cabe ressaltar que ambos os aplicativos foram testados na cena de jogo.

5.3 CURVA DE APRENDIZADO

Para determinar a curva de aprendizagem foram registradas as horas gastas no desenvolvimento do aplicativo, cada etapa tem um custo operacional e requer um tempo para ser finalizada. Com o intuito de demonstrar as dificuldades enfrentadas no desenvolvimento do aplicativo a Tabela 3 apresenta um comparativo entre os motores da *Unreal* e *Unity* especificando o tempo em horas para o desenvolvimento.

Tabela 3 - Tempo gasto por função.

Motores	Configuração do projeto	Criação do Menu	Movimentação Personagem	Tratamento de Colisão
Unity3D	1 h	3 h	5 h	6 h
Unreal Engine 4	3 h	2 h	5 h	6 h

Fonte: Autor da obra, 2017.

A Tabela 3 apresenta a carga horaria para implementar e configurar algumas funções em cada motor. A Unity apresentou um editor mais consistente, reduzindo o numero de configurações para iniciar a implementação da lógica do jogo. Em comparação com a *Unity* o principal problema encontrado no motor da *Unreal* foi a quantidade de configurações necessárias para iniciar o desenvolvimento, após configurar o projeto definindo o *hardware* o desenvolvedor tem que desabilitar o *joystick*, informar o local da pasta do sdk do Android, assim como o ndk e configurar a câmera do jogo.

6 CONCLUSÃO

Este trabalho teve como foco o estudo comparativo entre os motores de jogos da *Unity* e *Unreal*, para determinar os impactos no projeto e na aplicação recorrentes da escolha de um motor no desenvolvimento de jogos 2D para Android. O estudo de caso apresentado neste trabalho demonstrou o processo de desenvolvimento com cada motor bem como as dificuldades e os problemas encontrados.

Através do estudo de caso foi possível determinar que para o desenvolvimento de jogos 2D voltados ao sistema Android a *Unity* garante maior eficácia, além de proporcionar maior produtividade no desenvolvimento. O editor da *Unity* separa o plano 2D do 3D permitindo com que o desenvolvedor trabalhe apenas em um plano. Através dos testes realizados e do processo de desenvolvimento a ferramenta demonstrou dar

maior agilidade no desenvolvimento de jogos 2D.

6.1 TRABALHOS FUTUROS

Pretende-se como trabalhos futuros:

- Realizar um estudo de caso no desenvolvimento de jogos 3D voltados ao sistema operacional Android com os motores da *Unity* e *Unreal*.
- Mensurar parâmetros de renderização (fps) e consumo de bateria.
- Mensurar o uso da GPU.

7 REFERÊNCIAS

CRAWFORD, C. **The Art of Digital Game Design**, Washington State University, Vancouver, 1982.

DE MACEDO, Daniel Valente; RODRIGUES, Maria Andréia Formico; SERPA, Yvens Rebouças. **Desenvolvimento de Aplicações Gráficas Interativas com a Unreal Engine 4.** Revista de Informática Teórica e Aplicada, v. 22, n. 2, p. 181-202, 2015.

FALKEMBACH, Gilse A. Morgental. O lúdico e os jogos educacionais. Mídias na Educação. Disponível em: http://penta3.ufrgs.br/midiasedu/modulo13/etapa1/leituras/arquivos/leitura_1.pdf Acesso em 20 de maio de 2016, v. 16, 2006.

GREGORY, Jason**. Game engine architecture**. CRC Press, 2009.

GOLD, Julian. **Object-oriented game development**. Pearson Education, 2004.

HUIZINGA, J. **Homo ludens: o jogo como elemento da cultura**. 5o. ed. [S.l.]: Perspectiva, 2003. p. 256.

KOSTER, Raph. **Theory of fun for game design**. " O'Reilly Media, Inc.", 2013.

MORIBE, Vinícius Akira. **Jogo para Android com Unity3D**. Reverte-Revista de Estudos e Reflexões Tecnológicas da Faculdade de Indaiatuba, n. 10, 2012.

MENDES, Cláudio Lúcio. **Jogos eletrônicos: diversão, poder e subjetivação**. Papirus Editora, 2006.

NETMARKETSHARE, **Realtime Web Analytics With no Sampling**. Disponivel em: <http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=9&qpcustomb=1&qpct=4&qpsp=175&qpnp=12&qptimeframe=M > Acesso em 25 de fevereiro de 2015.

PRENSKY, Marc. Computer games and learning: Digital game-based learning**. Handbook of computer game studies**, 2005.

SUPERDATA, **Latin American digital games Market hits $4.5B.** Disponivel em: <http://www.superdataresearch.com/blog/latin-american-digital-games-market-hits-4-5b/> Acesso em 25 de fevereiro de 2015.