**UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC**
**CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**EDERSON DUARTE SCHAUKOSKI**

# ANÁLISE COMPARATIVA DE *FRAMEWORKS OPEN SOURCE* PARA O DESENVOLVIMENTO DE JOGOS MULTIPLATAFORMA

**CRICIÚMA**
**2018**

**EDERSON DUARTE SCHAUKOSKI**

# ANÁLISE COMPARATIVA DE *FRAMEWORKS OPEN SOURCE* PARA O DESENVOLVIMENTO DE JOGOS MULTIPLATAFORMA

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

Orientador: Prof. Me. Luciano Antunes

**CRICIÚMA**

**2018**

**EDERSON DUARTE SCHAUKOSKI**

**ANÁLISE COMPARATIVA DE *FRAMEWORKS OPEN SOURCE* PARA O DESENVOLVIMENTO DE JOGOS MULTIPLATAFORMA**

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

Criciúma, 26 de novembro de 2018.

**BANCA EXAMINADORA**

Prof. Luciano Antunes – Me. – (UNESC) – Orientador

Profa. Ana Claudia Garcia Barbosa – Ma. – (UNESC)

Prof. Paulo João Martins – Me. – (UNESC)

**Dedico este trabalho a todos que me apoiaram, meus pais e familiares.**

## AGRADECIMENTOS

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

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

Agradeço a meus pais, familiares e professores que me auxiliaram.

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

**“Algo só é impossível até que alguém duvide e acabe por provar o contrário.”**

**Albert Einstein**

## RESUMO

Jogos Digitais são um meio de entretenimento que vem crescendo bastante, estão disponíveis nos mais diversos dispositivos como aparelhos celulares inteligentes (smartphones). Diante deste crescimento surgem também a criação de *frameworks* de desenvolvimento, muitos disponibilizados gratuitamente e sendo de código aberto, podendo ser modificado por qualquer usuário ou comunidade. Por meio do uso de um *framework*, é possível gerar aplicações para diversas plataformas utilizando uma mesma linguagem de programação. Na criação de jogos digitais, os algoritmos de detecção de colisão são muito utilizados para a manipulação dos elementos do cenário e verificar a sobreposição entre dois ou mais objetos em um dado intervalo de tempo. A utilização de um protótipo fornece ao *game designer* uma visão aprimorada do desenvolvimento conceitual de um jogo, ou seja, não basta possuir a ideia, é necessário testar sua viabilidade ao longo do processo produtivo. Em diversos gêneros de jogos, o uso de comportamento nos personagens e elementos gráficos também impactam no desempenho do jogo, e consequentemente, na experiência do jogador. Nesta pesquisa, utilizando-se dos *frameworks* Phaser, Libgdx e Cocos2d-x, foi implementado o algoritmo da sobreposição de retângulos que por sua vez foi utilizado para a verificação de detecção de colisão e realizar a medição do tempo de execução. Para a comparação dos *frameworks,* foi utilizado métricas de avaliação de funcionalidades pelo qual foi definido critérios baseando-se em sua ponderação e o seu grau de criticidade. Dentre os protótipos testados, os resultados obtidos demonstraram que, o impacto computacional possui suas diferenças entre cada aplicação gerada e que dependem também da complexidade do mesmo.



**Palavras-chave:** Jogos Digitais. Frameworks. Phaser. Libgdx. Cocos2d-x.

## ABSTRACT

Digital games are a means of entertainment that has grown a lot, are available in various devices such as smartphones. Facing this growth also arise the creation of development frameworks, many available free and being open source, and can be modified by any user or community. Through the use of a framework, it is possible to generate applications for several platforms using the same programming language. In the creation of digital games, collision detection algorithms are widely used for manipulating the elements of the scenario and verifying the overlap between two or more objects in a given time interval. The use of a prototype gives the game designer an improved view of the conceptual development of a game, that is, it is not enough to have the idea, it is necessary to test its viability throughout the production process. In various gaming genres, the use of character and graphic behavior also impacts game performance, and consequently, player experience. In this research, using the Phaser, Libgdx and Cocos2d-x frameworks, the algorithm of the overlapping of rectangles was implemented, which in turn was used to verify collision detection and perform the measurement of execution time. In order to compare the frameworks, functional evaluation metrics were used by which criteria were defined based on their weighting and their degree of criticality. Among the prototypes tested, the obtained results demonstrated that, the computational impact has its differences between each application generated and that also depend on the complexity of the same.



**Keywords:** Digital Games. Frameworks. Phaser. Libgdx. Cocos2d-x.

## LISTA DE ILUSTRAÇÕES

## LISTA DE TABELAS

## LISTA DE QUADROS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| 2D | Duas Dimensões |
| 3D | Três Dimensões |
| API | *Application Programming Interface* |
| FPS | *First Person Shooter* |
| GWT | *Google Web Toolkit* |
| IA | Inteligência Artificial |
| IDE | Integrated Development Environment |
| LOC | *Lines Of Code* |
| LWJGL | *Lightweight Java Game Library* |
| UI | *User Interface* |

**SUMÁRIO**

# 1 INTRODUÇÃO

Em uma média de faturamento de dezenas de bilhões de dólares e em constante evolução, o mercado mundial de jogos apresenta ser promissor, uma vez que empresas estão investindo cada vez mais no desenvolvimento de técnicas e programas computacionais principalmente os que envolvam interfaces, animações gráficas, vídeos, sons tridimensionais e ambientes multiusuários baseados na Internet (MADEIRA, 2001).

Com o crescimento do poder computacional a partir da década de noventa, o paradigma de desenvolvimento de jogos digitais passou a ser modificado, diante disso, as indústrias começaram a lançar jogos cada vez mais complexos, pois o poder de hardware também foi evoluindo com o passar do tempo e consequentemente, o tempo de desenvolvimento acabou ficando mais longo (ROCHA, 2003).

*Framework* é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica, podendo atingir uma função específica durante a programação de uma aplicação (MEDEIROS, 2014).

O conceito de *frameworks* se tornou o ponto chave no processo de desenvolvimento de um jogo, pela sua facilidade de reutilização do código, tendo em vista a abstração da maior parte dos detalhes de implementação de baixo nível, logo, o risco dos projetos apresentarem problemas futuros diminuiu consideravelmente (ROCHA, 2003).

O interesse por esse estudo surgiu mediante a oportunidade de diferenciar três *frameworks* existentes no mercado. Ao final desta pesquisa, pretende auxiliar a comunidade de desenvolvedores de jogos, já que muitas vezes, a incerteza e a imprecisão na escolha de certas ferramentas, fará com que grandes projetos acabem não se predominando no mercado pela falta do planejamento inicial.

Desta forma, esta pesquisa aqui proposta, enfatiza uma análise comparativa em *frameworks open source* para o desenvolvimento de jogos multiplataforma considerando a validação das particularidades e características específicas de cada uma através de métricas de software.

## 1.1 OBJETIVO GERAL

Realizar a análise comparativa dos *frameworks* Libgdx, Phaser e Cocos2d-x voltados para o desenvolvimento de jogos digitais.

## 1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desta pesquisa consistem em:

a) identificar métricas de software a serem usadas para análise dos frameworks;
b) desenvolver um protótipo em cada framework para realizar a análise;
c) identificar as características dos frameworks utilizadas;
d) expor testes estatísticos para a comparação das aplicações;
e) analisar os resultados dos testes e medidas empregados.

## 1.3 JUSTIFICATIVA

Nos últimos anos, jogos de computador e vídeo games têm evoluído de maneira acelerada, acompanhando as novas tecnologias e impulsionando o aparecimento de outras (MADEIRA, 2001).

Com a grande concorrência de mercado, os projetistas buscam novas formas de programação que aliem custo baixo, agilidade e qualidade em um mercado tão lucrativo quanto competitivo, a reusabilidade tem um papel importante neste contexto. No entanto, existem diversas ferramentas que auxiliam nestas tarefas, como os motores de jogos (*game engine*) e os chamados *frameworks* de desenvolvimento (FIGUEIREDO, 2014).

De acordo com Zadra, Carvalho e Cardoso (2015) a crescente demanda pelo desenvolvimento de software torna necessário a medição e estimativa de fatores envolvidos nesta atividade.

Para diminuir os riscos, a indústria está crescentemente utilizando técnicas e metodologias da engenharia de software, em particular os *frameworks* para a construção de jogos. Mesmo para as aplicações mais simples, irá desencadear na utilização de conceitos de várias áreas da Ciência da Computação,

tais como Engenharia de Software, Computação Gráfica, Inteligência Artificial, Redes de Computadores e Computação Musical (ROCHA, 2003).

Em relação a importância que os *frameworks* trazem aos desenvolvedores, pode-se afirmar que o alcance do seu foco principal passa a estar somente na lógica do jogo, ou seja, tornando transparentes detalhes de hardware, da física dos objetos e detalhes como inteligência artificial. Desta forma, há uma redução nas linhas do código fonte, fazendo com que os algoritmos de mais alta complexidade são encapsulados facilitando o programador, pois não é necessário conhecer toda a arquitetura de cada plataforma para desenvolver, ajudando também na manutenção posterior.

Portanto, esta pesquisa é essencial para o aprofundamento dos conhecimentos dos desenvolvedores ou pesquisadores deste mesmo assunto e para ter consciência das vantagens e desvantagens das particularidades de cada *framework* e dos fatos que serão abordados através do levantamento bibliográfico e testes estatísticos.

## 1.4 ESTRUTURA DO TRABALHO

A estrutura do trabalho é organizada em seis capítulos. No capítulo 1 são descritos o tema proposto.

No segundo capítulo, é apresentado o conceito de jogos digitais assim como os métodos de desenvolvimento para a criação do mesmo. São descritos também ao final deste capítulo os modelos de prototipagem usados em jogos digitais.

No terceiro capítulo, discute-se sobre o conceito de *framework*, a diferenças entre os motores de jogos bem como a arquitetura ou estrutura dos *frameworks* usados para comparação.

No quarto capítulo, são abordados o uso de métricas de software e como se utilizar a avaliação de funcionalidades em motores de jogos.

No quinto capítulo são mencionados os trabalhos correlatos com base nesta pesquisa.

No sexto capítulo é relatado a descrição do trabalho proposto assim como sua metodologia e as métricas necessárias para serem aplicadas.

# 2 JOGOS DIGITAIS

Os jogos eletrônicos são um meio de entretenimento que vem crescendo bastante, com a chegada das novas tecnologias e aparelhos celulares inteligentes (smartphones), os jogos estão disponíveis nos mais diversos dispositivos. De acordo com uma pesquisa feita pela *Entertainment Software Association* nos Estados Unidos, 50% dos americanos jogam jogos eletrônicos. Destes apenas 35% são menores de 18 anos, a média de idade de 35 anos e em média os jogadores gastam cerca de 1h por dia. E outro dado interessante é que os jogadores gastam mais que o triplo do tempo jogando do que praticando esportes ou lendo. O mercado de jogos eletrônicos movimentou US$ 28 bilhões em todo o mundo em 2003, e que desde 2003 superou o mercado cinematográfico (BRASILIENSE, 2006).

Os primeiros jogos digitais que foram desenvolvidos não eram jogados em residências ou em salas de estar, mas sim, em departamentos de pesquisa, universidades, laboratórios e instalações militares. Enquanto isso, alguns estudantes disciplinados, programadores, professores e pesquisadores transformaram seus computadores mainframes em máquinas de jogos (NOVAK, 2011, tradução nossa).

## 2.1 DESENVOLVIMENTO EM JOGOS DIGITAIS

Jogos digitais são aplicações muito complexas que englobam não só a área da computação, mas também diversas outras, a utilização de *framework* para o desenvolvimento de jogos é muito útil para criar projetos com maior facilidade e agilidade pois proporcionam uma gama imensa de recursos pré-programáveis proporcionado por objetos e modelos de classes já existentes. Infelizmente, ainda não há uma definição clara de um *framework* para jogos, nem tão pouco um estudo detalhado da aplicabilidade de padrões de projeto, mas alguns autores utilizam algumas definições utilizando o senso comum de perspectiva (MADEIRA, 2001).

A complexidade para a criação de jogos digitais tanto em três dimensões (3D) ou duas dimensões (2D), demanda muito tempo até sua finalização, desde o esboço do projeto até a parte da programação. O uso de *frameworks* é imprescindível para o desenvolvimento em jogos digitais visto que auxiliam na

produtividade pelo fato de englobar diversas ferramentas centralizadas (PEREIRA et al., 2017).

### 2.1.1 Métodos de desenvolvimento

Muitos desenvolvedores tendem a evitar de utilizar metodologias ou processos formais e partem direto para a produção sem antes mesmo definir um agendamento claro do que realmente foi pensado, isso só funcionava 25 anos atrás onde os jogos eram muito mais simples, onde tornava mais fácil saber o que cada integrante da equipe precisava desenvolver. Hoje em dia, como as equipes são maiores e os jogos mais complexos, os desenvolvedores estão percebendo que devem encontrar outras soluções para gerenciar o ciclo de desenvolvimento (CHANDLER, 2012).

Um dos métodos de desenvolvimento ágeis de jogos eletrônicos bastante utilizados é o chamado Scrum que é baseado nas melhores práticas da indústria e que está em crescimento constante não só na área de jogos, mas também na criação de softwares comerciais. Um dos aspectos do Scrum é sua característica empírica e inovadora que é utilizado um método de desenvolvimento incremental, onde os requisitos sofrem constantes alterações durante o ciclo de vida do produto (LEITÃO, 2010).

### 2.1.2 O Processo do Game Design

O *game design* é o ato de decidir como o jogo deverá ser no futuro, mas para isso é necessário ter todos os conceitos fundamentados e ter criatividade. Um *game designer* é o profissional que cuidará de todo o planejamento, conceituação e gerenciamento de um jogo, sendo que em empresas maiores estas tarefas são divididas em mais profissionais. Todas estas funções são compartilhadas com a equipe de desenvolvimento para que todos tenham consciência das etapas que já foram implementadas ou o que deve ser corrigido (SCHELL, 2008).

Para Fullerton (2008, tradução nossa) o principal componente para o *design* é a prototipação de ideias pelos quais precisam ser testadas o quanto antes.

Logo, o processo de prototipação precisa ser realizado antes de ser efetuado a produção do jogo para evitar grandes prejuízos.

## 2.2 PROTOTIPAGEM

A prototipagem é a criação de um modelo funcional de uma ideia pelos quais nos permite verificar a viabilidade do projeto e fazer melhorias constantes. Quando se está construindo um protótipo, não é necessário se preocupar com a perfeição ou se a tecnologia parece estar bem otimizada, tudo que se precisa se preocupar são as mecânicas fundamentais. Há muitos casos de *designers* de jogos que optaram por adiantar o processo de prototipagem e iniciaram diretamente na implementação do código influenciando no tempo investido (FULLERTON, 2008, tradução nossa).

Salen e Zimmerman (2004, tradução nossa) afirmam que não é suficiente escrever um documento de *design* para ser possível prever a experiência de um jogo. Por consequência, os autores justificam que através do processo de *design* iterativo o jogo é desenvolvido baseando-se no ato de jogar e ao mesmo tempo dispondo de uma visão crítica, aprendendo os pontos que se destacam e os menos significantes.

A prototipagem faz com que se tenha uma importante ligação entre o processo de *game design* e uma melhor comunicação entre os membros da equipe. A utilização de um protótipo fornece ao *game designer* uma visão aprimorada do desenvolvimento conceitual de um jogo, ou seja, não basta possuir a ideia, é necessário testar sua viabilidade ao longo do processo produtivo (SATO, 2010).

Através do processo iterativo, obtemos diversas ações que são modificadas ao longo da criação de um determinado produto, gerando assim, novas variações ou versões deste produto. Portanto, cada etapa avançada ou avaliada é realizada através da observação e experimentação nos protótipos que são criados, conforme apresentado na figura 1 (CREDIDIO, 2007).

Figura 1 – Processo iterativo de design


Fonte: Credidio (2007).

Existem diversos tipos de prototipagem para jogos, serão mencionados três para o desenvolvimento de modelos de protótipos, a prototipagem física, em papel e digital.

### 2.2.1 Prototipagem Física

Os protótipos físicos são os mais fáceis de construir, a grande maioria são feitos usando folhas de papel, papelão e objetos simples de uso doméstico. Além disso, a vantagem dos protótipos físicos é fazer o usuário ficar apenas focado na jogabilidade ao invés da tecnologia, eliminando muitas vezes dúvidas na pré-produção do jogo (FULLERTON, 2008, tradução nossa).

### 2.2.2 Prototipagem em Papel

A prototipagem em papel é bastante utilizado para projetar, testar e refinar interfaces de usuário. Em meados da década de 1990, empresas como IBM, Digital, Honeywell e Microsoft experimentaram esta técnica e acharam útil para usarem como parte no processo de desenvolvimento dos seus produtos, mais tarde outras empresas de pequeno e médio porte também passaram a usa-la como prática comum em suas atividades (SNYDER, 2003, tradução nossa).

### 2.2.3 Prototipagem Digital

Assim como os protótipos físicos, os protótipos digitais são criados utilizando apenas as partes essenciais pelos quais os tornarão elementos funcionais. Eles não são jogos completos, logo, são feitos com artes minimalistas e efeitos sonoros simples (FULLERTON, 2008, tradução nossa).

Para Eric Todd, diretor de desenvolvimento do jogo Spore, o processo de prototipação é dividido em:

a) mecânicas do jogo: são características discretas dos aspectos formais do jogo e é importante sempre deixar o mais simples possível e focar em apenas uma particularidade, ou seja, não se deve generalizar muitas funcionalidades ao mesmo tempo;
b) estéticas: constitui dos elementos visuais e auditivos;
c) cinestéticas: são as “sensações” do jogo como a precisão dos controles, o quão receptivo a interface é, entre outros;
d) tecnologia: inclui as capacidades gráficas do jogo, sistemas de Inteligência Artificial (IA), a física ou qualquer problema específico (FULLERTON, 2008, tradução nossa).

A relação da prototipação com o *game design* é necessária para que se tenha um bom resultado final, além de um jogo muito mais imersivo e funcional concentrando-se no jogador. O uso da prototipagem deve ser tratado como um requisito essencial para um *game designer* e todos os criadores envolvidos no projeto (SATO, 2010).

# 3 FRAMEWORK PARA JOGOS

No desenvolvimento de aplicações, um *framework* é um conjunto cooperativo de classes que compõem um projeto reutilizável para um domínio específico de softwares. Ele fornece orientação arquitetural através da divisão do projeto em classes abstratas e definindo suas responsabilidades e colaborações. Um desenvolvedor customiza o *framework* para uma aplicação específica utilizando instâncias das classes do mesmo (GAMMA et al., 1994, tradução nossa).

Campo (1997, p. 32) salienta de uma maneira informal que “um *framework* pode ser considerado como uma infraestrutura de classes que fornecem o comportamento necessário para implementar aplicações dentro de um dado domínio”.

Foram selecionados para esta pesquisa três *frameworks* diferentes para o desenvolvimento de jogos 2D, são eles: Libgdx, Phaser e Cocos2d-x.

## 3.1 LIBGDX

Libgdx é um *framework* de código aberto (*open source*) multiplataforma utilizado principalmente para o desenvolvimento de jogos eletrônicos através da linguagem de programação Java. Seu principal destaque é conseguir rodar e debugar o código no desktop como uma aplicação nativa. Desde seu lançamento da versão 0.1 em março de 2010, muito trabalho foi feito para melhorar ainda mais esta biblioteca pois conta com uma comunidade ativa e em constante crescimento (NAIR; OEHLKE, 2015, tradução nossa).

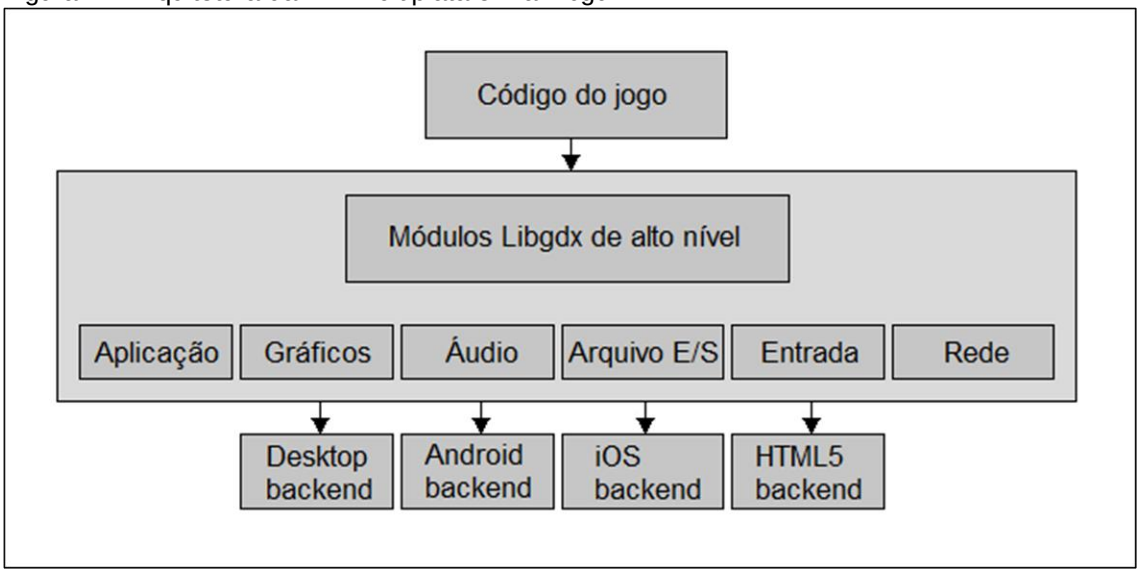
Nair e Oehlke (2015, tradução nossa) enaltecem que o *framework* consegue abstrair a natureza de todas as plataformas combinando-as em uma única Interface de Programação de Aplicações, do inglês *Application Programming Interface* (API). Ainda que não é considerada um motor de jogos (*game engine*), a Libgdx permite o usuário a usar de forma espontânea, bibliotecas de mais baixos níveis e fazer chamadas ao sistema à medida que for necessário.

Marquez e Sanchez (2014, tradução nossa) afirmam que a Libgdx possui uma API multiplataforma pelos quais os usuários podem escrever uma única vez sua aplicação através da linguagem Java e distribuir para todas as plataformas

suportadas. Cada plataforma dispõe de um *backend* que implementa subsistemas de baixo nível, que são: Aplicação, Gráficos, Áudio, Entrada, Arquivos e Rede.

A figura 2 apresenta a arquitetura principal da API da Libgdx destacando suas plataformas e seus respectivos subsistemas.

Figura 2 – Arquitetura da API multiplataforma Libgdx




Fonte: Marquez e Sanchez (2014, tradução nossa).

Segundo Marquez e Sanchez (2014, tradução nossa) para facilitar a distribuição multiplataforma, a API da Libgdx realiza algumas tarefas básicas nas seguintes plataformas:

a) desktop backend: depende principalmente da biblioteca *Lightweight Java Game Library* (LWJGL) para funcionar trabalhando em cima do OpenGL para efetuar as chamadas ao sistema;
b) android backend: para o desenvolvimento Android é realizada a busca nas versões oficiais do Android SDK além da versão integrada do sistema OpenGL, que é chamado OpenGL ES;
c) ios backend: traduz os *bytescodes* Java em código de máquina ARM ou x86 utilizando a abordagem de desenvolvimento do compilador RoboVM baseado na compilação antes do tempo (*ahead-of-time compilation*);
d) html5 backend: em aplicações html5, é exibido sistemas extremamente personalizáveis através das tecnologias WebGL e Javascript dado que há compatibilidade com navegadores através do

*Google Web Toolkit* (GWT) que compila todo o código Java em Javascript otimizado.

### 3.1.1 Ciclo de vida

Uma aplicação do *framework* Libgdx possui um ciclo de vida através de um sistema de estados bem definidos, esses estados são procedimentos implementados pela interface *ApplicationListener* conforme representado na figura 3, pelos quais cuidará de toda a parte lógica do jogo (NAIR; OEHLKE, 2015, tradução nossa).

Figura 3 – Interface ApplicationListener

```
public interface ApplicationListener {
   public void create ();
   public void resize (int width, int height);
   public void render ();
   public void pause ();
   public void resume ();
   public void dispose ();
}
```

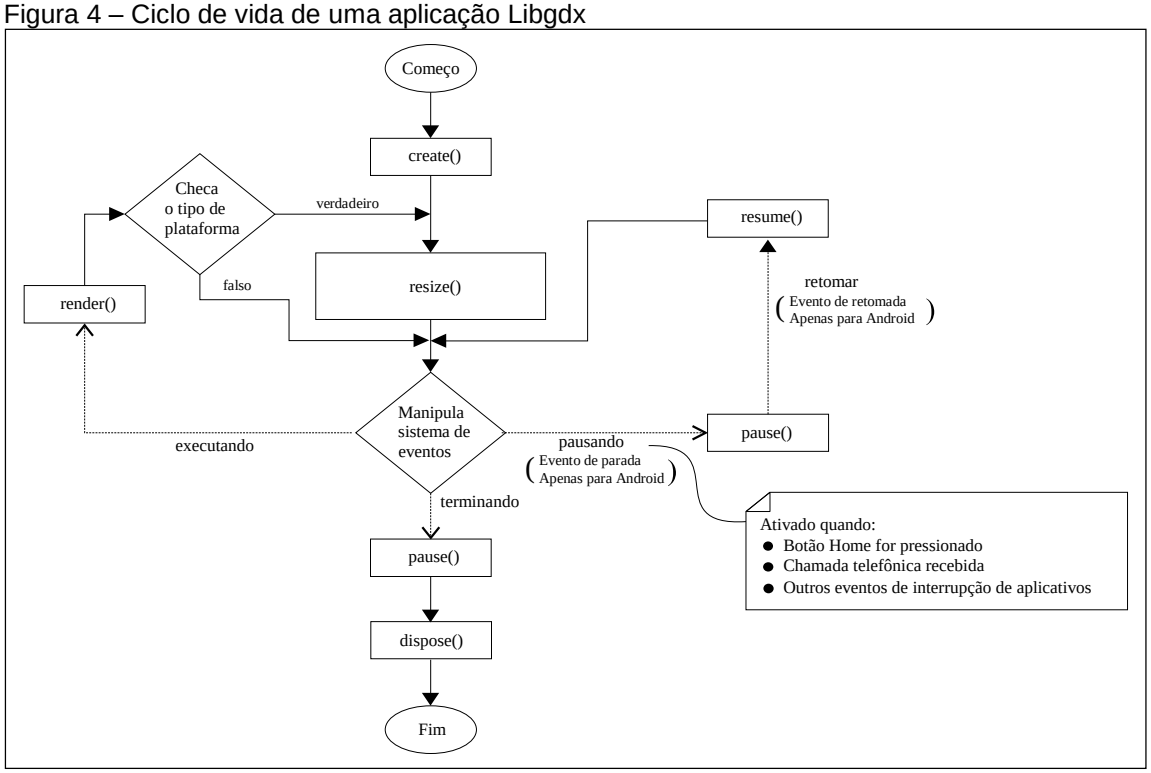
Fonte: Adaptado de Nair e Oehlke (2015).

Marquez e Sanchez (2014, tradução nossa) afirmam que cada estado se refere a um evento específico que a interface pode responder. Para cada método ou estado, é designado uma função específica do ciclo de vida da Libgdx que são:

a) create(): usado para inicializar subsistemas e carregar recursos;
b) resize(): utilizado para configurar um novo tamanho de tela que pode ser aplicado para reposicionar os elementos da *user interface* (UI) ou reconfigurar os objetos da câmera;
c) render(): atualiza e renderiza os elementos do jogo;
d) pause(): este é o estado de salvamento do jogo após ele perder o foco da câmera;
e) resume(): é chamado para alternar do estado pause() para o estado do jogo principal;
f) dispose(): usado para liberar recursos e limpar dados da memória.

Quando uma aplicação inicia, o método create() é chamado onde é feito a inicialização das variáveis, carregamento de dados para a memória e a criação do estado inicial do jogo. Posteriormente, resize() é o segundo método chamado onde será feito o ajuste necessário para que o aplicativo se adapte ao tamanho de exibição disponível, a figura 4 mostra o ciclo de vida desse processo com mais detalhes (NAIR; OEHLKE, 2015, tradução nossa).

Figura 4 – Ciclo de vida de uma aplicação Libgdx




Fonte: Nair e Oehlke (2015, tradução nossa).

## 3.2 PHASER

Criado por Richard Davey, a *framework* Phaser é usada para criar jogos 2D utilizando Javascript como sua linguagem de programação padrão e teve forte inspiração do motor de jogos chamado Flixel que foi bastante popular no mundo do desenvolvimento de jogos em *flash*. Possui uma comunidade grande de desenvolvedores, com suporte a animações, sons, diferentes motores para física e partículas (FAAS, 2016, tradução nossa).

De acordo com Afonso (2017), Phaser é uma ferramenta com o foco em usuários com conhecimentos em programação já que não possui qualquer tipo de interface gráfica para a construção do projeto. São utilizados tecnologias da web

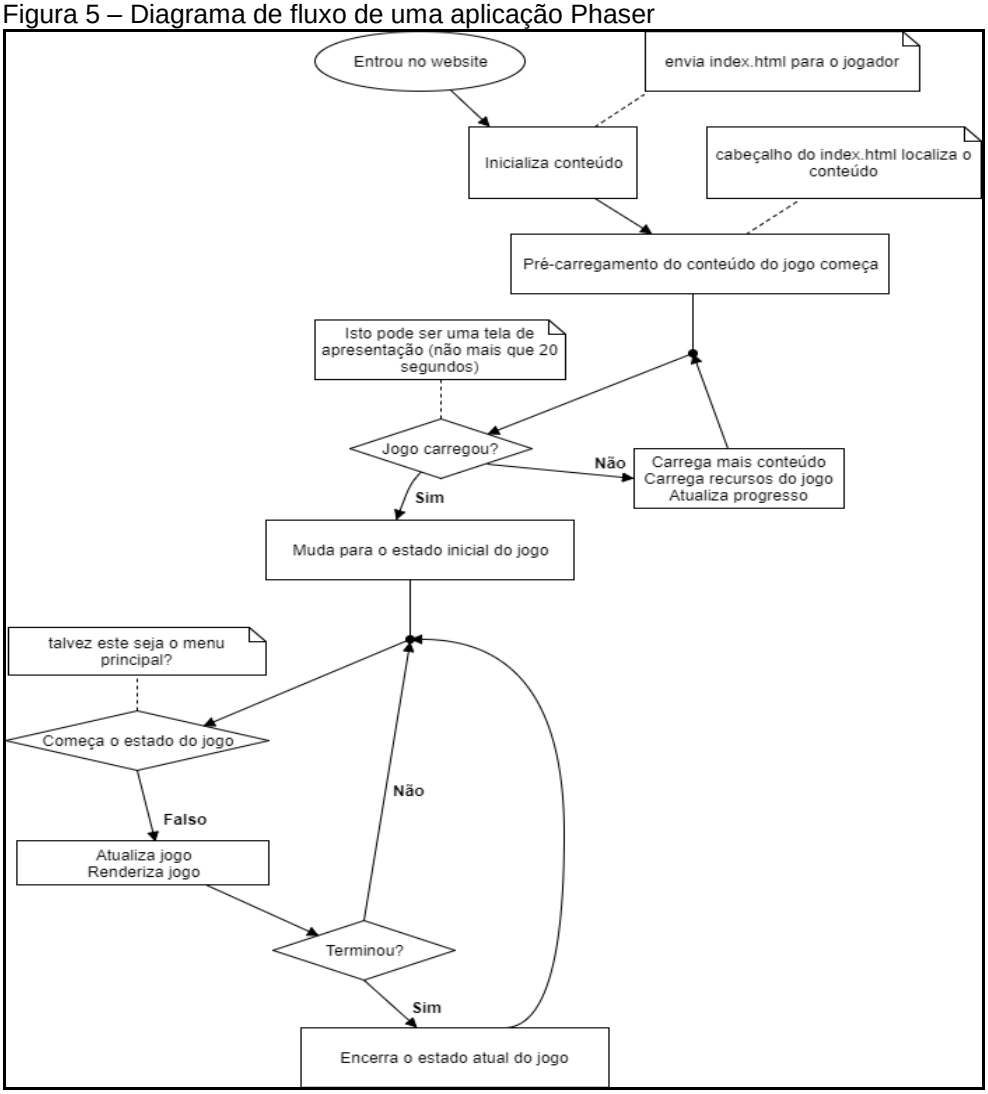
como WebGL e o Canvas para geração do jogo e está frequentemente a receber atualizações pelo fato de ser uma ferramenta *open source*.

Outra característica notada é a opção de alternar entre aceleração de hardware e modo WebGL para renderização do cenário. A renderização dos objetos é realizada através de outro *framework* chamado Pixi.js, que vai tratar toda a parte das animações e renderização das imagens, simplificando as chamadas de gráficos do Javascript e fornecendo uma única API que vai funcionar tanto para WebGL quanto para o canvas (FAAS, 2016, tradução nossa).

### 3.2.1 Fluxo de jogo

Segundo Bibat (2015, tradução nossa), o fluxo do jogo é composto por um pré-carregamento, onde todo o conteúdo do jogo é carregado, a criação onde é inicializado o estado inicial do jogo, a atualização que é disparada em um certo intervalo de tempo (geralmente 60 vezes por segundo) e a renderização que vem logo após a atualização. A figura 5 apresenta o fluxo completo de uma aplicação Phaser.

Figura 5 – Diagrama de fluxo de uma aplicação Phaser




Fonte: Gose (2016, tradução nossa).

## 3.3 COCOS2D-X

Cocos2d-x é uma ramificação do popular *framework* para jogos de sistema operacional IOS chamado Cocos2d. Seu primeiro lançamento foi em novembro de 2010 e mais tarde comprada pela *Chukong Technologies* em 2011 e é mantida até hoje por uma comunidade de mais de 400.000 desenvolvedores em todo o mundo (HERNANDEZ, 2015, tradução nossa).

Esta é uma ferramenta *open source* bastante popular que utiliza para o seu desenvolvimento três linguagens principais que são C++, Lua ou Javascript. Seu principal destaque é a possibilidade de exportar os projetos para diversas plataformas incluindo desktop e mobile utilizando apenas uma linguagem de programação (HUSSAIN; GURUNG; JONES, 2014, tradução nossa).

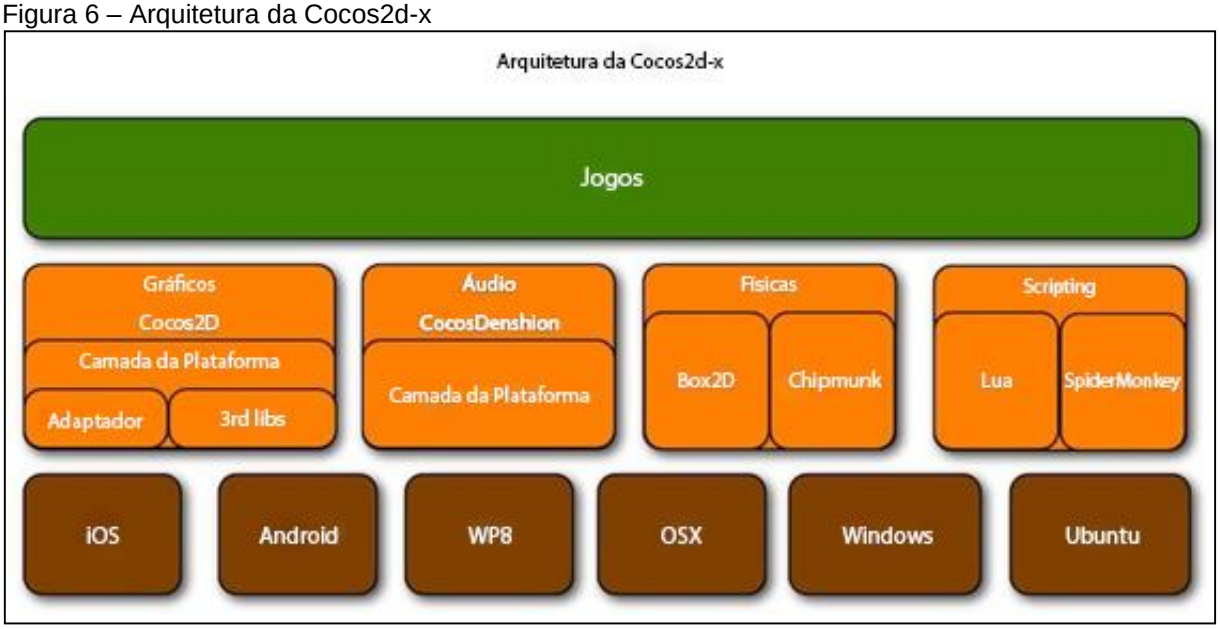
A Cocos2d-x suporta a criação de jogos 2d fornecendo várias funcionalidades, como gestão de cenas, efeitos visuais, partículas, animações baseadas em esqueletos entre outros (ABDUL-KARIM, 2014, tradução nossa).

Hernandez (2015, tradução nossa) menciona que a Cocos2d-x é capaz de encapsular todas as particularidades do jogo como som, música, física, entradas do usuário, *sprites*, cenas e transições. Portanto basta o desenvolvedor se concentrar na lógica do jogo invés de implementar todas as mecânicas partindo do zero.

### 3.3.1 Visão Geral da Arquitetura

A arquitetura do Cocos2d-x é dividida em três camadas principais que consistem no jogo ou aplicação, os componentes pelos quais são fornecidos pela própria ferramenta e a sua respectiva plataforma conforme ilustrada na figura 6 (ABDUL-KARIM, 2014, tradução nossa).

Figura 6 – Arquitetura da Cocos2d-x




Fonte: Abdul-karim (2014, tradução nossa).

A camada ilustrada de cor verde, representa o jogo criado pelos desenvolvedores, a laranja possui todas as funcionalidades que podem ser inseridas no projeto como gráficos, áudio, físicas e *scripts*. Por último, a camada marrom

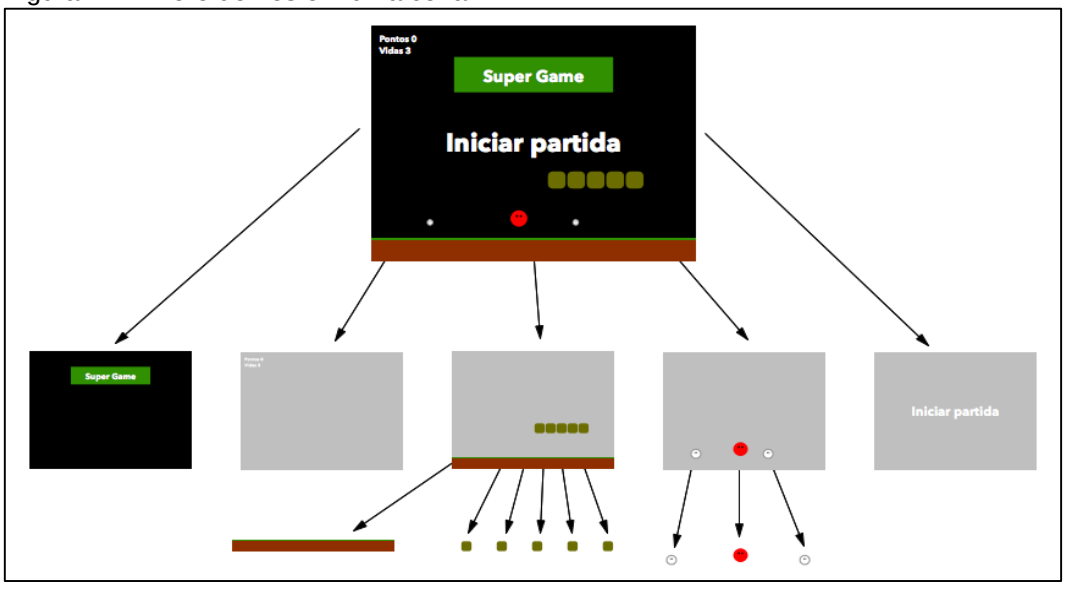
corresponde as plataformas que o jogo será executado (ABDUL-KARIM, 2014, tradução nossa).

### 3.3.2 Modelagem dos elementos

O Cocos2d-x representa todos os elementos da tela em uma estrutura de árvores mais conhecido como grafo de cena, conforme mostrado na figura 7. Os elementos que compõem esta árvore são os nós, onde a sequência de desenho na tela leva em consideração a ordem de inserção destes nós e pelo algoritmo de caminhamento da árvore, que é implementado como um caminhamento em ordem (ANDRADE; KNOP, 2015).

De acordo com Andrade e Knop (2015) todos os objetos da cena também são nós, ou seja, herdam do nó principal, já uma camada é um tipo de nó pelos quais possui o mesmo tamanho da tela e tem como objetivo agrupar outros nós geralmente com as mesmas funções. Além disso, um nó é capaz de executar diversas ações como por exemplo: se movimentar até um certo destino, rotacionar, modificar a escala e entre outras funções.

Figura 7 – Árvore de nós em uma cena




Fonte: Andrade e Knop (2015).

## 3.4 OPEN SOURCE

O termo *open source* ou código aberto é um software pelo qual pode ser executado, copiado, distribuído, modificado e aperfeiçoado pelos seus usuários segundo Projeto Software Livre Brasil. Eles podem ser desenvolvidos por pessoas em todo o mundo formando assim uma comunidade, não importando o tipo de software produzido (IWASAKI, 2008).

## 3.5 FERRAMENTAS MULTIPLATAFORMA

Com o aumento da quantidade de plataformas disponíveis hoje no mercado, dificultou-se a complexidade no desenvolvimento pois é necessário que uma aplicação atenda todos os dispositivos, assim como, possuir conhecimentos distintos de cada ferramenta. Porém, graças os *frameworks* podemos desenvolver aplicações únicas, utilizando de apenas uma linguagem de programação e ela ficará responsável por transformar o código-fonte na linguagem da plataforma correspondente, o que é conhecido como aplicações híbridas (PREZOTTO; BONIATI, 2014).

## 3.6 VANTAGENS E DESVANTAGENS NO USO DE FRAMEWORKS

O mercado de jogos digitais vem crescendo cada vez mais no mundo inteiro, os jogos independentes (*indie games*) quanto jogos AAA (com orçamento de milhões de dólares) demandam muito dinheiro não só para o seu desenvolvimento, mas também para a criação de campanhas de marketing e anúncios realizados em feiras de jogos para o aumento das vendas (MADEIRA, 2001).

O uso de *frameworks* acaba minimizando o tempo gasto usado para o desenvolvimento e tendo como fator principal a boa organização dos projetos e facilidade de produção do mesmo (FIGUEIREDO, 2014).

Sendo assim, o desenvolvimento de aplicações a partir do zero acaba sendo um processo muito demorado, além de aumentar a probabilidade de ocorrer eventuais problemas no futuro.

## 3.7 DIFERENÇAS E SEMELHANÇAS ENTRE FRAMEWORKS E MOTORES DE JOGOS

Existem muitas controvérsias entre a definição de *framework* e motor de jogo, muitos leigos no assunto acabam confundindo *frameworks* com motores de jogos. Infelizmente não há uma definição clara e abrangente, pois, muitos autores relatam inúmeras informações e características com contextos contrários, mas possuem algumas diferenças e semelhanças que é importante citar nesta pesquisa.

Segundo Gregory (2009, tradução nossa) o termo motor do jogo (*game engine*) surgiu em meados da década de 1990 em referência aos jogos de tiro em primeira pessoa, do inglês *First Person Shooter* (FPS). Onde foram construídos com uma separação razoavelmente bem definida entre seus principais componentes de software (como o sistema de renderização de gráficos tridimensional, o sistema de detecção de colisão ou o sistema de áudio) e recursos de arte, cenários e regras de jogo que incluíam a quantidade de experiência do jogador.

## 3.8 DETECÇÃO DE COLISÃO

Em jogos digitais, a detecção de colisão se refere ao problema computacional de verificar a sobreposição entre dois ou mais objetos em um dado intervalo de tempo. A resolução da detecção de colisão se dá a partir do que acontece após a detecção, como por exemplo reduzir a velocidade de um objeto para zero ou trocar de direção. A detecção e a colisão são problemas totalmente interligados, pois para que haja colisão é necessário primeiro que se tenha uma detecção (PYDD, 2015).

### 3.8.1 Algoritmos de detecção de colisão

Algoritmos de detecção de colisão são muito utilizados em jogos digitais para efetuar qualquer simulação do mundo físico pelo qual vem sendo estudado pela robótica, a computação gráfica e a geometria computacional (KERA, 2005).

Para realizar o cálculo de detecção de colisão existem dois algoritmos que são o da sobreposição de retângulos e o da distância euclidiana.

### 3.8.1.1 Sobreposição de retângulos

De acordo com Pydd (2015) este é o método primordial de detecção de colisão, para realizar a verificação, testa-se a coordenada x do primeiro retângulo e examina se está entre o x do segundo retângulo e o x + largura do segundo retângulo. Mesma coisa com a coordenada y, testa-se o y do primeiro retângulo está entre do y do segundo retângulo e o y + altura do segundo retângulo. A figura 8 demonstra o pseudocódigo do algoritmo.

Figura 8 – Pseudocódigo da sobreposição de retângulos

```
Funcao Colisao {

  retangulo1, retangulo2

  se ((retangulo1.X + retangulo1.Largura > retangulo2.X) &&
  (retangulo1.X < retangulo2.X) &&
  (retangulo1.Y +  retangulo1.Altura > retangulo2.Y) &&
  (retangulo1.Y < retangulo2.Y))
     //houve colisao
  senao
     //nao houve colisao
}
```

Fonte: Pydd (2015).

### 3.8.1.2 Distância euclidiana

Nesse algoritmo de detecção de colisão, dois objetos são envoltos de um círculo de tamanho específico, após isso, é necessário calcular dois elementos: a distância entre o centro dos dois círculos e os raios dos mesmos. A distância entre os centros dos círculos pode ser calculada utilizando a fórmula da distância euclidiana (PYDD, 2015):

$$\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

É preciso saber se a distância euclidiana entre os dois centros de círculos é maior que a soma do raio dos mesmos, conforme ilustrado na figura 9.

Figura 9 – Pseudocódigo da distância euclidiana

```
Funcao Colisao {

 circulo c1, circulo c2
 raio1, raio2

 distancia = raizquadrada((c1.x - c2.x)^2 + (c1.y - c2.y)^2)
 se (distancia > (raio1 + raio2))
   //nao houve colisao
 senao
   //houve colisao
}
```

Fonte: Pydd (2015).

Caso a distância entre os círculos for menor do que a soma dos raios, significa que houve colisão, caso contrário a colisão não aconteceu.

# 4 MÉTRICAS DE SOFTWARE

As métricas são um padrão de medidas importante para verificar a eficiência e as funcionalidades de diversas atividades no desenvolvimento de software. Elas em si não melhoram a produtividade, porém fornecem informações relevantes que podem ser usadas para essa melhoria. Através de um programa de medição de testes é possível identificar os dados que precisam ser coletados, como serão usados e devem ser especificados a cada fase do ciclo de testes realizado (TRODO, 2009).

Portanto, as métricas de software podem ser divididas em duas categorias: métricas diretas e indiretas.

## 4.1 MÉTRICAS DIRETAS E INDIRETAS

De acordo com Pressman (2011) as métricas no mundo físico podem ser classificadas em métricas diretas e indiretas. As métricas diretas em relação ao processo de software incluem custos, trabalho aplicado e linhas de código, do inglês *Lines of Code* (LOC), produzidas além de velocidade de execução, tamanho de memória e defeitos relatados durante um certo período de tempo.

Considerado o método mais trabalhoso, podemos aplicar medidas indiretas para a avaliação da qualidade, complexidade, eficiência, contabilidade, manutenção e as funcionalidades de um software (CORDEIRO, 2001; PRESSMAN, 2011).

### 4.1.1 Métricas orientadas a tamanho

As métricas orientadas a tamanho são criadas baseadas no tamanho do software produzido compondo em medidas de qualidade ou produtividade. Caso uma organização de software possuí registros simples, pode-se criar uma tabela contendo cada projeto a ser desenvolvido onde inclui todo trabalho e custo representando todas as atividades de engenharia de software (análise, projeto, código e teste). Métricas orientadas a tamanho não são consideradas a melhor

maneira de se medir os processos de software por se utilizar linhas de código como medida principal (PRESSMAN, 2011).

Esta medida de software é a mais simples e familiar, por ter o objetivo de contar apenas a quantidade de linhas de código do projeto desenvolvido. Existem alguns problemas em relação a este tipo de métrica como o uso de uma linguagem específica de programação fortemente ligado ao software que está sendo produzido, impossibilitando que se utilize os mesmos dados adquiridos em outro projeto que use uma linguagem diferente (CORDEIRO, 2001).

### 4.1.2 Métricas orientadas a função

As métricas orientadas a função tem como objetivo de se concentrar nas funcionalidades do software ao invés de apenas contar as linhas do código fonte. A métrica orientada a função mais usada é a de pontos de função (*function point* - FP). Para realizar o cálculo por pontos de função é necessário ter como base as propriedades de controle de informação e a complexidade do software (PRESSMAN, 2011).

#### 4.1.2.1 Análise por Pontos de função

Em 1979, Allan Albrecht que trabalhou na IBM, introduziu uma técnica de avaliação conhecida como Pontos de função pelo qual permitia calcular o esforço de programação e melhorar a avaliação de projetos (CORDEIRO, 2001).

A análise por pontos de função é uma métrica orientada a função que tem como objetivo medir a funcionalidade do sistema baseado na visão do usuário tendo como características a independência da tecnologia utilizada, ou seja, podemos agora realizar a comparação de sistemas com linguagem de programação diferentes e o apoio para a tomada de decisão e contratação de serviços (DIAS, 2003).

Tanto os métodos antigos de medição por linhas de código quanto pela análise por pontos de função possuem certas características descritas no quadro 1.

Quadro 1 – Comparação das técnicas apresentadas

| **Características** | **Linhas de Código** | **Pontos de Função** |
|---|---|---|
| Independência de tecnologia | Não | Sim |
| Resultados consistentes | Sim | Sim |
| Visão do usuário | Não | Sim |
| Significância para o Usuário final | Não | Sim |
| Utilização em estimativas | Não | Sim |
| Passível de automação | Sim | Sim |

Fonte: Adaptado de Dias (2003).

### 4.1.3 Métrica para avaliação de funcionalidades

A métrica para avaliação das funcionalidades entre motores de jogos é constituída de certos fatores e é classificada em cinco categorias:

a) gráficas: são os recursos que o motor dispõe para renderização do ambiente;
b) acessórias: leva em consideração a simulação física, a conexão de rede e a inteligência artificial;
c) suporte: corresponde a documentação da ferramenta, suporte e as atividades da comunidade de desenvolvedores e utilizadores;
d) software: inclui o tipo de sistema operacional e a linguagem de programação utilizada;
e) aplicações: tipo de aplicação a ser utilizado como simulação ou entretenimento (RIBEIRO; SANTOS, 2009).

Os critérios para avaliação das funcionalidades são definidos para auxiliar em determinar qual motor de jogo é mais adequado. Cada critério de avaliação é definido conforme mostrado no quadro 2, onde a sigla AC significa *assessment criterion* (Critério de avaliação) e XXX é um índice único para cada referência. O grau de importância é descrito seguindo o padrão de acordo com o quadro 3 (EVES; MEEHAN, 2008, tradução nossa).

Quadro 2 – Definição do critério para avaliação

| AC-XXX | Importância |
|---|---|
| O texto do critério | |

Fonte: Eves e Meehan (2008, tradução nossa).

Quadro 3 – Critério e sua importância

| Importância | Definição |
|---|---|
| Essencial | Esse critério deve ser satisfatório. Sua ausência inibiria o uso do motor do jogo para alcançar a funcionalidade necessária e características de um jogo. |
| Altamente desejável | Este critério é altamente desejável. Sua ausência reduz a extensão e facilita em usar o motor de jogo para alcançar a funcionalidade necessária e características de um jogo. |
| Desejável | Esse critério é desejável. Sua ausência reduz ligeiramente a extensão e facilidade de usar o motor de jogo para alcançar a funcionalidade necessária e as características de um jogo. |

Fonte: Eves e Meehan (2008, tradução nossa).

Para cada motor de jogo, uma pontuação é combinada com o nível do critério variando de 1 a 3 como é representado no quadro 4 (EVES; MEEHAN, 2008, tradução nossa).

Quadro 4 – Ponderação da avaliação

| Critério | Peso |
|---|---|
| Essencial | 3 |
| Altamente desejável | 2 |
| Desejável | 1 |

Fonte: Eves e Meehan (2008, tradução nossa).

Com base nestas pontuações, é possível avaliar cada funcionalidade de uma ferramenta para desenvolvimento de jogos acompanhado do grau de criticidade conforme mencionado anteriormente.

# 5 TRABALHOS CORRELATOS

Há outras formas de análises semelhantes a esta no ramo dos jogos digitais, neste capítulo serão apresentados trabalhos e pesquisas referentes a estudos relacionados a comparações entre tecnologias da computação e também motores de jogos para enriquecer ainda mais o trabalho proposto.

## 5.1 UMA ANÁLISE COMPARATIVA DE FRAMEWORKS PARA DESENVOLVIMENTO DE JOGOS NO ANDROID

O artigo publicado *no International Conference on Information Systems and Technology Management* por Pedro Henrique Pereira et al. (2017), realizou a avaliação de dois *frameworks* para o desenvolvimento de jogos para Android com a finalidade de apresentar qual seria mais eficiente tendo como base uma pesquisa do tipo exploratória.

Em consequência disso, com o objetivo de ajudar a comunidade de desenvolvedores na escolha da ferramenta adequada, fez-se necessário descrever as funcionalidades de cada *framework*, componentes, elementos, características e limitações.

Por meio da análise realizada, conclui-se que nenhum dos *frameworks* tirou total vantagem um sobre o outro pois cada um teve seus pontos positivos e negativos e para que seja feita a escolha apropriada é preciso atender a certos fatores como: o tamanho da aplicação, a exclusividade com o sistema Android, tamanho da equipe de desenvolvimento e os recursos que o jogo deverá conter.

## 5.2 ANÁLISE COMPARATIVA DE FRAMEWORKS DE PERSISTÊNCIA

Foi desenvolvido um trabalho na Universidade Federal de Lavras, por Raphael Barreto Palhares de Campos, onde visou a comparação de *frameworks* de persistência pelos quais foram definidos critérios gerais e específicos tendo como base uma pesquisa bibliográfica sobre o tema.

Além disso, foram realizados testes comparativos utilizando uma configuração específica de computador e foram coletados o tempo de inicialização

do sistema, geração do esquema do banco de dados e tempo para execução de um teste de entidade.

Com base nos resultados obtidos, foi possível observar que os *frameworks* tiveram soluções parecidas, os critérios utilizados foram de recursos adicionais simples pelos quais auxiliam ao desenvolvedor.

## 5.3 COMPARATIVO ENTRE GAME ENGINES COMO ETAPA INICIAL PARA O DESENVOLVIMENTO DE UM JOGO DE EDUCAÇÃO FINANCEIRA

O artigo publicado no III Congresso sobre Tecnologias na Educação (Ctrl+E 2018) por Cavalcante e Pereira (2018), realizou a análise comparativa entre os motores de jogos Unity, Godot e Phaser para o desenvolvimento de jogos apresentando suas principais características e os pontos fortes e fracos de cada uma.

A primeira etapa a ser realizada foi a análise das *Integrated Development Environment* (IDE) de cada uma das *engines*, onde foi realizado um estudo e coletadas suas principais características. Na segunda etapa, desenvolveu-se um jogo, unindo várias funcionalidades de cada *engine*. A última etapa consistiu em apresentar as vantagens e desvantagens de cada motor de jogo na visão do autor do trabalho.

Segundo o autor, após a realização do experimento, constatou que foi possível atingir o objetivo do trabalho, onde se pretendia determinar a escolha de uma *engine* mais adequada e que facilitasse o desenvolvimento do jogo a ser implementado.

## 5.4 ANALISE COMPARATIVA DAS FUNCIONALIDADES DE FERRAMENTAS PARA A CRIAÇÃO DE JOGO EDUCACIONAL

A pesquisa desenvolvida por Nascimento, Simões e Lima no Instituto Federal Norte de Minas Gerais objetivou a comparação de ferramentas para desenvolvimento de jogos educacionais através de um quadro indicativo das funcionalidades presente em cada uma das ferramentas analisadas.

A comparação foi feita seguindo o conceito de especificação de software introduzido por Sommerville onde é realizado a definição das funcionalidades (requisitos) e das restrições do software. Portanto, a comparação tem como foco identificar quais funcionalidades seriam de suma importância para o “Cliente”.

Como resultado obtido, pode-se afirmar que as escolhas das ferramentas para desenvolvimento dependem da proposta do jogo e de fatores relacionados a equipe de desenvolvedores, tais como a familiaridade com o material de estudo disponível e o tempo de aprendizado das ferramentas escolhidas.

## 5.5 ANÁLISE DE GAME ENGINES PARA PLATAFORMAS MÓVEIS

Foi realizada uma pesquisa na Universidade Federal da Paraíba por Rennan Araújo Barbosa no qual foi feito a avaliação de motores de jogos para dispositivos móveis com o propósito de ajudar os desenvolvedores a escolher a que atenda suas necessidades.

Para executar a análise, levou-se em consideração a arquitetura proposta por Gregory (2009) onde buscavam generalizar os componentes do motor de jogo e descrever suas funcionalidades. Para verificar a presença dos componentes, foi realizado uma pesquisa na documentação dos motores, quanto mais componentes o motor possuir, mais benefícios estarão disponíveis para o desenvolvedor, logo, a que será eleita a mais viável para o desenvolvimento.

De acordo com os dados levantados, conclui-se que o motor de jogo Unity apresenta ter todos os componentes disponíveis segundo a tabela comparativa, sendo assim, foi estimada a que possui maior capacidade para auxiliar os desenvolvedores.

# 6 ANÁLISE COMPARATIVA DE FRAMEWORKS OPEN SOURCE PARA O DESENVOLVIMENTO DE JOGOS MULTIPLATAFORMA

Neste trabalho, foram desenvolvidos três protótipos equivalentes em cada *framework* utilizando o mesmo número de elementos e características semelhantes, tendo como objetivo principal comparar diferentes resultados.

Como resultados obtidos, foram avaliadas as funcionalidades de cada *framework* com base na métrica de avaliação de funcionalidades onde é feito a definição do critério para avaliação, sua importância e a ponderação da avaliação.

Não foram utilizadas as métricas orientadas a tamanho e a função, pois os protótipos foram desenvolvidos em linguagens de programação diferentes (Java e Javascript), assim, o mesmo exigia a contagem de linhas de código e o cálculo de pontos de função pelo qual não foi necessária sua realização.

## 6.1 METODOLOGIA

Para alcançar os objetivos do trabalho, as seguintes etapas metodológicas foram realizadas: levantamento bibliográfico, planejamento do cenário do jogo com os elementos essenciais, avaliação de desempenho em detecção de colisão, avaliação das funcionalidades de motores de jogos e a medição da média de *frames* por segundo.

A pesquisa bibliográfica proporcionou que na fundamentação teórica fossem abordados conceitos sobre métodos de desenvolvimento em jogos digitais, assim como a formulação para a construção de um protótipo digital.

O projeto se iniciou com o planejamento do cenário do jogo e os elementos necessários para posterior avaliação, em seguida, o desenvolvimento dos protótipos foram realizados.

Com relação ao planejamento do cenário do jogo, foram utilizados objetos minimalistas pelos quais farão parte dos elementos do cenário.

Para a avaliação de desempenho foi realizado o cálculo do tempo em detecção de colisão utilizando o algoritmo de sobreposição de retângulos e como resultado, a média aritmética do tempo obtido.

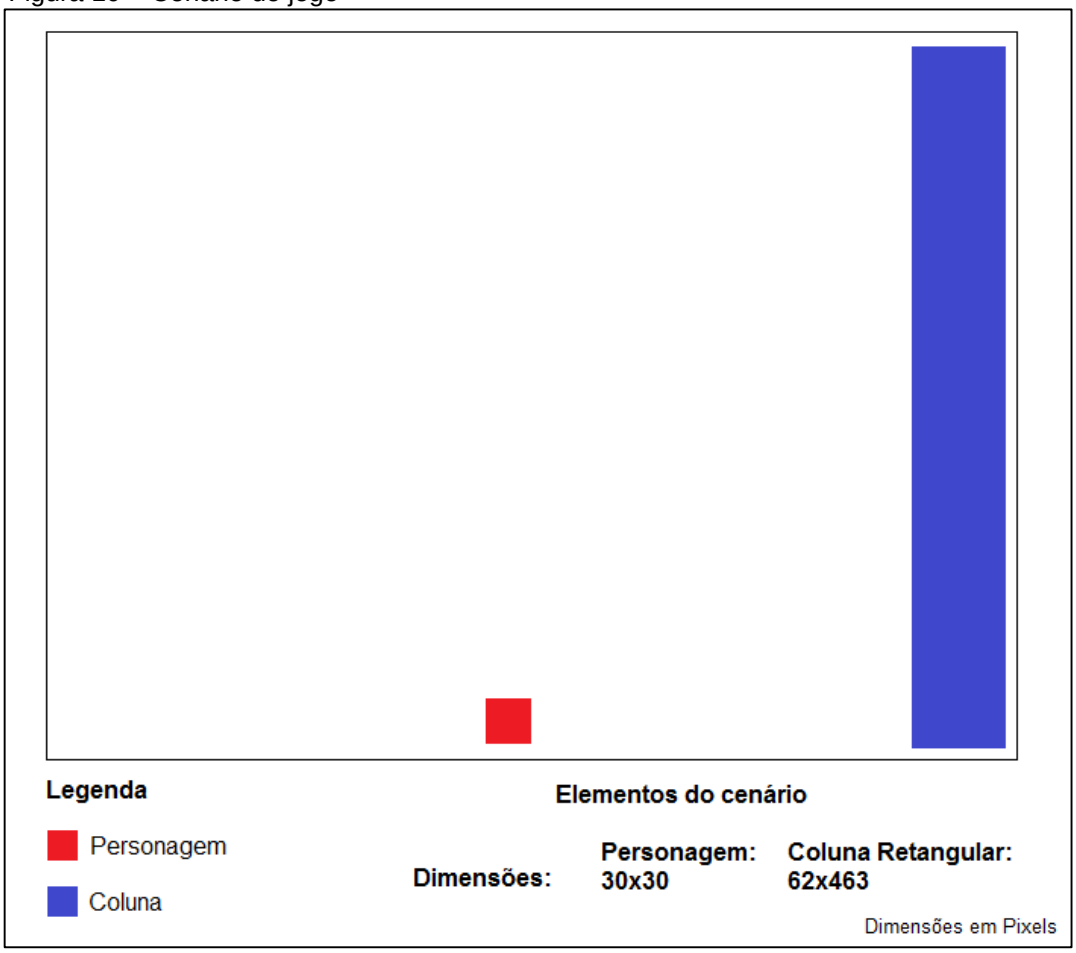
A fim de realizar a comparação entre os três frameworks, executou-se a avaliação das funcionalidades baseando-se em critérios de avaliação.

Ao final, efetuou-se a medição da média de *frames* por segundo realizados nos três protótipos.

### 6.1.1 Planejamento do cenário

Para o desenvolvimento do trabalho, a primeira etapa consistiu no planejamento do cenário. Os elementos que compõem o cenário do jogo foram baseados em objetos minimalistas onde terão as características principais para realizar a análise e os testes de desempenho conforme figura 10.

Figura 10 – Cenário do jogo




Fonte: Do autor.

A resolução adotada para a construção do cenário foi de 640x480px (640 pixels de largura e 480 pixels de altura), a fim de fazer com que o mesmo se adeque a qualquer plataforma.

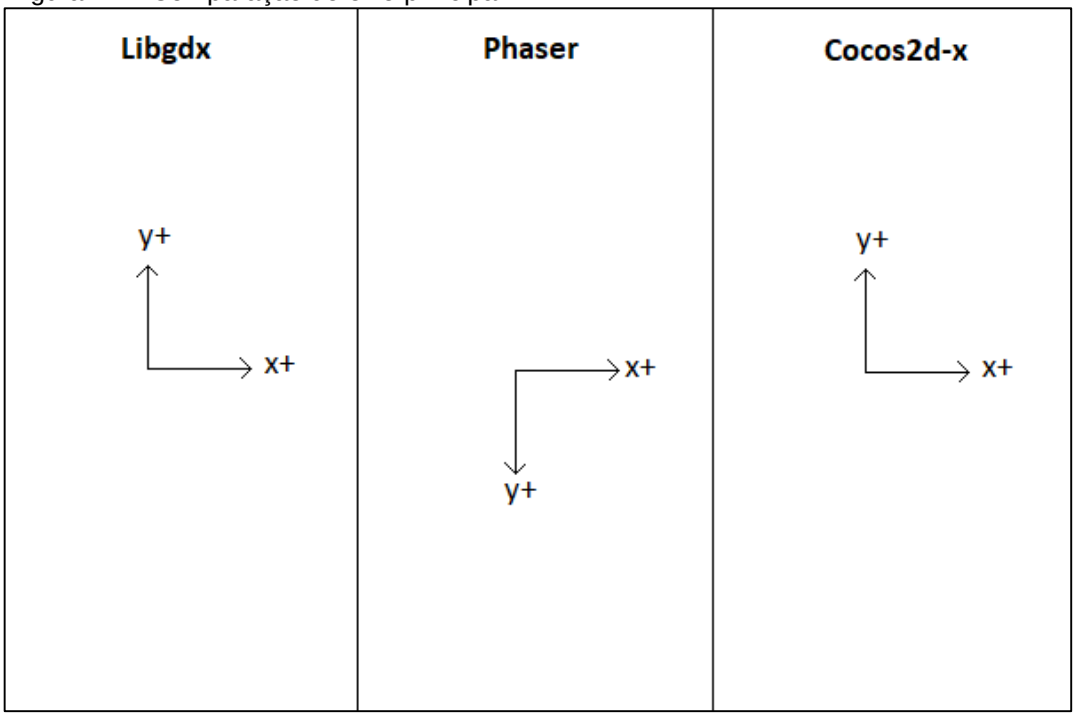
6.1.1.1 Elementos presentes no cenário

Para a criação dos elementos do cenário, adotou-se os seguintes objetos:

a) um personagem de 30 pixels de largura e 30 pixels de altura pelos quais tem a possibilidade de movimentação para esquerda e direita;
b) uma coluna de 62 pixels de largura e 463 pixels de altura;

Cada *framework* possui seu próprio eixo principal variando de x e y como demonstra a figura 11, sendo assim, o *framework* Phaser foi o único que se mantém com seu eixo principal diferente dos demais pois o mesmo foi baseado no eixo dos *joysticks,* variando de -1 (esquerda ou cima) e 1 (direita ou baixo), desta forma, foi designado coordenadas diferentes seguindo conforme o planejamento do cenário.

Figura 11 – Comparação do eixo principal



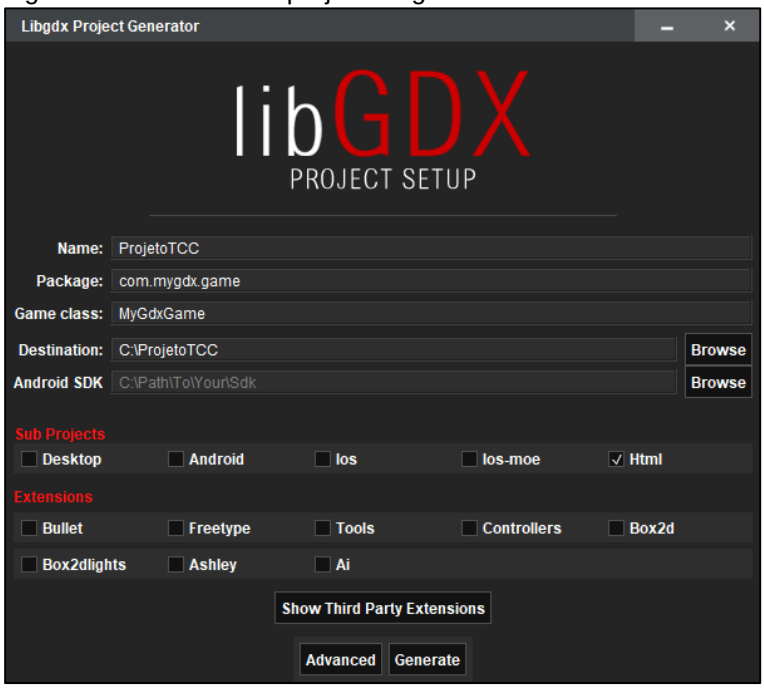

Fonte: Do autor.

### 6.1.2 Desenvolvimento do protótipo no framework Libgdx

Para o desenvolvimento do protótipo no *framework* Libgdx foi necessário realizar sua instalação. Na criação do jogo e configuração inicial foi optado pela IDE Netbeans.

O primeiro passo para a instalação foi o *download* do arquivo java *gdx-setup.jar* disponível no site oficial do *framework*. Em seguida, para a geração do projeto é necessário preencher e marcar algumas informações na interface gráfica disponibilizada pela Libgdx, como: nome da aplicação, nome do pacote, nome da classe principal e o diretório de destino como ilustra a figura 12.

Figura 12 – Gerador de projeto Libgdx

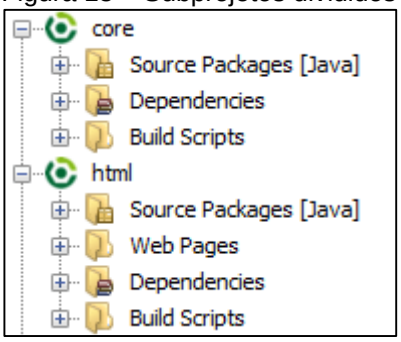



Fonte: Do autor.

O *framework* Libgdx nos permite escolher em quais plataformas e extensões o projeto será gerado. Para fim de comparação, adotou-se o uso exclusivo da plataforma *Html*. A ferramenta tem como principal particularidade o desenvolvimento multiplataforma, portanto se utiliza apenas um código fonte pelo qual possui a opção de exportação para diversas plataformas.

Após o projeto criado, foi necessário a instalação do plugin *graddle* para a leitura do projeto e dividir o diretório raiz em subprojetos conforme figura 13.

Figura 13 – Subprojetos divididos

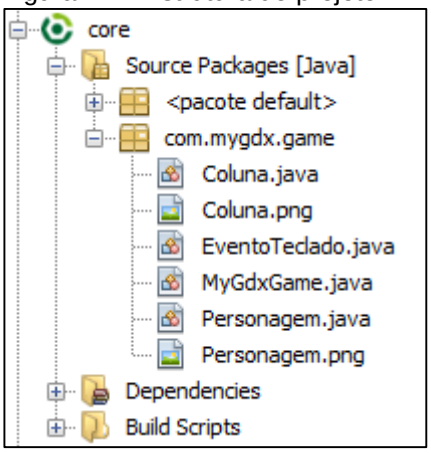



Fonte: Do autor.

O subprojeto *html* é responsável por compilar todo o código java para o javascript através do GWT onde possui as classes necessárias para executar uma aplicação em *html5* nativo. Já o subprojeto *core* contém todo o código da aplicação e todas as plataformas exportadas são referenciadas a este mesmo subprojeto.

A figura 14 demonstra a estrutura do projeto, onde foi inserido no pacote *com.mygdx.game* arquivos de classes java e recursos do jogo como imagens.

Figura 14 – Estrutura do projeto




Fonte: Do autor.

Na estrutura do projeto foram criadas duas classes que representam os elementos do cenário, nominadas de Coluna e Personagem. Para ambas as classes foram implementados atributos do plano cartesiano representado pelas variáveis x, y, largura e altura ilustrado na figura 15.

Para a renderização das texturas dos objetos foi designado os atributos *texturaColuna* e *texturaPersonagem* da classe *Texture* pelos quais são posteriormente carregados pelo método *carregaTextura* utilizando recursos de imagens do jogo.

Por fim, as classes possuem construtores responsáveis por inicializar as coordenadas de x e y, bem como carregar a sua respectiva textura e definir os valores de largura e altura automaticamente.

Figura 15 – Classes coluna e personagem

```
public class Coluna {

    public float x, y;
    private Integer largura, altura;

    private Texture texturaColuna;

    public Coluna() {
        x = 575;
        y = 5;

        carregaTextura();
        setLargura(texturaColuna.getWidth());
        setAltura(texturaColuna.getHeight());
    }
```

```
public class Personagem {

    public float x, y;
    private Integer largura, altura;

    private Texture texturaPersonagem;

    public Personagem() {
        x = 320;
        y = 5;

        carregaTextura();
        setLargura(texturaPersonagem.getWidth());
        setAltura(texturaPersonagem.getHeight());
    }
```

Fonte: Do autor.

A classe EventoTeclado por sua vez, têm o papel de realizar os comandos necessários para efetuar a movimentação do personagem através das setas direcionais do teclado. Para isso, foi necessário implementar duas variáveis booleanas rotuladas de *LEFT_TOUCHED* e *RIGHT_TOUCHED* com o objetivo de mover o jogador utilizando-se como condicionais a partir do método *update*.

A classe principal *MyGdxGame* é responsável por inicializar os elementos do cenário, carregar os recursos do jogo e componentes gráficos como o *SpriteBatch* que por sua vez, tem como a finalidade de desenhar as imagens na tela usando o OpenGL. A implementação desta classe conta ainda, com os métodos *render* onde ela é chamada a todo momento que uma operação de renderização é executada, *verificaColisao,* utilizado para exercer a função da sobreposição de retângulos e por fim, o método *calcularTempo* pelo qual possui as instruções relacionadas ao cálculo do tempo em detecção de colisão.

### 6.1.3 Desenvolvimento do protótipo no framework Phaser

Para começar no desenvolvimento do protótipo no *framework* Phaser realizou-se a preparação dos arquivos do projeto. O primeiro passo foi a configuração da página e a inclusão dos *scripts* requeridos pela ferramenta. As

aplicações criadas no Phaser são executadas nos navegadores através do html5 e é usado como renderização o canvas e a tecnologia WebGL internamente, permitindo o seu funcionamento para dispositivos mobile e desktop.

Para iniciar a codificação do projeto, foi necessário importar inicialmente a biblioteca disponibilizada pelo *framework* através da tag *script* da estrutura do html. Logo após isto, efetuou-se a configuração básica do jogo pela variável *config* como demonstra a figura 16.

Figura 16 – Configuração do protótipo

```
<!DOCTYPE html>
<html>
    <head>
        <title>Projeto TCC</title>
        <script src="//cdn.jsdelivr.net/npm/phaser@3.0.0/dist/phaser.min.js"></script>
    </head>

    <body>
        <script>
            var config = {
                type: Phaser.AUTO,
                width: 640,
                height: 480,
                backgroundColor: '#FFFFFF',
                scene: {
                    preload: preload,
                    create: create,
                    update: update
                }
            };
```

Fonte: Do autor.

Para efetivar a configuração do jogo, adotou-se os seguintes valores para os atributos:

a) type: foi definido para o atributo o valor *Phaser.AUTO* pelo qual irá fazer com que o *browser* interprete a tecnologia WebGL canvas;
b) width: é a largura do elemento canvas;
c) height: é a altura do elemento canvas;
d) backgroundColor: usado para definir uma coloração de fundo;
e) scene: o objeto scene utiliza-se como parâmetros os métodos *preload*, *create* e *update.*

O método *preload* é responsável por carregar qualquer tipo de recurso do jogo como áudio, texturas e gráficos, esta função é executada antes de ser

carregada a aplicação por completo, já o método *create* tem o papel de configurar os elementos de interface do usuário e criar os objetos que vão aparecer pela primeira vez no jogo. Por fim, no método *update* possui as instruções necessárias para a movimentação do personagem e executar a função de detecção de colisão em um dado intervalo de tempo.

Para a criação dos objetos ou elementos do cenário usou-se o conceito de herança para inicializar suas devidas classes através da chamada da instância *Phaser.Class* como demonstra a figura 17.

Figura 17 – Instância dos elementos do cenário

```
var Personagem = new Phaser.Class({

    Extends: Phaser.GameObjects.Image,

    initialize:

        function Personagem(scene, x, y) {
            Phaser.GameObjects.Image.call(this, scene)

            this.setTexture('personagem');
            this.setPosition(x, y);
            this.setOrigin(0);

            scene.children.add(this);
        },

    moverEsquerda: function () {
        this.setPosition(this.x - 5, this.y);
    },
    moverDireita: function () {
        this.setPosition(this.x + 5, this.y);
    }

});
```

```
var Coluna = new Phaser.Class({

    Extends: Phaser.GameObjects.Image,

    initialize:

        function Coluna(scene, x, y) {
            Phaser.GameObjects.Image.call(this, scene)

            this.setTexture('coluna');
            this.setPosition(x, y);
            this.setOrigin(0);

            scene.children.add(this);
        }

});
```

Fonte: Do autor.

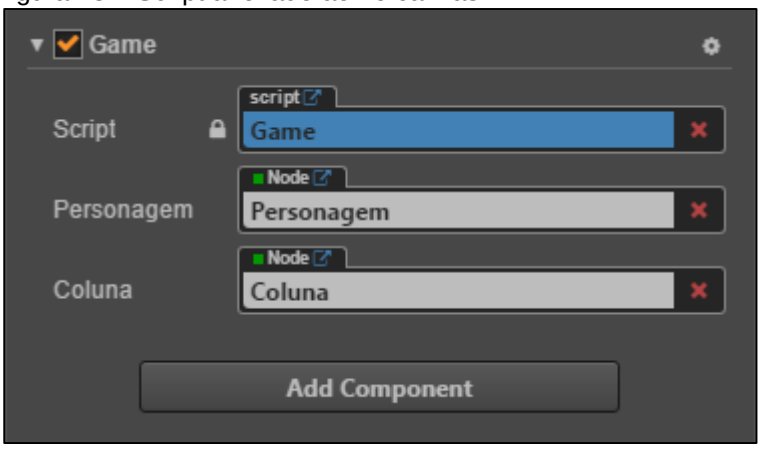
Em ambas as classes foram atribuídas texturas pré-definidas pela chamada do método *preload* e para o elemento personagem, os métodos *moverEsquerda* e *moverDireita* possui o comportamento de movimentar o eixo x nas duas direções. Além disto, foi definida uma herança da classe *Image* para mostrar imagens estáticas no cenário e o construtor rotulado de *initialize* que vai inicializar todas suas dependências como posições na tela e chamadas para renderização de texturas.

### 6.1.4 Desenvolvimento do protótipo no framework Cocos2d-x

Para a criação do protótipo no *framework* Cocos2d-x foi necessário preparar o ambiente de desenvolvimento utilizando o pacote de ferramentas chamado *Cocos Creator,* os componentes do cenário e os *scripts* foram inseridos usando a mecânica de clicar e arrastar disponibilizada pela ferramenta.

O próximo passo foi a criação do *script* usado para manipular os elementos do jogo, criou-se então, o *script* rotulado de *Game,* pelo qual foi anexado junto ao nó canvas como ilustrado na figura 18.

Figura 18 – Script anexado ao nó canvas




Fonte: Do autor.

Esta classe possui como referência os nós dos elementos presentes no cenário, chamados de Personagem e Coluna. Por meio desta classe, é possível definir uma herança da classe *cc.Component* como demonstra a figura 19.

Figura 19 – Herança e propriedades

```
cc.Class({
    extends: cc.Component,

    properties: {
        personagem: {
            default: null,
            type: cc.Node
        },

        coluna: {
            default: null,
            type: cc.Node
        }
    },
```

Fonte: Do autor.

Além disto, foi realizado o uso de propriedades que herdam da classe *cc.Node* representados como os nós da árvore do cenário. A implementação desta classe conta ainda, com o método *onLoad,* responsável por inicializar os eventos referentes ao teclado.

Para tratar os aspectos da movimentação do personagem, foram definidas duas variáveis globais intituladas de *DIREITA* e *ESQUERDA* pelo qual são condicionadas ao método *update* conforme figura 20. Este método por sua vez, têm o papel de ser executado a cada frame disparado no decorrer do processo de execução da aplicação.

Figura 20 – Método update

```
update: function (dt) {
    if (DIREITA) {
        this.moverDireita();
        this.verificaColisao();
    } else if (ESQUERDA) {
        this.moverEsquerda();
        this.verificaColisao();
    }
}
```

Fonte: Do autor.

Dentre as condições, são chamados os métodos *moverDireita* e *moverEsquerda* cuja função principal é realizar a movimentação do personagem, já

o método *verificaColisao* é responsável por executar o método da sobreposição de retângulos e em seguida realizar o cálculo do tempo em detecção de colisão que por sua vez, é chamado pela função *calcularTempo*.

### 6.1.5 Mecânicas e tecnologia utilizada

Para o desenvolvimento dos protótipos foram utilizados gráficos de duas dimensões (2D) e o uso de evento de teclado pelo qual possui movimentação para esquerda e direita.

A linguagem de programação final de utilização de ambos os *frameworks* foi o Javascript para uma comparação eficaz já que linguagens diferentes não é possível de se obter uma medida fidedigna o que se propõe analisar. Já no *framework* Libgdx, a linguagem de programação utilizada foi o java e convertido para javascript através da tecnologia GWT.

Como o Javascript é executado no navegador, a coleta dos dados de tempo de execução será realizada pelas variáveis de entrada *tempo_inicial*, *tempo_final* e as variáveis de saída *diff* e *tempo*.

### 6.1.6 Cálculo do tempo

Em ambos os protótipos criados, existe um método para realizar o cálculo do tempo, as variáveis de entrada *tempo_inicial* e *tempo_final* são responsáveis por obter a diferenciação do tempo decorrido conforme a figura 21.

Figura 21 – Cálculo do tempo

```
var tempo;
var tempo_inicial, tempo_final;

tempo_inicial = Date.now();

for (var i = 0; i < 50000; i++) {
    for (var j = 0; j < 50000; j++);
}

tempo_final = Date.now();
```

Fonte: Do autor.

Para gerar a diferença do tempo, as funções *Date.now* e *System.currentTimeMillis* são responsáveis para a obtenção do tempo em milissegundos. As duas funções citadas são baseadas na diferença entre o tempo atual e a data de 1 de janeiro, de 1970.

Após realizado a coleta dos dados do cálculo do tempo, é exibido no console ou componente de saída, as informações referentes ao tempo inicial, final, a diferença do tempo e o tempo em segundos.

### 6.1.7 Avaliação das funcionalidades

Esta etapa refere-se a análise dos *frameworks* através da avaliação das funcionalidades. Para realizar a avaliação, definiram-se critérios pelo qual faz o direcionamento em relação a uma decisão em como uma ferramenta será escolhida em um determinado contexto. Segundo Eves e Meehan (2008, tradução nossa), os critérios de avaliação são divididos nas seguintes categorias:

a) Custo;
b) Renderização;
c) Efeitos Sonoros;
d) Avatares;
e) Inteligência Artificial;
f) Físicas;
g) Interface de Usuário;
h) Rede;
i) Ferramentas de Edição;
j) *Level Design*;
k) Conteúdo;
l) Gravação;
m) Documentação;
n) Suporte.

Para elaboração dos critérios, foi utilizado a sigla AC que significa assessment criterion (Critério de avaliação) e XXX que é um índice único para cada referência, por fim, o grau de importância e o texto do critério.

Para a categoria custo, foram definidos os seguintes critérios:

Quadro 5 – Critérios de Custo

| | |
|---|---|
| AC-001 | Desejável |
| Possui extensões ou bibliotecas extras gratuitas | |
| AC-002 | Altamente desejável |
| Suporte a ferramenta de ambiente de desenvolvimento de jogos gratuitamente | |
| AC-003 | Desejável |
| Suporte a plug-ins de terceiros gratuitamente | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria renderização, foram definidos os seguintes critérios:

Quadro 6 – Critérios de Renderização

| | |
|---|---|
| AC-004 | Altamente desejável |
| Suporte a renderização pela API gráfica OpenGL | |
| AC-005 | Essencial |
| Suporte a renderização gráficas como sprites ou texturas | |
| AC-006 | Altamente desejável |
| Suporte a renderização de arquivos de fontes | |
| AC-007 | Desejável |
| Suporte a renderização de shaders | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria efeitos sonoros, foram definidos os seguintes critérios:

Quadro 7 – Critérios de Efeitos Sonoros

| | |
|---|---|
| AC-008 | Essencial |
| Possui suporte aos formatos de áudio MP3, OGG e WAV | |
| AC-009 | Desejável |
| Tem a capacidade de executar arquivos de áudio de até 1mb de tamanho | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria Inteligência Artificial, foram definidos os seguintes critérios:

Quadro 8 – Critérios de Inteligência Artificial

| | |
|---|---|
| AC-010 | Desejável |
| Suporte a API de Pathfinding | |
| AC-011 | Desejável |
| Suporte a máquina de estados para tomada de decisões | |
| AC-012 | Desejável |
| Suporte a movimentações inteligentes como comportamentos de direção ou a técnicas de movimento de formação | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria físicas, foram definidos os seguintes critérios:

Quadro 9 – Critérios de Físicas

| | |
|---|---|
| AC-013 | Desejável |
| Possui extensões de bibliotecas de físicas como box2d | |
| AC-014 | Desejável |
| Suporte a um motor de física integrado onde possui os elementos básicos como massa, posição, rotação, velocidade e amortecimento | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria interface de usuário, foram definidos os seguintes critérios:

Quadro 10 – Critérios de Interface de Usuário

| | |
|---|---|
| AC-015 | Desejável |
| Possui acesso a componentes de UI | |
| AC-016 | Altamente desejável |
| Possui acesso a um ambiente de interface gráfica para desenvolvimento | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria rede, foram definidos os seguintes critérios:

Quadro 11 – Critérios de Rede

| | |
|---|---|
| AC-017 | Desejável |
| Suporte a cliente TCP e a sockets com opções de configuração | |
| AC-018 | Altamente desejável |
| Possui uma interface de rede onde os objetos precisam se comunicar com a rede | |
| AC-019 | Desejável |
| Suporte a requisições HTTP | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria ferramentas de edição, foram definidos os seguintes critérios:

Quadro 12 – Critérios de Ferramentas de Edição

| | |
|---|---|
| AC-020 | Desejável |
| Suporte a mecânicas de clicar e arrastar componentes | |
| AC-021 | Desejável |
| Possui ferramenta ou extensão de edição de níveis | |
| AC-022 | Desejável |
| Possui ferramenta ou extensão de edição de componentes de UI | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria *Level Design*, foram definidos os seguintes critérios:

Quadro 13 – Critérios de *Level Design*

| | |
|---|---|
| AC-023 | Desejável |
| Possui editor de físicas | |
| AC-024 | Desejável |
| Possui editor de níveis | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria conteúdo, foram definidos os seguintes critérios:

Quadro 14 – Critérios de Conteúdo

| | |
|---|---|
| AC-025 | Altamente desejável |
| Suporte a sistema de eventos como clique do mouse, teclado e outros | |
| AC-026 | Altamente desejável |
| Suporte a sistema de ações como movimentação, rotação e interpolação | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria gravação, foram definidos os seguintes critérios:

Quadro 15 – Critérios de Gravação

| | |
|---|---|
| AC-027 | Desejável |
| Suporte a gravação de áudio | |
| AC-028 | Desejável |
| Suporte a leitura ou gravação de arquivos | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria documentação, foram definidos os seguintes critérios:

Quadro 16 – Critérios de Documentação

| | |
|---|---|
| AC-029 | Essencial |
| Possui documentação | |
| AC-030 | Desejável |
| Possui artigos relacionados a ferramenta | |

Fonte: Adaptado de Eves e Meehan (2008).

Para a categoria suporte, foram definidos os seguintes critérios:

Quadro 17 – Critérios de Suporte

| | |
|---|---|
| AC-031 | Desejável |
| Possui conteúdo ou tópicos de suporte para ajudar na ferramenta | |
| AC-032 | Desejável |
| Suporte a erros e correções através do envio de relatórios | |

Fonte: Adaptado de Eves e Meehan (2008).

Após a definição dos critérios, é necessário realizar a avaliação de cada funcionalidade através da decisão de múltiplas escolhas para então efetuar o cálculo da ponderação final para cada *framework*.

### 6.1.8 Avaliação de desempenho

Para a realização da etapa de testes de desempenho, foi utilizado o algoritmo da sobreposição de retângulos. O cálculo do tempo é gerado a partir da diferença do tempo inicial e o final, onde por sua vez, aplicou-se como critério de parada o número de iterações. Para calcular o número máximo de iterações foi utilizado a fórmula proposta por Villwock e Steiner (2007) no qual o número de iterações $N_{max}$ foi definido como $N_{max} = 500.n.m$, onde $m$ é o número de padrões e $n$ é o número de atributos.

Nesta análise, foi determinado um padrão, que é o cálculo do tempo e um atributo, que é o tempo em milissegundos, como resultado do cálculo, obteve-se um valor de 500 iterações que será usada na análise de detecção de colisão.

Cada iteração é composta pela diferença do tempo em milissegundos e em segundos. Ao final, é gerado uma média aritmética entre todas as iterações de tempo obtido.

Para a realização da etapa de testes, foi utilizado uma máquina com a seguinte configuração: Processador *Intel* Core i5-7400 CPU 3,00 G*hz*, 8Gb de memória *RAM*, 1 TB de armazenamento interno e sistema operacional *Windows 10 Pro*.

Foram realizadas vinte execuções em cada protótipo entre cada *framework*, utilizando dos mesmos elementos que compõem o planejamento do cenário e analisando as seguintes métricas:

a) número de iterações realizadas pelo algoritmo;
b) tempo gasto para executar o algoritmo de detecção de colisão em milissegundos;
c) tempo gasto para executar o algoritmo de detecção de colisão em segundos;

Após a finalização das execuções dos algoritmos e da obtenção dos dados será realizada a comparação entre a média de tempo calculada.

### 6.1.9 Medição da média de frames por segundo

Para a efetuar o algoritmo de medição, foi realizado o cálculo do número máximo de iterações pelo qual foi utilizado novamente a fórmula proposta por Villwock e Steiner (2007).

Para esta análise, foi determinado um padrão, que é o cálculo da média de frames por segundo, e um atributo, que é os frames por segundo, como resultado do cálculo, obteve-se um valor de 500 iterações no qual é executado no *loop* do protótipo ou método *update* conforme figura 22.

Figura 22 – Medição do tempo de *frame* por segundo

```
function medirFrames() {
    if(i != totalIteracoes) {
        console.log(game.loop.actualFps);
        somatorioFPS += game.loop.actualFps;
        i++;
    } else if(!mediaCalculada) {
        mediaFrames = somatorioFPS / 500;
        console.log("Media de frames: " + mediaFrames);
        mediaCalculada = true;
    }
}
```

Fonte: Do autor.

A função *medirFrames* é responsável por calcular a média total de frames entre as iterações, assim como realizar o somatório do total de *frames* e efetuar a média aritmética, condicionada pela variável booleana *mediaCalculada* e efetivada pela variável de saída *mediaFrames.*

## 6.2 RESULTADOS OBTIDOS

Os resultados referentes ao cálculo do tempo em detecção de colisão realizadas pelo algoritmo da sobreposição de retângulos foram obtidos por meio de uma média aritmética em cada protótipo.

Os dados coletados em tempo de execução são compostos pelo tempo em milissegundos e em segundos. Para efetivar o cálculo da média, é necessário

ser executado vinte iterações pelo qual será coletado o tempo em cada protótipo criado.

A coleta referente a avaliação das funcionalidades é dividida entre as categorias dos critérios de avaliação, onde foi realizada o recolhimento de um amontoado de escolhas ou caixas de seleção para os critérios já definidos.

Por fim, para a medida do tempo de *frame* por segundo, ou seja, o tempo médio que o algoritmo levou para executar entre as 500 iterações executadas.

### 6.2.1 Protótipo 1: Phaser

A tabela 1 mostra a coleta de dados das primeiras 20 iterações executadas no protótipo realizado no *framework* Phaser. Considerando a média aritmética dentre as 500 iterações executadas, foi obtido o tempo em milissegundos de aproximadamente 1551ms e o tempo em segundos de 1,551s.

Tabela 1 – Tempo em detecção de colisão no framework Phaser

| Tempo em milissegundos | Tempo em segundos |
|---|---|
| 1545ms | 1,545s |
| 1548ms | 1,548s |
| 1559ms | 1,559s |
| 1578ms | 1,578s |
| 1545ms | 1,545s |
| 1568ms | 1,568s |
| 1528ms | 1,528s |
| 1549ms | 1,549s |
| 1585ms | 1,585s |
| 1552ms | 1,552s |
| 1537ms | 1,537s |
| 1557ms | 1,557s |
| 1542ms | 1,542s |
| 1568ms | 1,568s |
| 1543ms | 1,543s |
| 1514ms | 1,514s |
| 1530ms | 1,530s |
| 1523ms | 1,523s |
| 1568ms | 1,568s |
| 1578ms | 1,578s |

Fonte: Do autor.

### 6.2.2 Protótipo 2: Libgdx

A tabela 2 mostra a coleta de dados das primeiras 20 iterações executadas no protótipo realizado no *framework* Libgdx. Considerando a média aritmética dentre as 500 iterações executadas, foi obtido o tempo em milissegundos de aproximadamente 1516ms e o tempo em segundos de 1,516s.

Tabela 2 – Tempo em detecção de colisão no framework Libgdx

| Tempo em milissegundos | Tempo em segundos |
|---|---|
| 1535ms | 1,535s |
| 1507ms | 1,507s |
| 1489ms | 1,489s |
| 1539ms | 1,539s |
| 1490ms | 1,490s |
| 1581ms | 1,581s |
| 1538ms | 1,538s |
| 1507ms | 1,507s |
| 1515ms | 1,515s |
| 1509ms | 1,509s |
| 1487ms | 1,487s |
| 1534ms | 1,534s |
| 1514ms | 1,514s |
| 1489ms | 1,489s |
| 1487ms | 1,487s |
| 1532ms | 1,532s |
| 1484ms | 1,484s |
| 1487ms | 1,487s |
| 1542ms | 1,542s |
| 1518ms | 1,518s |

Fonte: Do autor.

### 6.2.3 Protótipo 3: Cocos2d-x

A tabela 3 mostra a coleta de dados das primeiras 20 iterações executadas no protótipo realizado no *framework* Cocos2d-x. Considerando a média aritmética dentre as 500 iterações executadas, foi obtido o tempo em milissegundos de aproximadamente 1594ms e o tempo em segundos de 1,594s.

Tabela 3 – Tempo em detecção de colisão no framework Cocos2d-x

| Tempo em milissegundos | Tempo em segundos |
|---|---|
| 1614ms | 1,614s |
| 1618ms | 1,618s |
| 1622ms | 1,622s |
| 1561ms | 1,561s |
| 1644ms | 1,644s |
| 1578ms | 1,578s |
| 1567ms | 1,567s |
| 1553ms | 1,553s |
| 1577ms | 1,577s |
| 1626ms | 1,626s |
| 1599ms | 1,599s |
| 1582ms | 1,582s |
| 1646ms | 1,646s |
| 1620ms | 1,620s |
| 1584ms | 1,584s |
| 1602ms | 1,602s |
| 1555ms | 1,555s |
| 1613ms | 1,613s |
| 1557ms | 1,557s |
| 1618ms | 1,618s |

Fonte: Do autor.

### 6.2.4 Comparação dos frameworks

Para a comparação dos *frameworks* foi utilizado os critérios de avaliação já definidos anteriormente. A tabela 4 demonstra todas as particularidades de um *framework* específico baseando-se em um dado critério.

Tabela 4 – Comparação com base nos critérios

| Critério | Phaser | Libgdx | Cocos2d-x |
|---|---|---|---|
| AC-001 | | X | X |
| AC-002 | | | X |
| AC-003 | | X | X |
| AC-004 | X | X | X |
| AC-005 | X | X | X |
| AC-006 | X | X | X |
| AC-007 | X | X | X |
| AC-008 | X | X | X |
| AC-009 | X | X | X |
| AC-010 | | X | |
| AC-011 | | X | |
| AC-012 | | X | |
| AC-013 | X | X | X |
| AC-014 | X | X | X |
| AC-015 | X | X | X |
| AC-016 | | | X |
| AC-017 | | X | |
| AC-018 | | X | X |
| AC-019 | X | X | X |
| AC-020 | | | X |
| AC-021 | X | X | |
| AC-022 | | X | X |
| AC-023 | X | X | X |
| AC-024 | X | X | X |
| AC-025 | X | X | X |
| AC-026 | X | X | X |
| AC-027 | | X | |
| AC-028 | | X | X |
| AC-029 | X | X | X |
| AC-030 | X | X | X |
| AC-031 | X | X | X |
| AC-032 | X | X | X |

Fonte: Do autor.

Com base nos critérios, efetuou-se a pontuação nos três *frameworks*.

## 6.3 DISCUSSÃO DOS RESULTADOS

Ao analisar os resultados obtidos, foi possível perceber uma pequena variação em relação ao tempo coletado em detecção de colisão. Considerando os três protótipos desenvolvidos, eles podem ser considerados cenários de baixo nível de complexidade já que por sua vez, não possui texturização em sua parte gráfica e utiliza de mecânicas e elementos ou objetos minimalistas.

É importante ressaltar que, para as comparações das médias de *frames,* foi considerado um bom valor entre os *frameworks* Phaser e Cocos2d-x, já que se manteve perto da casa dos 60 *frames* por segundo. Na plataforma Libgdx houve uma queda em relação aos outros *frameworks*, mantendo-se a uma média de 52,8 *frames* por segundo como ilustrado na tabela 5.

Tabela 5 – Médias de frames por segundo

| Framework | Frames por segundo |
|---|---|
| Phaser | 59,52 |
| Libgdx | 52,8 |
| Cocos2d-x | 59,99 |

Fonte: Do autor.

Em relação as médias em tempo de execução em detecção de colisão utilizando o algoritmo de sobreposição de retângulos, observa-se, conforme tabela 6 que não se obteve uma variação significante diante das iterações executadas.

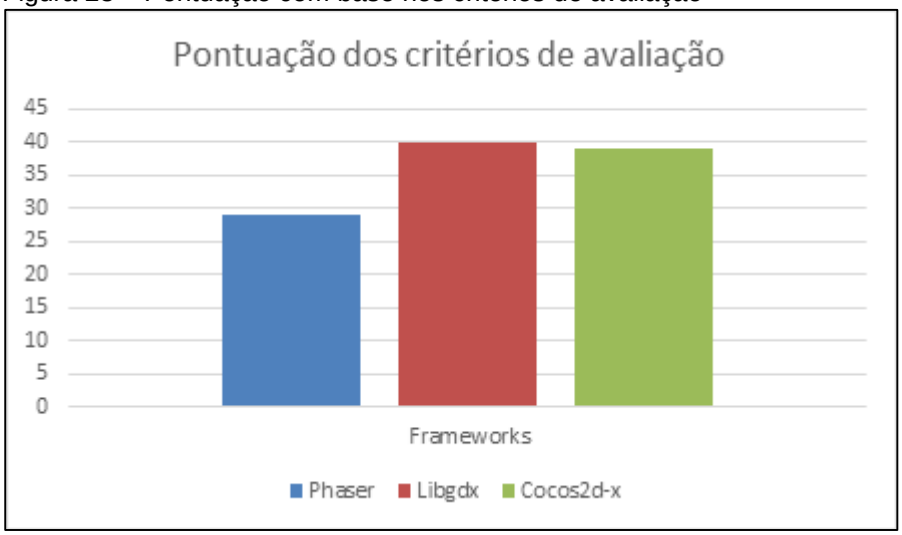
Tabela 6 – Médias em tempo de execução em detecção de colisão

| Framework | Tempo em milissegundos |
|---|---|
| Phaser | 1551ms |
| Libgdx | 1516ms |
| Cocos2d-x | 1594ms |

Fonte: Do autor.

Analisando primeiramente os resultados referentes a ponderação dos critérios de avaliação, o gráfico demonstrado na figura 23 dispõe estes dados em relação a comparação entre os três *frameworks.*

Figura 23 – Pontuação com base nos critérios de avaliação




Fonte: Do autor.

Tabela 7 – Pontuação com base nos critérios de avaliação

| Framework | Pontuação |
|---|---|
| Phaser | 29 Pontos |
| Libgdx | 40 Pontos |
| Cocos2d-x | 39 Pontos |

Fonte: Do autor.

Observando os dados da figura 23 e da tabela 7, nota-se uma diferença considerável entre a pontuação da plataforma Phaser nos demais *frameworks.* Foi obtido uma disparidade mínima em relação as ferramentas Libgdx e Cocos2d-x já que ambas resultaram em uma diferença de 1 ponto em sua média final.

# 7 CONCLUSÃO

Por meio da realização desta pesquisa, foi possível perceber a relevância dos algoritmos de detecção de colisão e o quão impactante deve ser levado em consideração a escolha da ferramenta adequada no processo de desenvolvimento de um jogo digital, influenciando diretamente seu desempenho, e consequentemente, a experiência do usuário.

Considerando as abordagens realizadas ao longo deste trabalho, o empenho se manteve com foco principal na avaliação da aplicação ou protótipo e também nos *frameworks*, determinando a pesquisa acerca destas ferramentas disponibilizadas gratuitamente a fim de se compreender os resultados obtidos.

Para o desenvolvimento das comparações apresentadas neste trabalho foram utilizados os conhecimentos de *frameworks,* juntamente dos conhecimentos de desenvolvimento de jogos digitais por meio da implementação do algoritmo de sobreposição de retângulos e da área da computação gráfica.

Os resultados obtidos foram essenciais para a compreensão da influência que uma dada ferramenta é exercida em um dado cenário. Concluiu-se que segundo a coleta de dados referentes ao tempo de execução em detecção de colisão sofreu uma variação mínima nos valores do tempo em milissegundos. Na importância destes testes, é visto que o uso de protótipos de baixa complexidade afeta significativamente, a fim de se obter resultados que sirvam de parâmetro a aplicação em cenários maiores, logo sua variação será maior.

A implementação do algoritmo da sobreposição de retângulos proporcionou que sua aplicação nos ambientes criados obtivesse resultados de seu desempenho. No entanto, a discussão se deu acerca da influência do algoritmo nestes resultados baseando-se nas métricas de avaliação de desempenho.

Após a finalização das comparações da avaliação de funcionalidades e da medição da média de *frames* por segundo, foi possível perceber que as diferenças geradas nos dados através dos resultados obtidos não impactaram na escolha ideal de uma ferramenta, mas sim, suas vantagens e desvantagens em relação aos critérios abordados.

Como uma proposta de trabalho futuro, para realizar a etapa da análise da comparação de desempenho, poderia ser utilizado outro algoritmo de detecção

de colisão como o da distância euclidiana para medir o tempo de execução do mesmo. Por fim, para realizar uma outra análise comparativa, uma proposta seria o uso de outros *frameworks* de desenvolvimento e também além dos critérios já utilizados, a definição de novos para efetuar a avaliação das funcionalidades.

# REFERÊNCIAS

ABDUL-KARIM, Ali. **Game Development Project with Cocos2D-X.** 2014. Disponível em: <https://www.theseus.fi/bitstream/handle/10024/74656/ali_thesis_final2.pdf?sequence=1>. Acesso em: 14 nov. 2017.

AFONSO, João António Areias Ribeiro Letra. **Framework para geração de Jogos Sérios para avaliação de competências.** 2017. Disponível em: <https://repositorio-aberto.up.pt/handle/10216/102866>. Acesso em: 13 nov. 2017.

AGUIAR, Mauricio. **Pontos de Função ou Pontos por Caso de Uso Como Estimar Projetos Orientados a Objetos.** 2003. Disponível em: <http://www.bfpug.com.br/Artigos/UCP/Aguiar-Pontos_de_Funcao_ou_Pontos_por_Caso_de_Uso.pdf>. Acesso em: 14 out. 2017.

ALBRECHT, Allan J.; GAFFNEY, John E.. **Software Function, Source Lines of Code, and Development Effort Prediction:** A Software Science Validation. 1983. Disponível em: <http://ieeexplore.ieee.org/document/1703110/metrics>. Acesso em: 14 out. 2017.

ANDRADE, Marlon M.; KNOP, Igor O.. **Projeto e desenvolvimento de jogos eletrônicos multiplataforma:** um estudo de caso utilizando Cocos2d-x. 2015. Disponível em: <https://seer.cesjf.br/index.php/cesi/article/view/300/396>. Acesso em: 15 nov. 2017.

BARBOSA, Rennan Araújo. **ANÁLISE DE GAME ENGINES PARA PLATAFORMAS MÓVEIS.** 2013. 52 f. TCC (Graduação) - Curso de Sistemas de Informação, Universidade Federal da Paraíba, Rio Tinto, 2013.

BIBAT, Bryan. **HTML 5 Shoot 'em Up in an Afternoon.** [S. L.]: Independently Published, 2015. 138 p.

BOMFIM, Márcia Regina Guiotti; ANDRADE, José Romildo. **Guia de Contagem de Pontos de Função.** 2015. Disponível em: <http://www.planejamento.gov.br/servicos/central-de-conteudos/GuiadeContagemdePontosdeFuncao.pdf>. Acesso em: 16 out. 2017.

BRASILIENSE, Fabricio**. Desenvolvimento de um Framework de Jogos 3D para Celulares.** 2006. Disponível em: <https://projetos.inf.ufsc.br/arquivos_projetos/projeto_427/TCC.pdf>. Acesso em: 12 set. 2017.

CAMPO, Marcelo Ricardo. **Compreensão visual de frameworks através da introspeção de exemplos.** 1997. Disponível em: <http://www.lume.ufrgs.br/handle/10183/17972>. Acesso em: 15 nov. 2017.

CAMPOS, Raphael Barreto Palhares de. **Análise comparativa de frameworks de persistência.** 2010. 63 f. TCC (Graduação) - Curso de Ciência da Computação,

Universidade Federal de Lavras, Lavras, 2010.

CAVALCANTE, Carlos Henrique Leitão; PEREIRA, Maria Luciana Almeida. **Comparativo entre Game Engines como Etapa Inicial para o Desenvolvimento de um Jogo de Educação Financeira.** 2018. Disponível em: <http://ceur-ws.org/Vol-2185/CtrlE_2018_paper_110.pdf>. Acesso em: 03 dez. 2018.

CHANDLER, Heather M.. **Manual de Produção de Jogos Digitais.** 2. ed. Porto Alegre: Bookman, 2012. 508 p.

CORDEIRO, Marco Aurélio. **Métricas de Software.** 2001. Disponível em: <http://underpop.online.fr/a/analise-de-sistemas/tecnicas-para-mensuracao-de-software/metricas-de-software.pdf>. Acesso em: 14 out. 2017.

COSTA, Valter José Correia. **LiNGS:** Framework de desenvolvimento de jogos multijogador em rede para dispositivos móveis. 2014. 169 f. Dissertação (Mestrado) - Curso de Engenharia Informática - Computação Móvel, Instituto Politécnico de Leiria, Leiria, 2014. Disponível em: <https://iconline.ipleiria.pt/bitstream/10400.8/1369/1/Valter_Jose_Correia_Costa_Projeto.pdf>. Acesso em: 15 nov. 2017.

CREDIDIO, Diego de Camargo. **Metodologia de Design aplicada à concepção de jogos digitais.** 2007. Disponível em: <http://repositorio.ufpe.br/bitstream/handle/123456789/3415/arquivo4382_1.pdf?sequence=1&isAllowed=y>. Acesso em: 16 nov. 2017.

CRUZ, Simone Jussara Araújo Santa. **Ferramenta Computacional Para Avaliação Heurística De Jogos Digitais.** 2011. Disponível em: <http://repositorio.ufpe.br/bitstream/handle/123456789/2438/arquivo3312_1.pdf>. Acesso em: 26 mar. 2017.

DIAS, Raquel. Análise por Pontos de Função: Uma Técnica para Dimensionamento de Sistemas de Informação. **Revista Eletrônica de Sistemas de Informação**, [s.l.], v. 2, n. 2, p.1-15, 31 dez. 2003. IBEPES (Instituto Brasileiro de Estudos e Pesquisas Sociais). http://dx.doi.org/10.21529/resi.2003.0202002.

EVES, Gary; MEEHAN, Pete. **The Development Of Selection Criteria For Game Engines In The Development Of Simulation Training Systems.** 2008. Disponível em: <http://www.simulationaustralasia.com/files/upload/pdf/research/110_Paper_O.pdf>. Acesso em: 15 nov. 2017.

FAAS, Travis. **An Introduction to HTML5 Game Development with Phaser.js.** New York: A K Peters/crc Press, 2016. 300 p.

FIGUEIREDO, Roberto Tenorio. **Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Eletrônicos.** 2014. Disponível em: <http://repositorio.ufpe.br/bitstream/handle/123456789/11981/DISSERTAÇÃO Roberto Tenorio Figueiredo.pdf>. Acesso em: 26 mar. 2017.

FULLERTON, Tracy. **Game Design Workshop:** A Playcentric Approach to Creating Innovative Games. 2. ed. Burlington: Morgan Kaufmann, 2008. 496 p.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph. **Design Patterns:** Elements of Reusable Object-Oriented Software. New York: Addison-wesley, 1994.

GOSE, Stephen. **Game Design Workbook:** Game development guide using Phaser JavaScript Game Framework. Tempe: Independently Published, 2016. 226 p.

GREGORY, Jason. **Game Engine Architecture.** Boca Raton: A K Peters, 2009. 864 p.

HERNANDEZ, Raydelto. **Building Android Games with Cocos2d-x.** Birmingham: Packt Publishing, 2015. 162 p.

HUSSAIN, Frahaan; GURUNG, Arutosh; JONES, Gareth. **Cocos2d-x Game Development Essentials:** Create iOS and Android games from scratch using Cocos2d-x. Birmingham: Packt Publishing, 2014. 136 p.

IWASAKI, Eliane Yumi. **MOVIMENTO OPEN SOURCE:** A importância da comunicação e da relação entre empresas e comunidades para o mercado. 2008. Disponível em: <http://www.dicas-l.com.br/download/movimento_open_source.pdf>. Acesso em: 26 out. 2017.

KERA, Marcello. **Detecção de colisão utilizando hierarquias em ferramentas de realidade virtual para treinamento médico.** 2005. 92 f. TCC (Graduação) - Curso de Ciência da Computação, Centro Universitário Eurípides de Marília, Marília, 2005. Disponível em: <http://aberto.univem.edu.br/handle/11077/444>. Acesso em: 09 nov. 2018.

LEITÃO, Michele de Vasconcelos. **Aplicação de Scrum em Ambiente de Desenvolvimento de Software Educativo.** 2010. 72 f. TCC (Graduação) - Curso de Engenharia da Computação, Universidade de Pernambuco, Recife, 2010. Disponível em: <http://tcc.ecomp.poli.br/20101/TCC_final_Michele.pdf>. Acesso em: 04 set. 2017.

MADEIRA, Charles Andryê Galvão. **FORGE V8:** Um framework para o desenvolvimento de jogos de computador e aplicações multimídia. 2001. Disponível em: <http://charles.madeira.free.fr/publications/MadeiraMasterThesis2001.pdf>. Acesso em: 26 mar. 2017.

MARQUEZ, David Saltares; SANCHEZ, Alberto Cejas. **Libgdx Cross-platform Game Development Cookbook.** Birmingham: Packt Publishing, 2014. 516 p.

MEDEIROS, Tainá Jesus. **Um Framework para criação de jogos voltados para o ensino de lógica de programação.** 2014. Disponível em: <https://repositorio.ufrn.br/jspui/bitstream/123456789/19596/1/TainaJesusMedeiros_DISSERT.pdf>. Acesso em: 15 maio 2017.

NAIR, Suryakumar Balakrishnan; OEHLKE, Andreas. **Learning Libgdx Game**

**Development.** 2. ed. Birmingham: Packt Publishing, 2015. 500 p.

NASCIMENTO, Áquilla Odlanier Faria; SIMÕES, Thállys Lisboa; LIMA, Joselice Ferreira. **Analise comparativa das funcionalidades de ferramentas para a criação de jogo educacional.** 2016. Disponível em: <http://200.131.5.234/ojs/index.php/anaisviiisimposio/article/download/73/58>. Acesso em: 04 dez. 2018.

NOVAK, Jeannie. **Game Development Essentials:** An Introduction. 3. ed. Nova Iorque: Cengage Learning, 2011. 510 p.

PEREIRA, André Luiz Kuczner et al. Frameworks para desenvolvimentos de jogos: uma abordagem vantajosa no desenvolvimento de jogos eletrônicos. **Revista Gestão em Foco**, [s. L.], v. 1, n. 9, p.1-12, maio 2017. Disponível em: <http://unifia.edu.br/revista_eletronica/revistas/gestao_foco/artigos/ano2017/058_frameworks.pdf>. Acesso em: 15 nov. 2017.

PRESSMAN, Roger S.. **Engenharia de Software:** Uma Abordagem Profissional. 7. ed. São Paulo: Bookman, 2011. 780 p.

PREZOTTO, Ezequiel Douglas; BONIATI, Bruno Batista. **Estudo de Frameworks Multiplataforma Para Desenvolvimento de Aplicações Mobile Híbridas.** 2014. Disponível em: <http://www.eati.info/eati/2014/assets/anais/artigo8.pdf>. Acesso em: 19 nov. 2017.

PYDD, Ezequiel Butzke. **Avaliação de algoritmos de detecção de colisão em jogos.** 2015. 48 f. TCC (Graduação) - Curso de Ciência da Computação, Universidade Federal do Pampa, Alegrete, 2015. Disponível em: <http://dspace.unipampa.edu.br:8080/bitstream/riu/1600/1/Avaliação de algoritmos de detecção de colisão em jogos.pdf>. Acesso em: 15 nov. 2017.

RIBEIRO, Italo Mendes da S.; SANTOS, Selan Rodrigues dos. **Métricas para Avaliação de Motores de Jogos Tridimensionais.** 2009. Disponível em: <https://www.sbgames.org/~sbgameso/papers/sbgames09/computing/short/cts22_09.pdf>. Acesso em: 15 nov. 2017.

ROCHA, Eduardo José Torres Sampaio. **Forge 16V:** Um Framework para o Desenvolvimento de Jogos Isométricos. 2003. Disponível em: <http://repositorio.ufpe.br/bitstream/handle/123456789/2531/arquivo4819_1.pdf>. Acesso em: 19 maio 2017.

SALEN, Katie; ZIMMERMAN, Eric. **Rules of Play:** Game Design Fundamentals. Cambridge: The Mit Press, 2004. 689 p.

SATO, Adriana Kei Ohashi. **Game Design e Prototipagem:** Conceitos e Aplicações ao Longo do Processo Projetual. 2010. Disponível em: <http://www.sbgames.org/papers/sbgames10/artanddesign/Full_A&D_10.pdf>. Acesso em: 15 nov. 2017.

SCHELL, Jesse. **The Art of Game Design:** A Book of Lenses. Burlington: Morgan

Kaufmann, 2008. 489 p.

SILVA, Pedro Henrique Pereira et al. **UMA ANÁLISE COMPARATIVA DE FRAMEWORKS PARA DESENVOLVIMENTO DE JOGOS NO ANDROID.** 2017. Disponível em: <http://www.contecsi.fea.usp.br/envio/index.php/contecsi/14CONTECSI/paper/viewPaper/4586>. Acesso em: 16 out. 2017.

SNYDER, Carolyn. **Paper Prototyping:** The Fast and Easy Way to Design and Refine User Interfaces. San Francisco: Morgan Kaufmann, 2003. 408 p.

TRINTA, Fernando Antonio Mota. **Definindo e Provendo Serviços de Suporte a Jogos Multiusuário e Multiplataforma:** Rumo à Pervasividade. 2007. Disponível em: <http://repositorio.ufpe.br/bitstream/handle/123456789/2060/arquivo6616_1.pdf>. Acesso em: 26 mar. 2017.

TRODO, Lia Degrazia. **Uso de Métricas nos Testes de Software.** 2009. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/18574/000730980.pdf>. Acesso em: 26 mar. 2017.

VILLWOCK, Rosangela; STEINER, Maria Teresinha Arns. **Análise do Desempenho do Algoritmo de Agrupamento Baseado em Colônia de Formigas Modificado.** 2007. Disponível em: <https://www.researchgate.net/publication/265882528_Analise_do_Desempenho_do_Algoritmo_de_Agrupamento_Baseado_em_Colonia_de_Formigas_Modificado>. Acesso em: 03 dez. 2018.

ZADRA, Augusto Nogueira; CARVALHO, Erivelton Oliveira; CARDOSO, Joécio Farley Santos. **Métricas de software:** Comparação entre Pontos de Função e Cocomo. 2015. Disponível em: <http://revistapensar.com.br/tecnologia/pasta_upload/artigos/a124.pdf>. Acesso em: 15 maio 2017.

# APÊNDICE(S)

## APÊNDICE A – ARTIGO CIENTÍFICO

# Análise comparativa de frameworks open source para o desenvolvimento de jogos multiplataforma

**Ederson Duarte Schaukoski[1], Luciano Antunes[1]**

[1]Universidade Do Extremo Sul Catarinense (UNESC) – Criciúma, SC - Brasil

eder-duarte@hotmail.com.br, luc@unesc.net

***Abstract.*** *Digital games are a means of entertainment that has grown a lot, are available in various devices such as smartphones. Facing this growth also arise the creation of development frameworks, many available free and being open source, and can be modified by any user or community. Through the use of a framework, it is possible to generate applications for several platforms using the same programming language. In the creation of digital games, collision detection algorithms are widely used for manipulating the elements of the scenario and verifying the overlap between two or more objects in a given time interval. The use of a prototype gives the game designer an improved view of the conceptual development of a game, that is, it is not enough to have the idea, it is necessary to test its viability throughout the production process. In various gaming genres, the use of character and graphic behavior also impacts game performance, and consequently, player experience. In this research, using the Phaser, Libgdx and Cocos2d-x frameworks, the algorithm of the overlapping of rectangles was implemented, which in turn was used to verify collision detection and perform the measurement of execution time. In order to compare the frameworks, functional evaluation metrics were used by which criteria were defined based on their weighting and their degree of criticality. Among the prototypes tested, the obtained results demonstrated that, the computational impact has its differences between each application generated and that also depend on the complexity of the same.*

***Resumo.*** *Jogos Digitais são um meio de entretenimento que vem crescendo bastante, estão disponíveis nos mais diversos dispositivos como aparelhos celulares inteligentes (smartphones). Diante deste crescimento surgem também a criação de frameworks de desenvolvimento, muitos disponibilizados gratuitamente e sendo de código aberto, podendo ser modificado por qualquer usuário ou comunidade. Por meio do uso de um framework, é possível gerar aplicações para diversas plataformas utilizando uma mesma linguagem de programação. Na criação de jogos digitais, os algoritmos de detecção de colisão são muito utilizados para a manipulação dos elementos do cenário e verificar a sobreposição entre dois ou mais objetos em um dado intervalo de tempo. A utilização de um protótipo fornece ao game designer uma visão aprimorada do desenvolvimento conceitual de um jogo, ou seja, não basta possuir a ideia, é necessário testar sua viabilidade ao longo do processo produtivo. Em diversos gêneros de jogos, o uso de comportamento nos personagens e elementos gráficos também impactam no desempenho do jogo, e consequentemente, na experiência do jogador. Nesta pesquisa, utilizando-se dos frameworks Phaser, Libgdx e Cocos2d-x, foi implementado o algoritmo da sobreposição de retângulos que por sua vez foi utilizado para a verificação de detecção de colisão e realizar a medição do tempo de execução. Para a comparação dos frameworks, foi utilizado métricas de avaliação de funcionalidades pelo qual foi*

*definido critérios baseando-se em sua ponderação e o seu grau de criticidade. Dentre os protótipos testados, os resultados obtidos demonstraram que, o impacto computacional possui suas diferenças entre cada aplicação gerada e que dependem também da complexidade do mesmo.*

## 1. Introdução

Em uma média de faturamento de dezenas de bilhões de dólares e em constante evolução, o mercado mundial de jogos apresenta ser promissor, uma vez que empresas estão investindo cada vez mais no desenvolvimento de técnicas e programas computacionais principalmente os que envolvam interfaces, animações gráficas, vídeos, sons tridimensionais e ambientes multiusuários baseados na Internet (MADEIRA, 2001).

Framework é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica, podendo atingir uma função específica durante a programação de uma aplicação (MEDEIROS, 2014).

O interesse por esse estudo surgiu mediante a oportunidade de diferenciar três frameworks existentes no mercado. Ao final desta pesquisa, pretende auxiliar a comunidade de desenvolvedores de jogos, já que muitas vezes, a incerteza e a imprecisão na escolha de certas ferramentas, fará com que grandes projetos acabem não se predominando no mercado pela falta do planejamento inicial.

## 2. Jogos Digitais

Os jogos eletrônicos são um meio de entretenimento que vem crescendo bastante, com a chegada das novas tecnologias e aparelhos celulares inteligentes (smartphones), os jogos estão disponíveis nos mais diversos dispositivos. De acordo com uma pesquisa feita pela Entertainment Software Association nos Estados Unidos, 50% dos americanos jogam jogos eletrônicos. Destes apenas 35% são menores de 18 anos, a média de idade de 35 anos e em média os jogadores gastam cerca de 1h por dia. E outro dado interessante é que os jogadores gastam mais que o triplo do tempo jogando do que praticando esportes ou lendo. O mercado de jogos eletrônicos movimentou US$ 28 bilhões em todo o mundo em 2003, e que desde 2003 superou o mercado cinematográfico (BRASILIENSE, 2006).

### 2.1. Desenvolvimento em jogos digitais

Jogos digitais são aplicações muito complexas que englobam não só a área da computação, mas também diversas outras, a utilização de framework para o desenvolvimento de jogos é muito útil para criar projetos com maior facilidade e agilidade pois proporcionam uma gama imensa de recursos pré-programáveis proporcionado por objetos e modelos de classes já existentes. Infelizmente, ainda não há uma definição clara de um framework para jogos, nem tão pouco um estudo detalhado da aplicabilidade de padrões de projeto, mas alguns autores utilizam algumas definições utilizando o senso comum de perspectiva (MADEIRA, 2001).

#### 2.1.1. Métodos de desenvolvimento

Muitos desenvolvedores tendem a evitar de utilizar metodologias ou processos formais e partem direto para a produção sem antes mesmo definir um agendamento claro do que realmente foi pensado, isso só funcionava 25 anos atrás onde os jogos eram muito mais simples, onde tornava mais fácil saber o que cada integrante da equipe precisava desenvolver. Hoje em dia, como as equipes são maiores e os jogos mais complexos, os desenvolvedores estão percebendo que devem encontrar outras soluções para gerenciar o ciclo de desenvolvimento (CHANDLER, 2012).

Um dos métodos de desenvolvimento ágeis de jogos eletrônicos bastante utilizados é o chamado Scrum que é baseado nas melhores práticas da indústria e que está em crescimento constante não só na área de jogos, mas também na criação de softwares comerciais. Um dos aspectos do Scrum é sua característica empírica e inovadora que é utilizado um método de desenvolvimento incremental, onde os requisitos sofrem constantes alterações durante o ciclo de vida do produto (LEITÃO, 2010).

### 2.2. Prototipagem

A prototipagem é a criação de um modelo funcional de uma ideia pelos quais nos permite verificar a viabilidade do projeto e fazer melhorias constantes. Quando se está construindo um protótipo, não é necessário se preocupar com a perfeição ou se a tecnologia parece estar bem otimizada, tudo que se precisa se preocupar são as mecânicas fundamentais. Há muitos casos de designers de jogos que optaram por adiantar o processo de prototipagem e iniciaram diretamente na implementação do código influenciando no tempo investido (FULLERTON, 2008, tradução nossa).

## 3. Framework para jogos

No desenvolvimento de aplicações, um framework é um conjunto cooperativo de classes que compõem um projeto reutilizável para um domínio específico de softwares. Ele fornece orientação arquitetural através da divisão do projeto em classes abstratas e definindo suas responsabilidades e colaborações. Um desenvolvedor customiza o framework para uma aplicação específica utilizando instâncias das classes do mesmo (GAMMA et al., 1994, tradução nossa).

Campo (1997, p. 32) salienta de uma maneira informal que “um framework pode ser considerado como uma infraestrutura de classes que fornecem o comportamento necessário para implementar aplicações dentro de um dado domínio”.

Foram selecionados para esta pesquisa três frameworks diferentes para o desenvolvimento de jogos 2D, são eles: Libgdx, Phaser e Cocos2d-x.

### 3.1. Libgdx

Libgdx é um framework de código aberto (open source) multiplataforma utilizado principalmente para o desenvolvimento de jogos eletrônicos através da linguagem de programação Java. Seu principal destaque é conseguir rodar e debugar o código no desktop como uma aplicação nativa. Desde seu lançamento da versão 0.1 em março de 2010, muito trabalho foi feito para melhorar ainda mais esta biblioteca pois conta com uma comunidade ativa e em constante crescimento (NAIR; OEHLKE, 2015, tradução nossa).

Marquez e Sanchez (2014, tradução nossa) afirmam que a Libgdx possui uma API multiplataforma pelos quais os usuários podem escrever uma única vez sua aplicação através da linguagem Java e distribuir para todas as plataformas suportadas. Cada plataforma dispõe de um backend que implementa subsistemas de baixo nível, que são: Aplicação, Gráficos, Áudio, Entrada, Arquivos e Rede.

A figura 1 apresenta a arquitetura principal da API da Libgdx destacando suas plataformas e seus respectivos subsistemas.

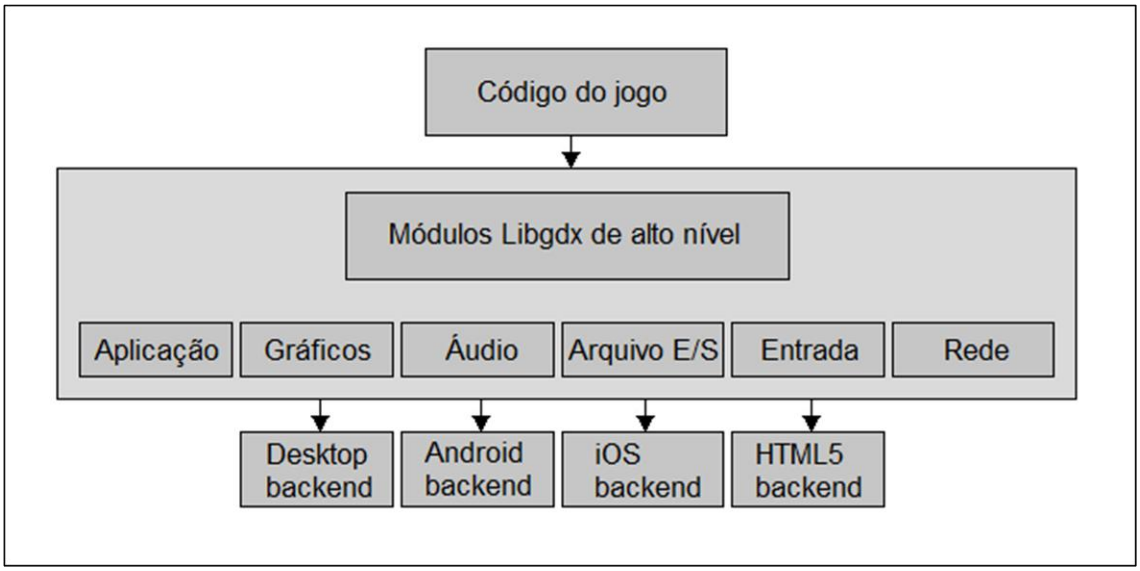



**Figura 1. Arquitetura da API multiplataforma Libgdx**

### 3.2. Phaser

Criado por Richard Davey, a framework Phaser é usada para criar jogos 2D utilizando Javascript como sua linguagem de programação padrão e teve forte inspiração do motor de jogos chamado Flixel que foi bastante popular no mundo do desenvolvimento de jogos em flash. Possui uma comunidade grande de desenvolvedores, com suporte a animações, sons, diferentes motores para física e partículas (FAAS, 2016, tradução nossa).

De acordo com Afonso (2017), Phaser é uma ferramenta com o foco em usuários com conhecimentos em programação já que não possui qualquer tipo de interface gráfica para a construção do projeto. São utilizados tecnologias da web como WebGL e o Canvas para geração do jogo e está frequentemente a receber atualizações pelo fato de ser uma ferramenta open source.

Outra característica notada é a opção de alternar entre aceleração de hardware e modo WebGL para renderização do cenário. A renderização dos objetos é realizada através de outro framework chamado Pixi.js, que vai tratar toda a parte das animações e renderização das imagens, simplificando as chamadas de gráficos do Javascript e fornecendo uma única API que vai funcionar tanto para WebGL quanto para o canvas (FAAS, 2016, tradução nossa).

### 3.3. Cocos2d-x

Cocos2d-x é uma ramificação do popular framework para jogos de sistema operacional IOS chamado Cocos2d. Seu primeiro lançamento foi em novembro de 2010 e mais tarde comprada pela Chukong Technologies em 2011 e é mantida até hoje por uma comunidade de mais de 400.000 desenvolvedores em todo o mundo (HERNANDEZ, 2015, tradução nossa).

Esta é uma ferramenta open source bastante popular que utiliza para o seu desenvolvimento três linguagens principais que são C++, Lua ou Javascript. Seu principal destaque é a possibilidade de exportar os projetos para diversas plataformas incluindo desktop e mobile utilizando apenas uma linguagem de programação (HUSSAIN; GURUNG; JONES, 2014, tradução nossa).

Hernandez (2015, tradução nossa) menciona que a Cocos2d-x é capaz de encapsular todas as particularidades do jogo como som, música, física, entradas do usuário, sprites, cenas

e transições. Portanto basta o desenvolvedor se concentrar na lógica do jogo invés de implementar todas as mecânicas partindo do zero.

## 4. Análise comparativa de frameworks open source para o desenvolvimento de jogos multiplataforma

Neste trabalho, foram desenvolvidos três protótipos equivalentes em cada framework utilizando o mesmo número de elementos e características semelhantes, tendo como objetivo principal comparar diferentes resultados.

Como resultados obtidos, foram avaliadas as funcionalidades de cada framework com base na métrica de avaliação de funcionalidades onde é feito a definição do critério para avaliação, sua importância e a ponderação da avaliação.

### 4.1. Planejamento do cenário

Para o desenvolvimento do trabalho, a primeira etapa consistiu no planejamento do cenário. Os elementos que compõem o cenário do jogo foram baseados em objetos minimalistas onde terão as características principais para realizar a análise e os testes de desempenho conforme figura 2.

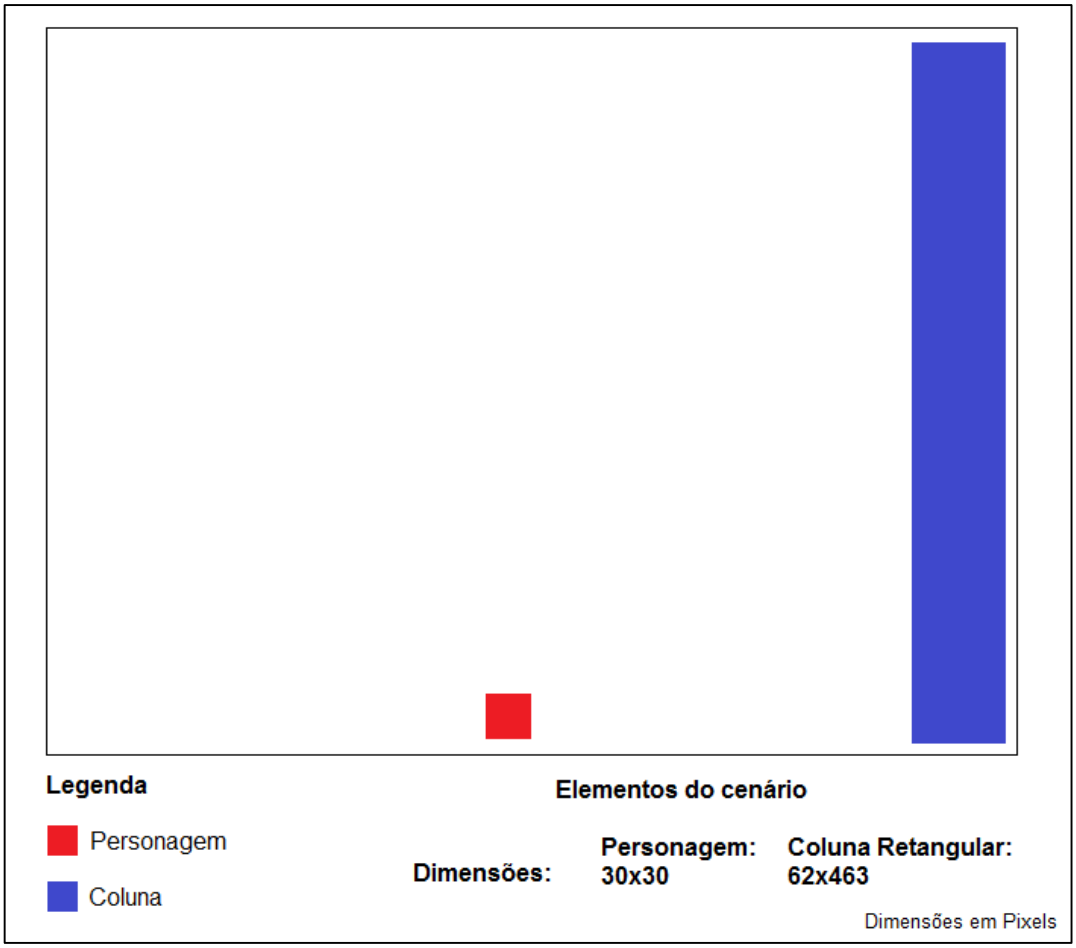



**Figura 2. Cenário do jogo**

A resolução adotada para a construção do cenário foi de 640x480px (640 pixels de largura e 480 pixels de altura), a fim de fazer com que o mesmo se adeque a qualquer plataforma.

### 4.1.1. Elementos presentes no cenário

Para a criação dos elementos do cenário, adotou-se os seguintes objetos:

a) um personagem de 30 pixels de largura e 30 pixels de altura pelos quais tem a possibilidade de movimentação para esquerda e direita;
b) uma coluna de 62 pixels de largura e 463 pixels de altura;

Cada framework possui seu próprio eixo principal variando de x e y como demonstra a figura 3, sendo assim, o framework Phaser foi o único que se mantém com seu eixo principal diferente dos demais pois o mesmo foi baseado no eixo dos joysticks, variando de -1 (esquerda ou cima) e 1 (direita ou baixo), desta forma, foi designado coordenadas diferentes seguindo conforme o planejamento do cenário.

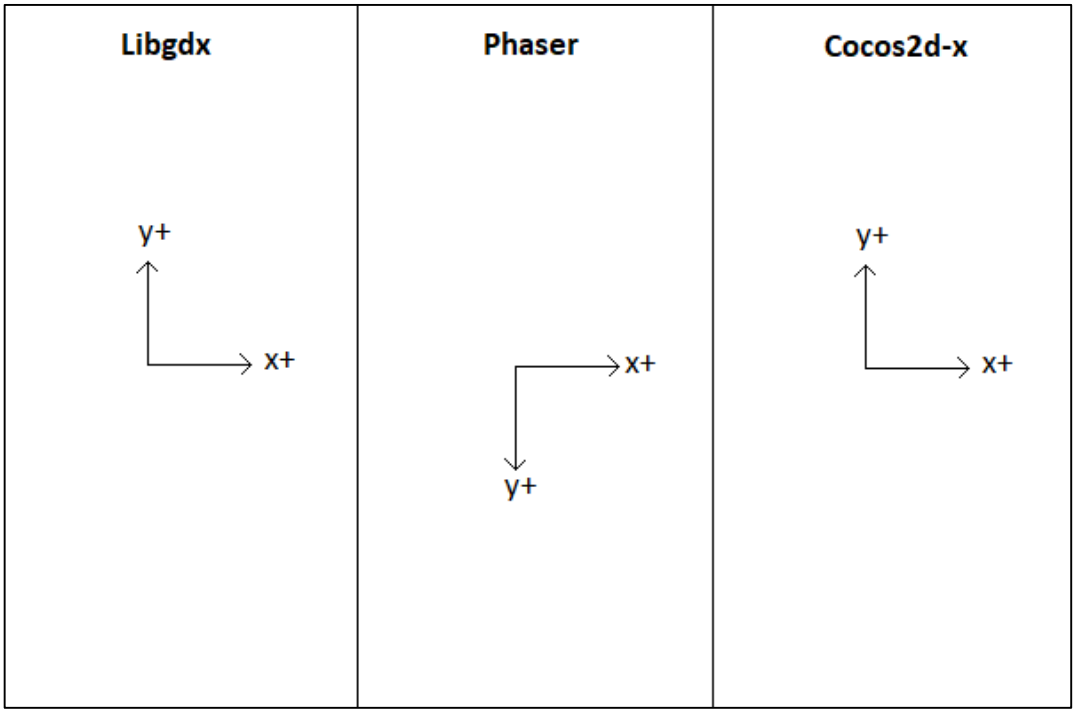



**Figura 3. Comparação do eixo principal**

## 4.2. Mecânicas e tecnologia utilizada

Para o desenvolvimento dos protótipos foram utilizados gráficos de duas dimensões (2D) e o uso de evento de teclado pelo qual possui movimentação para esquerda e direita.

A linguagem de programação final de utilização de ambos os frameworks foi o Javascript para uma comparação eficaz já que linguagens diferentes não é possível de se obter uma medida fidedigna o que se propõe analisar. Já no framework Libgdx, a linguagem de programação utilizada foi o java e convertido para javascript através da tecnologia GWT.

Como o Javascript é executado no navegador, a coleta dos dados de tempo de execução será realizada pelas variáveis de entrada tempo_inicial, tempo_final e as variáveis de saída diff e tempo.

## 4.3. Cálculo do tempo

Em ambos os protótipos criados, existe um método para realizar o cálculo do tempo, as variáveis de entrada tempo_inicial e tempo_final são responsáveis por obter a diferenciação do tempo decorrido conforme a figura 4.

```
var tempo;
var tempo_inicial, tempo_final;

tempo_inicial = Date.now();

for (var i = 0; i < 50000; i++) {
    for (var j = 0; j < 50000; j++);
}

tempo_final = Date.now();
```

**Figura 4. Comparação do eixo principal**

Para gerar a diferença do tempo, as funções Date.now e System.currentTimeMillis são responsáveis para a obtenção do tempo em milissegundos. As duas funções citadas são baseadas na diferença entre o tempo atual e a data de 1 de janeiro, de 1970.

### 4.4. Avaliação de desempenho

Para a realização da etapa de testes de desempenho, foi utilizado o algoritmo da sobreposição de retângulos. O cálculo do tempo é gerado a partir da diferença do tempo inicial e o final, onde por sua vez, aplicou-se como critério de parada o número de iterações. Para calcular o número máximo de iterações foi utilizado a fórmula proposta por Villwock e Steiner (2007) no qual o número de iterações $N_{max}$ foi definido como $N_{max} = 500.n.m$, onde $m$ é o número de padrões e $n$ é o número de atributos.

### 4.5. Medição da média de frames por segundo

Para a efetuar o algoritmo de medição, foi realizado o cálculo do número máximo de iterações pelo qual foi utilizado novamente a fórmula proposta por Villwock e Steiner (2007).

Para esta análise, foi determinado um padrão, que é o cálculo da média de frames por segundo, e um atributo, que é os frames por segundo, como resultado do cálculo, obteve-se um valor de 500 iterações no qual é executado no loop do protótipo ou método update conforme figura 5.

```
function medirFrames() {
    if(i != totalIteracoes) {
        console.log(game.loop.actualFps);
        somatorioFPS += game.loop.actualFps;
        i++;
    } else if(!mediaCalculada) {
        mediaFrames = somatorioFPS / 500;
        console.log("Media de frames: " + mediaFrames);
        mediaCalculada = true;
    }
}
```

**Figura 5. Medição do tempo de frame por segundo**

## 4.6. Resultados Obtidos

Os resultados referentes ao cálculo do tempo em detecção de colisão realizadas pelo algoritmo da sobreposição de retângulos foram obtidos por meio de uma média aritmética em cada protótipo.

Os dados coletados em tempo de execução são compostos pelo tempo em milissegundos e em segundos. Para efetivar o cálculo da média, é necessário ser executado vinte iterações pelo qual será coletado o tempo em cada protótipo criado.

A coleta referente a avaliação das funcionalidades é dividida entre as categorias dos critérios de avaliação, onde foi realizada o recolhimento de um amontoado de escolhas ou caixas de seleção para os critérios já definidos.

Por fim, para a medida do tempo de frame por segundo, ou seja, o tempo médio que o algoritmo levou para executar entre as 500 iterações executadas.

## 4.7. Discussão dos Resultados

Ao analisar os resultados obtidos, foi possível perceber uma pequena variação em relação ao tempo coletado em detecção de colisão. Considerando os três protótipos desenvolvidos, eles podem ser considerados cenários de baixo nível de complexidade já que por sua vez, não possui texturização em sua parte gráfica e utiliza de mecânicas e elementos ou objetos minimalistas.

É importante ressaltar que, para as comparações das médias de frames, foi considerado um bom valor entre os frameworks Phaser e Cocos2d-x, já que se manteve perto da casa dos 60 frames por segundo. Na plataforma Libgdx houve uma queda em relação aos outros frameworks, mantendo-se a uma média de 52,8 frames por segundo como ilustrado na tabela 1.

**Tabela 1. Médias de frames por segundo**

| Framework | Frames por segundo |
|---|---|
| Phaser | 59,52 |
| Libgdx | 52,8 |
| Cocos2d-x | 59,99 |

Em relação as médias em tempo de execução em detecção de colisão utilizando o algoritmo de sobreposição de retângulos, observa-se, conforme tabela 2 que não se obteve uma variação significante diante das iterações executadas.

**Tabela 2. Médias em tempo de execução em detecção de colisão**

| Framework | Tempo em milissegundos |
|---|---|
| Phaser | 1551ms |
| Libgdx | 1516ms |
| Cocos2d-x | 1594ms |

Analisando primeiramente os resultados referentes a ponderação dos critérios de avaliação, o gráfico demonstrado na figura 6 dispõe estes dados em relação a comparação entre os três frameworks.

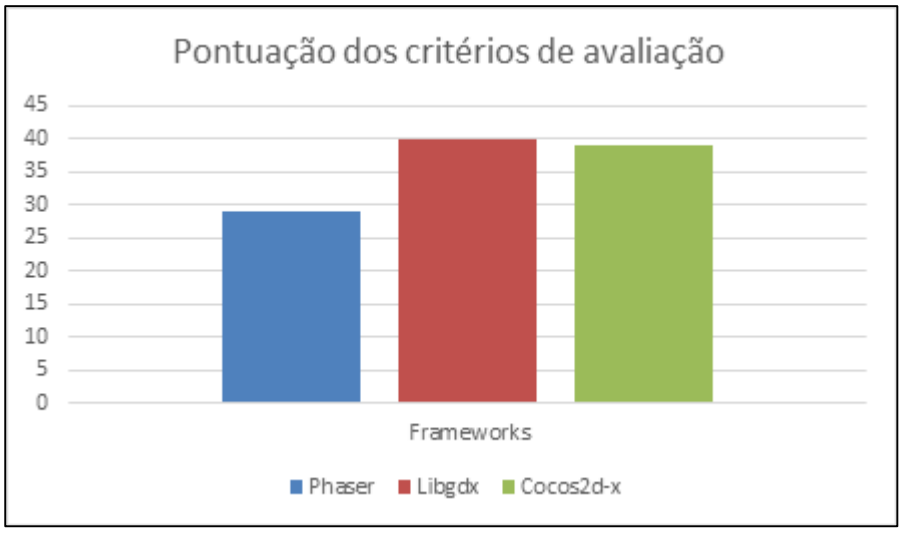



**Figura 6. Pontuação com base nos critérios de avaliação**

**Tabela 3. Pontuação com base nos critérios de avaliação**

| Framework | Pontuação |
|---|---|
| Phaser | 29 Pontos |
| Libgdx | 40 Pontos |
| Cocos2d-x | 39 Pontos |

Observando os dados da figura 6 e da tabela 3, nota-se uma diferença considerável entre a pontuação da plataforma Phaser nos demais frameworks. Foi obtido uma disparidade mínima em relação as ferramentas Libgdx e Cocos2d-x já que ambas resultaram em uma diferença de 1 ponto em sua média final.

## 5. Conclusão

Por meio da realização desta pesquisa, foi possível perceber a relevância dos algoritmos de detecção de colisão e o quão impactante deve ser levado em consideração a escolha da ferramenta adequada no processo de desenvolvimento de um jogo digital, influenciando diretamente seu desempenho, e consequentemente, a experiência do usuário.

Após a finalização das comparações da avaliação de funcionalidades e da medição da média de frames por segundo, foi possível perceber que as diferenças geradas nos dados através dos resultados obtidos não impactaram na escolha ideal de uma ferramenta, mas sim, suas vantagens e desvantagens em relação aos critérios abordados.

Como uma proposta de trabalho futuro, para realizar a etapa da análise da comparação de desempenho, poderia ser utilizado outro algoritmo de detecção de colisão como o da distância euclidiana para medir o tempo de execução do mesmo. Por fim, para realizar uma outra análise comparativa, uma proposta seria o uso de outros frameworks de desenvolvimento e também além dos critérios já utilizados, a definição de novos para efetuar a avaliação das funcionalidades.

## Referências

AFONSO, João Antônio Areias Ribeiro Letra. **Framework para geração de Jogos Sérios**

**para avaliação de competências.** 2017. Disponível em: <https://repositorio-aberto.up.pt/handle/10216/102866>. Acesso em: 13 nov. 2017.

BRASILIENSE, Fabricio. **Desenvolvimento de um Framework de Jogos 3D para Celulares**. 2006. Disponível em: <https://projetos.inf.ufsc.br/arquivos_projetos/projeto_427/TCC.pdf>. Acesso em: 12 set. 2017.

CAMPO, Marcelo Ricardo. **Compreensão visual de frameworks através da introspeção de exemplos.** 1997. Disponível em: <http://www.lume.ufrgs.br/handle/10183/17972>. Acesso em: 15 nov. 2017.

CHANDLER, Heather M.. **Manual de Produção de Jogos Digitais**. 2. ed. Porto Alegre: Bookman, 2012. 508 p.

FAAS, Travis. **An Introduction to HTML5 Game Development with Phaser.js.** New York: A K Peters/crc Press, 2016. 300 p.

FULLERTON, Tracy. **Game Design Workshop:** A Playcentric Approach to Creating Innovative Games. 2. ed. Burlington: Morgan Kaufmann, 2008. 496 p.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph. **Design Patterns**: Elements of Reusable Object-Oriented Software. New York: Addison-wesley, 1994.

HERNANDEZ, Raydelto. **Building Android Games with Cocos2d-x.** Birmingham: Packt Publishing, 2015. 162 p.

HUSSAIN, Frahaan; GURUNG, Arutosh; JONES, Gareth. **Cocos2d-x Game Development Essentials:** Create iOS and Android games from scratch using Cocos2d-x. Birmingham: Packt Publishing, 2014. 136 p.

LEITÃO, Michele de Vasconcelos. **Aplicação de Scrum em Ambiente de Desenvolvimento de Software Educativo**. 2010. 72 f. TCC (Graduação) - Curso de Engenharia da Computação, Universidade de Pernambuco, Recife, 2010. Disponível em: <http://tcc.ecomp.poli.br/20101/TCC_final_Michele.pdf>. Acesso em: 04 set. 2017.

MADEIRA, Charles Andryê Galvão. **FORGE V8:** Um framework para o desenvolvimento de jogos de computador e aplicações multimídia. 2001. Disponível em: <http://charles.madeira.free.fr/publications/MadeiraMasterThesis2001.pdf>. Acesso em: 26 mar. 2017.

MARQUEZ, David Saltares; SANCHEZ, Alberto Cejas. **Libgdx Cross-platform Game Development Cookbook.** Birmingham: Packt Publishing, 2014. 516 p.

MEDEIROS, Tainá Jesus. **Um Framework para criação de jogos voltados para o ensino de lógica de programação**. 2014. Disponível em: <https://repositorio.ufrn.br/jspui/bitstream/123456789/19596/1/TainaJesusMedeiros_DISSERT.pdf>. Acesso em: 15 maio 2017.

NAIR, Suryakumar Balakrishnan; OEHLKE, Andreas. **Learning Libgdx Game Development.** 2. ed. Birmingham: Packt Publishing, 2015. 500 p.

VILLWOCK, Rosangela; STEINER, Maria Teresinha Arns. **Análise do Desempenho do Algoritmo de Agrupamento Baseado em Colônia de Formigas Modificado.** 2007. Disponível em: <https://www.researchgate.net/publication/265882528_Analise_do_Desempenho_do_Algoritmo_de_Agrupamento_Baseado_em_Colonia_de_Formigas_Modificado>. Acesso em: 03 dez. 2018.