DESENVOLVIMENTO DE JOGOS MULTIPLATAFORMA EM HTML5 COM *PHASER* [1,2]

Tiago Zanchi de Paula[3]
Aline Bossi Pereira da Silva[4]

RESUMO

Este trabalho tem o objetivo de apresentar as etapas de desenvolvimento de um jogo multiplataforma através da *game Engine Phaser*, que utiliza como recurso as novas funcionalidades da linguagem de marcação HTML5 e a linguagem de programação JavaScript. Essa pesquisa apresenta o benefício da utilização da Phaser para desenvolvimento de jogos multiplataforma, demonstrando de forma prática os pontos positivos de sua utilização através de um projeto real de estudo de caso.
Palavras Chave: HTML5 ; Jogos *Web* ; Jogos Digitais ; *Phaser*

ABSTRACT

This paper aims to present the steps of the development of a multiplatform game with the Phaser Game Engine, that uses as resources the new functionalities of the markup language HTML5 and the programming language JavaScript. This research will show the benefits of the utilization of Phaser to develop multiplatform games, showing in a practical way the positive aspects of its use through a case study real project.
Keywords: *HTML5 ; Web games ; Digital games ; Phaser*

INTRODUÇÃO

O HTML5 proporcionou uma grande evolução para os *browsers*, evitando o uso de *plug-ins*[5] ou *APIs*[6] externas para apresentação de conteúdo multimídia na *Web.* A possibilidade de incluir áudios, vídeos e gráficos 2D e 3D no navegador do usuário sem nenhum auxílio externo interessou diversos desenvolvedores e popularizou a nova versão da linguagem de marcação *HyperText Markup Language* (HTML).

Com o avanço tecnológico dos *smarthphones* e *tablets* é cada vez mais necessário a inclusão de conteúdo atualizado na tela dos dispositivos portáteis, pois os dispositivos são atualizados com versões robustas de *browsers* que garantem pouquíssimas limitações em relação às suas versões de navegadores *desktop,* além disso estas atualizações possibilitam a criação de conteúdo multimídia e multiplataforma de forma mais fácil, atingindo um público maior.

Com a popularização dos jogos desenvolvidos para a *Web* em HTML5, um grande número de ferramentas surgiu para facilitar a criação desse tipo de conteúdo. Essas ferramentas são chamadas de *game engines* ou motores de jogos e abrangem diversas ferramentas preparadas para auxiliar a criação de conteúdos multimídia.

Uma destas ferramentas é a *Phaser Game Engine, que* é um motor de jogos para HTML5 e JavaScript que surgiu com o objetivo de facilitar ainda mais a criação de jogos multiplataforma proporcionada pelo HTML5. As funções presentes na *Phaser* são *design*adas para a execução em diversas plataformas, desta forma os desenvolvedores e criadores procuram um bom desempenho na maior gama de dispositivos possível. Os jogos desenvolvidos com *Phaser* também podem ser exportados com instaladores para serem executados como um aplicativo nativo, tanto em *desktops* quanto em dispositivos móveis.

O estudo deste tema justifica-se, pois com a ascensão de *smartphones* no mundo, o mercado de jogos multiplataforma cresceu de forma considerável. Segundo a IDC BRASIL (2014), as vendas de *smartphones* no Brasil bateram recorde no segundo trimestre de 2014, aumentando ainda mais a base instalada de aparelhos no país. Com o número de *smartphones* crescendo, o número de potenciais consumidores também cresce, o que aumenta o interesse do mercado nessa plataforma. Ao desenvolver um

---

[1] Artigo baseado em Trabalho e Conclusão de Curso (TCC) desenvolvido em cumprimento à exigência curricular do Curso de Tecnologia em Jogos Digitais depositado no 2º semestre de 2014
[2] Depósito na Biblioteca em 23/12/2014
[3] Discente de Tecnologia em Jogos Digitais – Fatec Americana – Centro Estadual de Educação Tecnológica Paula Souza ; Contato: tiagozdp@hotmail.com
[4] Prof. da.Fatec Americana - Mestrando em Tecnologia e Inovação pela Faculdade de Tecnologia – Unicamp ; Contato: alinebossi@hotmail.com
[5] Segundo MOZILLA (2014), um *plug-in* é um programa instalado no navegador que permite a utilização de recursos não presentes na linguagem HTML, na qual são criadas as páginas.
[6] API é uma Interface de Programação de Aplicativos, essa interface é um conjunto de ferramentas que será executada em conjunto com uma aplicação *web*.

jogo multiplataforma, é possível lançar um produto para uma plataforma muito promissora que está em crescimento além de lançar o mesmo produto para uma plataforma já consolidada, aumentando consideravelmente o número de pessoas atingidas e, consequentemente, aumentando a probabilidade de sucesso do produto.

Em relação à metodologia, o trabalho é elaborado através de uma pesquisa bibliográfica e um estudo de caso. A pesquisa bibliográfica, conforme destacado por FONSECA (2002), permite ao pesquisador conhecer o que já se estudou sobre o assunto e tem como principais fontes livros, artigos e *Internet,* enquanto o estudo *de caso* será realizado utilizando todos os conhecimentos obtidos através das pesquisas realizadas.

O objetivo central desse trabalho é demonstrar de forma prática o desenvolvimento de um jogo multiplataforma na E*ngine Phaser*. As etapas que serão estudadas envolvem desde a prototipação até o produto final, sendo executado e testado em diversos dispositivos. Para isso, esse trabalho está dividido em cinco capítulos: o primeiro conceitua a linguagem de programação JavaScript, as novas funcionalidades do HTML5, a E*ngine Phaser* e o conceito que será adotado como multiplataforma. O segundo explora a prototipação, os testes do protótipo e os desafios encontrados desta etapa, o terceiro capítulo apresenta o desenvolvimento do jogo em fase beta e os pontos positivos e negativos do desenvolvimento através da *Phaser*, seguido pelo quarto capítulo que apresenta o desempenho do jogo em diferentes plataformas, prosseguindo com o quinto capitulo que apresenta a conclusão do trabalho.

1 DESENVOLVENDO UM JOGO MULTIPLATAFORMA COM *PHASER*

Um jogo multiplataforma, desenvolvido em *Phaser,* utiliza mecânicas como o *canvas* que cria uma área para se desenhar na página e o HTML5 *Audio* que permite a execução de áudio sem o uso de plug-ins, introduzidas no HTML5 e que são controladas através de JavaScript, isso permite que o jogo seja executado em diversos dispositivos que reconhecem essas tecnologias, sem auxilio externo de nenhum complemento. Para esse trabalho, antes de definir os conceitos dessas tecnologias, precisamos entender o que elas são e como surgiram.

1.1 HTML5

A última versão do HTML (e seus associados CSS e JavaScript) é o HTML5 e está gerando um entusiasmo considerável por causa de funcionalidades como o *canvas* para apresentar imagens e animações; suporte para vídeo e áudio; e novas *tags* para definir elementos em comum nos documentos como o cabeçalho, seção e rodapé. Com esta linguagem pode-se criar um *Website* sofisticado e altamente interativo. (MEYER, 2010, tradução nossa).

Segundo o W3C (2014), o HTML5 foi fruto de uma cooperação entre a *World Wide Web Consortium (*W3C) e a *Web Hypertext Application Technology Working Group (*WHATWG) que ocorreu em 2006. O objetivo era criar um padrão único para as páginas da *Web* que fossem criadas a partir daquele momento, para isso foram determinadas algumas regras, como: novas funcionalidades deveriam ser baseadas em HTML, CSS, DOM[7] e JavaScript; a necessidade de *plug-ins* externos deveria ser reduzida; e, o HTML5 deveria ser independente para ser interpretado em qualquer dispositivo.

Com estas informações iniciais, observa-se que a união entre duas organizações foi muito importante para o desenvolvimento de páginas estruturadas de forma única e também auxiliou o desenvolvimento de um conteúdo independente de *plug-ins* externos e dispositivos, o que fortaleceu muito o HTML5. Antes disso, as páginas com conteúdo interativo e jogos para *Web* eram, em sua maioria, desenvolvidas através da tecnologia *Adobe Flash*, um *software* para a criação de animações interativas para navegadores *Web* e também *desktops*. Em 2011 foi anunciado que o suporte para a tecnologia *Flash* em dispositivos móveis seria encerrado. Segundo a ADOBE (2014) o HTML5 era universalmente suportado pelos maiores dispositivos portáteis, fortalecendo ainda mais a presença da linguagem de marcação em *tablets* e *smartphones*.

Desde a primeira versão, lançada em 2008, o HTML5 evoluiu muito e continua a evoluir. A W3C apresentou em 2012 um plano para o futuro do HTML5, onde é especificado que uma versão recomendada do HTML 5.0 será lançada no fim de 2014 e que continuará a evoluir até 2016, onde as versões 5.1 e 5.2 estarão em fase de conclusão e desenvolvimento, respectivamente.

[7] DOM é Modelo de Objetos do Documento, é utilizado pelo JavaScript para acessar os objetos do documento. Esse modelo também cria uma hierarquia para o documento, onde os objetos possuem pais e filhos.

1.2 JavaScript

A linguagem de programação JavaScript é comumente utilizada por navegadores *Web* para diversas funcionalidades como: interagir com o usuário; implementação de *scripts client-side*[8]; controlar o navegador, entre outros. Essa linguagem é suportada pelos navegadores mais modernos e é utilizada em conjunto com o HTML5 e CSS para criar W*ebsites* interativos e jogos.

FLANAGAN (2006) descreve em seu livro *JavaScript: The Definitive Guide* que o *JavaScrip*t surgiu no *Netscape* quando um de seus funcionários, Brendan Eich, desenvolveu a linguagem para competir com a *Microsoft* na batalha sob a *Web*, o objetivo principal era criar uma linguagem leve e simples, que usuários inexperientes conseguissem utilizar. Quando a versão beta foi oficialmente lançada, a linguagem se chamava *LiveScript*, nome que foi alterado na versão final para *JavaScript*.

Desde então, o *JavaScript* se tornou a mais popular linguagem de programação para a *Web*, sendo suportada pelos navegadores mais modernos tais como *Google Chrome, Mozilla Firefox, Internet Explorer,* Opera e Safari. Grande parte da popularização do *JavaScript* na programação do *cliente-side* na W*eb* se deve para adoção da Microsoft à linguagem, fato que ocorreu em 1996 quando a Microsoft lançou o Internet Explorer 3.0 com suporte a *JavaScript*, o que representava o reconhecimento da empresa em relação a funcionalidade da linguagem.

Assim como ocorreu com o HTML/HTML5, por ser uma linguagem para a *Web,* o JavaScript passou por muitas alterações independentes ao longo de sua história, onde alguns navegadores poderiam interpretar funções e outros não. Com o objetivo de padronizar o desenvolvimento do *JavaScript* em 2009 foi criado o projeto *CommomJS*. Atualmente a linguagem é oficialmente administrada pela *Mozilla Foundation*, que adiciona novas funcionalidades periodicamente que geralmente são adotadas pelos navegadores mais populares.

1.3 Phaser game engine

A *Phase*r é uma *engine*[9] gratuita e de código aberto, criada por desenvolvedores de jogos que sabem o quão rápido as tecnologias dos navegadores evoluem e, por isso, prometem que ela sempre será atualizada com as mais novas técnicas para facilitar o desenvolvimento para os usuários. Segundo Davey (2014), esta *engine* é projetada para criar jogos que serão executados em navegadores de *desktops* e dispositivos móveis.

Além de ser focada para navegadores de dispositivos móveis, a *Phaser* possui algumas outras características, como: facilidade em carregar componentes de um jogo como imagens, áudio, texturas, *scripts,* entre outros; renderização através de *canvas* e *WebGL* (ambas tecnologias introduzidas com o HTML5); manipulação de áudio com maior facilidade; facilidade para manipular a entrada de dados do usuário (muito importante para um jogo multiplataforma, onde diversas formas de entrada de dados são permitidas) e vários tipos de física incluída com a biblioteca da *engine*.

A renderização em *WebGL*, apesar de ser um grande atrativo para jogos 3D na *Web*, ainda é relativamente nova e não foi implementada em todos os navegadores. O mesmo ocorre com a W*eb* áudio: não está presente em todos os navegadores, principalmente os *mobile.* Isso deve ser um ponto de atenção para aqueles que forem utilizar a ferramenta com o objetivo de criar um jogo para todas as plataformas: nem todas as funcionalidades do HTML5 são aceitas em todos os navegadores.

Phaser é uma ferramenta nova, que surgiu em 2013, e busca facilitar o desenvolvimento para *Web* e multiplataformas sendo fácil de utilizar e bem eficaz, principalmente em jogos 2D para a *Web*. Segundo os desenvolvedores desta ferramenta, quando os navegadores evoluírem passando a suportar *WebGL* nativamente, os jogos W*eb* 3D irão fazer muito sucesso devido o suporte para essa tecnologia.

Apesar de existirem outras *engines* para o desenvolvimento de jogos multiplataforma, como a *Unity,* a ferramenta *Phaser* continua sendo a melhor opção para o desenvolvimento de um jogo que será executado em diversos dispositivos. A justificativa para esta ferramenta ser considerada a melhor é de que enquanto um jogo desenvolvido em *Unity* precisa de um *plug-in* externo para a execução através de um navegador *Web*, um jogo desenvolvido em *Phaser* não necessita de nenhum complemento adicional, além de funcionar sem problemas em navegadores *mobile*.

[8] *Scripts client-side* são scripts que serão executados no computador do usuário, reduzindo o consumo do servidor de um *website*.

[9] Segundo THORN (2011), uma *game engine* é uma ferramenta que integra em um sistema único tecnologias que podem ser utilizadas para a criação de diversos jogos diferentes. Apesar do som e gráficos dos jogos serem distintos, há uma ferramenta suportando esse conteúdo, em sua parte mais genérica. Essa ferramenta é conhecida como *game engine*.

A *Phaser* é uma ótima opção para disponibilizar um jogo no maior número de dispositivos possíveis, pois apesar de ser uma ferramenta relativamente nova, promete facilitar bastante o desenvolvimento de jogos para W*eb* e multiplataforma, sendo fácil de utilizar e bem eficaz. A *engine* está evoluindo junto com os navegadores constantemente, sempre recebendo atualizações e melhorias.

1.4 Jogos multiplataforma

Jogos multiplataforma podem ser compreendidos de diversas formas possíveis. Se considerarmos que um jogo é completamente multiplataforma, ele precisa ser suportado por todo e qualquer sistema ativo no mercado, incluindo *consoles* da última geração, *tablets, smartphones*, computadores e *Smart TVs*. Como exemplo, é possível citar o jogo *Angry Birds*, que suporta todas essas plataformas conforme apresentado no site da desenvolvedora Rovio.

Atualmente, diversas plataformas possuem compatibilidade nativa para as linguagens HTML5 e *JavaScript*, permitindo que um aplicativo ou jogo desenvolvido com apenas essas linguagens seja executado sem problemas. Porém, desenvolver um jogo nativo com apenas HTML5 e *JavaScrip*t para os *consoles* não é muito simples pois cada *console* utiliza uma forma de entrada de dados diferente, possui uma arquitetura computacional diferente de um *smartphone* ou computador, além de possuírem *engines* próprias para desenvolvimento de jogos, tornando o desenvolvimento um pouco mais difícil.

Algumas empresas resolveram esse problema ao lançar *Frameworks*[10] específicos para o desenvolvimento de um jogo com HTML5 e JavaScript nativo para seu sistema, como a Nintendo fez ao lançar a *Nintendo Web Framework*, que possibilita que um jogo desenvolvido em HTML5 seja convertido e executado em um console *Wii U*.

Existe também outra opção para a execução de jogos multiplataforma em um console de última geração. Através do navegador do *console*, é possível acessar a página onde o jogo está hospedado e jogá-lo. Outra opção é acessar algum site especializado em jogos para navegadores de *console*, como o *site Play Boxie,* que permite que os usuários joguem uma grande biblioteca de jogos apenas identificando o sistema utilizado. Porém, é importante destacar que os navegadores de *consoles* não são tão robustos e muitas vezes são mais limitados do que os navegadores de dispositivos móveis, como consequência disso, é possível que os jogos tenham comportamentos diferentes nestes dois ambientes distintos.

Para esse trabalho, considera-se como multiplataforma dispositivos móveis que executem os sistemas operacionais *Android, Windows Phone* e *iOS* com um bom desempenho, além de considerar também as *Smart TVs* e os computadores *desktop.* A compatibilidade com estes ambientes proporcionará uma grande gama de dispositivos para teste e desenvolvimento.

2 PROTOTIPAÇÃO E TESTES DO PROTÓTIPO

Segundo WARFEL (2009), um protótipo é um modelo representativo ou simulação de um sistema/aplicativo final. Ao começar a desenvolver um jogo o primeiro passo para o seu desenvolvimento é a elaboração de um documento contendo todas as especificações do jogo de forma escrita, geralmente chamado de *Game Design Document* (GDD). Porém, esse documento falha na ilustração visual de como tudo será realizado, o que pode gerar uma interpretação incorreta do que é desejado.

Um protótipo do jogo busca evitar que a interpretação incorreta ocorra, demonstrando de forma prática como o *gameplay* do jogo funcionará para toda a sua equipe. Com a compreensão de todos os membros da equipe, o time de desenvolvimento economizará tempo, esforço e dinheiro, pois não haverá retrabalho por má interpretação, além de incentivar novas ideias, por que é na fase de prototipação onde muitas ideias surgem.

Neste contexto é perceptível a importância da utilização de protótipos no desenvolvimento de um determinado produto. Tendo em vista que estes permitem a realização de testes que possibilitam a verificação de possíveis problemas, ou mesmo a oportunidade de otimização das suas funções. Desta forma, a prototipação se destaca como uma poderosa ferramenta, que ajuda não só ao desenvolvimento mais rápido e seguro, como também poderá evitar drásticas perdas financeiras. (FILHO; BENICIO; CAMPOS; NEVES; 2013).

Apesar disso, os protótipos não se limitam apenas à utilização na fase de testes de usabilidade para detecção de erros ou oportunidades de otimização. De acordo com BUSKIRK e MORONEY (2003), o protótipo pode ser usado para validar os requisitos do sistema com seu cliente, auxiliando na criação de

[10] Um *framework* é uma biblioteca de classes que capturam padrões de interação entre objetos. Ele consiste em uma suíte de classes concretas e abstratas, explicitamente projetadas para serem utilizadas juntas. (YACOUB; AMMAR, 2004, tradução nossa).

manuais e no treinamento para a equipe que dará suporte ao sistema. Em um jogo, um protótipo de alta fidelidade pode ser utilizado para promover o produto final e chamar atenção de seus consumidores.

Como HOM (1998) descreve em seu artigo *The Usability Methods Toolbox Handbook*, a prototipação é uma técnica que pode ser utilizada em qualquer momento do desenvolvimento, porém em uma fase mais avançada de ela pode ser menos eficiente por ser mais fácil utilizar uma versão anterior do sistema para testes. O autor classifica a técnica de prototipagem em algumas categorias, demonstradas abaixo:

- **Prototipação rápida:** metodologia que rapidamente desenvolve novos *design*s para o sistema, avalia esses *design*s e depois os descarta quando o novo é desenvolvido em conjunto com novos protótipos;
- **Prototipação reutilizável:** também conhecida como prototipação evolucionária. Nessa técnica de prototipação, o esforço utilizado para construir o protótipo praticamente não é perdido, pois parte (ou tudo) do protótipo será utilizado para fazer o sistema final;
- **Prototipação modular:** também conhecida como prototipação incremental. Novas partes são adicionadas enquanto o ciclo do *design* progride;
- **Prototipação horizontal:** o protótipo irá cobrir uma grande parte das funcionalidades e funções do sistema, porém a maioria não estará funcionando. Ótima para testar telas e *interfaces,* onde será possível verificar todo o fluxo do sistema sem que seja necessário que as opções funcionem;
- **Prototipação vertical:** o protótipo irá cobrir uma pequena parte das funcionalidades e funções do sistema, porém elas funcionarão. Ótima para testar pequenas partes do sistema;
- **Prototipação de baixa fidelidade:** protótipo que é construído através de caneta e papel, assim simulando como o sistema funcionará, mas não se parecendo em nada o produto final; e,
- **Prototipação de alta fidelidade:** protótipo que é desenvolvido para se parecer o máximo possível com o produto final em todos os aspectos.

É a função do *game designer* decidir qual tipo de prototipação utilizar durante o desenvolvimento de seu jogo, pois cada tipo funciona com mais eficácia de acordo com a fase do projeto. Por exemplo, é comum observar protótipos de baixo nível de fidelidade na fase de pré-produção de um jogo, que se caracteriza pela fase inicial de desenvolvimento e a formação de ideias, e protótipos de alto nível de fidelidade na fase de produção, onde os conceitos e mecânicas do jogo já foram aprovados.

Logo depois da fase de prototipação é realizada a fase de testes com o protótipo que foi desenvolvido. Esse processo é extremamente importante para o jogo, pois é nele onde tudo o que foi planejado em seu protótipo será testado e corrigido para ser desenvolvido na versão final do mesmo. Seja um protótipo de alta ou baixa fidelidade, ele deve ser testado e ajustado de acordo com o desejo do *game designer*.

De acordo com SCHELL (2008), existem quatro tipos de testes realizados na indústria de jogos digitais, descritos abaixo:

- **Grupos de foco:** comumente utilizados em projetos comerciais. São testes realizados pelas empresas para identificar se a ideia do jogo é atrativa para o consumidor. É realizada através de entrevistas com grupos de potenciais jogadores;
- **Teste de garantia de qualidade:** são testes realizados para garantir a qualidade do produto (remoção de erros e *bugs*), seja do produto final ou, caso seja um protótipo reutilizável, do próprio protótipo para que seja evoluído para o produto final;
- **Testes de usabilidade:** são testes realizados para garantir a qualidade do *design* do produto. Esse tipo de teste busca validar a facilidade de navegar pela interface do jogo e o quanto ele é intuitivo para o usuário; e,
- **Teste de jogabilidade:** são testes realizados com o jogo em fase beta ou com um protótipo de alta fidelidade do mesmo. Esses testes devem avaliar a experiência do usuário através da jogatina de acordo com o que foi projeto pelo *game designer*.

Durante os testes, devem ser criados cenários de teste para verificar o que o usuário conseguirá fazer no sistema de acordo com o que ele pode fazer. Nos testes de usabilidade, por exemplo, é necessário se assegurar que o protótipo foi desenvolvido com o *design* focado no usuário para que ele saiba o que fazer e como fazer: “um bom cenário de teste é focado no objetivo, enquanto acomoda as atividades e o processo necessário para atingir esse objetivo” WARFEL (2009).

Já nos testes de garantia de qualidade e jogabilidade é muito importante anotar todos os erros e *bugs* encontrados no sistema. O ideal é que alguém realize os cenários de teste enquanto outra pessoa faça as anotações de acordo com o que o usuário está possuindo facilidade ou dificuldade. Todas as anotações ajudarão o desenvolvedor a polir o jogo em sua etapa de desenvolvimento.

2.1 Prototipação em um jogo multiplataforma

Como um jogo multiplataforma será exportado para diversas plataformas, é essencial que o protótipo já contenha uma resolução ideal para o maior número de dispositivos no mercado ou que ele possua um sistema para redefinir o tamanho de seus recursos de acordo com o tamanho da tela. Como é possível observar na figura 1, a Phaser possui funções preparadas para alterar o tamanho do *canvas* e através de três linhas de código é possível ajustar o jogo de acordo com o tamanho da tela do dispositivo em que ele está sendo executado.

Figura 1 – Definindo a resolução do jogo

```
//define o modo de scale do jogo para preencher totalmente
this.game.scale.scaleMode = Phaser.ScaleManager.EXACT_FIT;
//define o scale do tamanho da tela do dispositivo
this.game.scale.setScreenSize();
//atualiza o scale
this.game.scale.refresh();
```

Os *inputs* ou entradas do jogo devem ser definidos desde seu protótipo, em um jogo multiplataforma é necessário considerar todos os dispositivos e suas limitações, como ausência de teclado e *mouse* em um *smartphone*. É possível observar duas tendências em jogos desenvolvidos para *desktops* e *mobiles*, em jogos como *Angry Birds* todos os *inputs* são realizados através do toque em dispositivos *mobile* e simulados através do *mouse* em *desktops*, como é possível observar na figura 2.

Figura 2 - Controles de toque em Angry Birds

Porém, em jogos com comandos mais complexos como controlar a velocidade que o personagem corre ou mirar em algum local especifico (geralmente em jogos de tiro), os desenvolvedores optam por realizar os *inputs* através do teclado em *desktops* e simular esses comandos em dispositivos *mobile* através de um controle virtual, como é o caso de GTA: San Andreas (figura 3).

Figura 3 - Controles simulados na tela do *tablet*

O protótipo deve ser jogável em todas as plataformas que ele será lançado, pois é necessário garantir que todos os testes iniciais sejam realizados, garantindo a qualidade final do produto. Como as

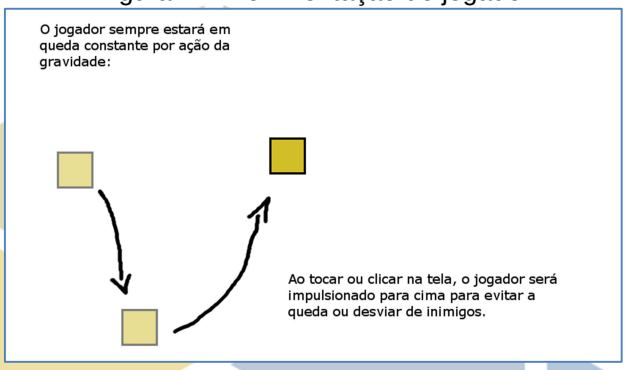
versões de navegadores não são as mesmas para *desktops* e *mobile* é possível que alguma funcionalidade esteja presente apenas em uma e apresente *bugs* em outra, se os testes não forem realizados na primeira fase do desenvolvimento, a correção desses *bugs* será mais trabalhosa em um estágio avançado de desenvolvimento.

2.2 Estudo de caso: prototipação e testes

O jogo escolhido para desenvolvimento e demonstração das técnicas utilizadas com *Phaser engine* se chama *Rocket Komodo*. O gênero do jogo é *Infinite Runner 2D*, que são jogos caracterizados por não possuir fim, e em que o desafio é conseguir a maior pontuação possível. No início do desenvolvimento do jogo foram criados dois protótipos. O primeiro com o objetivo de apresentar as funcionalidades de *gameplay* e o segundo os menus e opções do jogo.

Para demonstrar a jogabilidade do projeto foi criado um protótipo horizontal de baixa fidelidade, ele foi desenhado utilizando o *software* de criação de imagens GIMP e seu objetivo foi cobrir todas as possibilidades que o jogador possui no jogo e todos os desafios que ele encontrará ao jogar. Na figura 4 é possível observar os movimentos do jogador.

Figura 4 – Movimentação do jogador



Nas figuras de 5 a 9 é possível observar os perigos que foram idealizados para atrapalhar o jogador:

Figura 5 – Movimentação do jogador

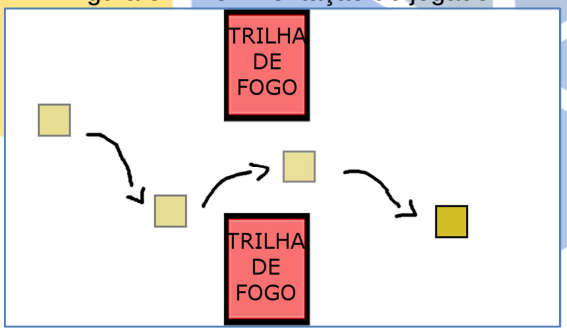


Figura 6 – Pedras

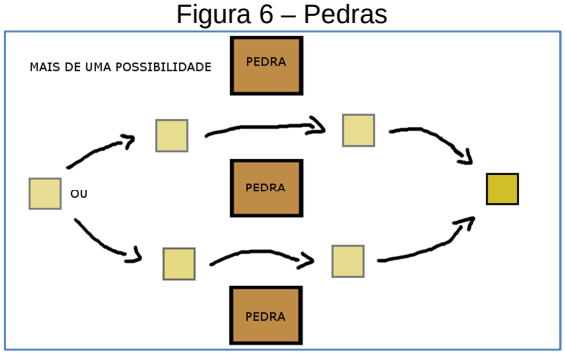
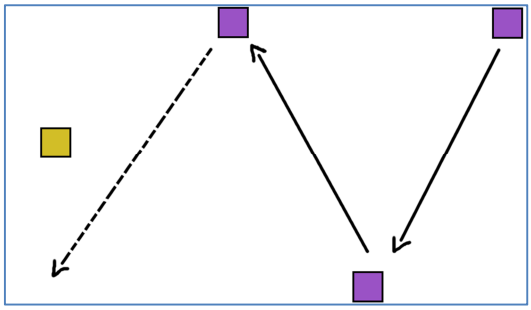


Figura 7 – Morcego



Para auxiliar o jogador, foram incluídos no protótipo dois *power-ups* demonstrados nas figuras 7 e 8.

Figura 8 – Escudo

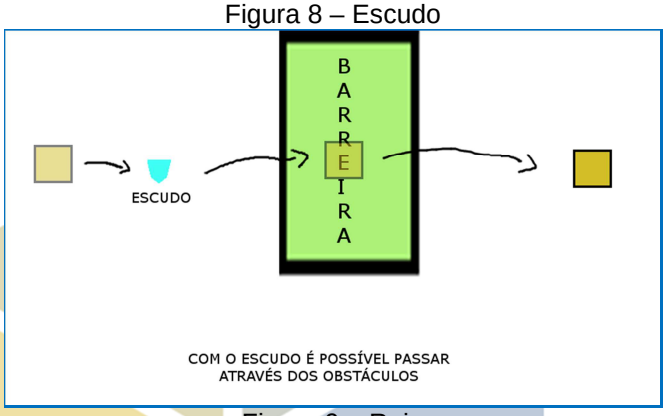
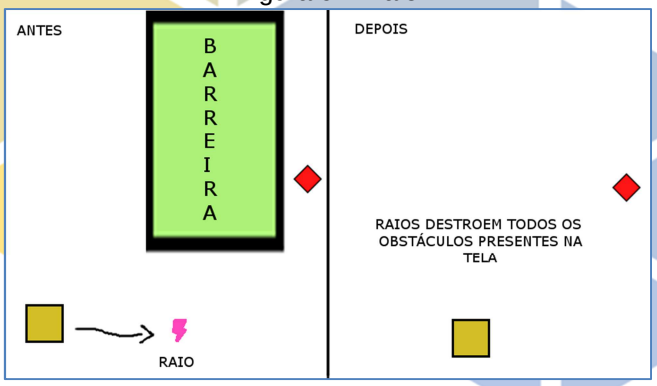
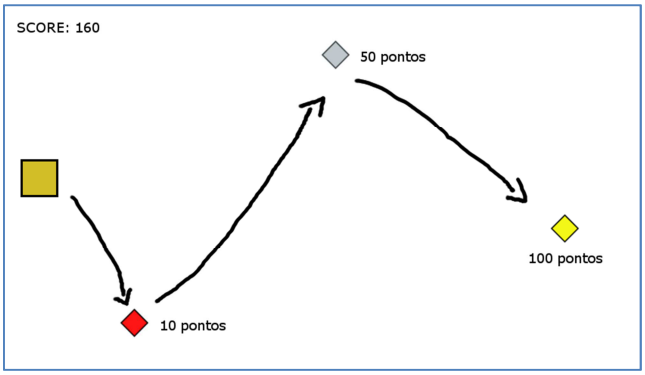


Figura 9 – Raio



O jogador poderá obter mais pontuação coletando as joias distribuídas pelo cenário, como apresentado na figura 10.

Figura 10 – *Power-Ups*



Depois de realizar todos os desenhos, todas as funcionalidades de *gameplay* do jogo estavam presentes no protótipo. Além disso, foi demonstrado no protótipo como o jogo será controlado nos dispositivos que ele será executado.

Para apresentar a forma em que o menu e as opções serão demonstrados na tela foi criado um protótipo reutilizável de baixa fidelidade. O objetivo é preparar o *design* do jogo para todos os testes de usabilidade e já corrigi-los na primeira fase do desenvolvimento. Esse protótipo foi criado usando a ferramenta *Phaser* e o sistema de estados que a *engine* oferece, transformando esta etapa em uma tarefa mais fácil. Na figura 11 é possível observar a estrutura de um estado no código e como funciona sua inclusão e execução no mesmo:

Figura 11: Estrutura de um estado no código

```
//declara um novo estado
var novo_estado = {
    preload: function(){
        //função de carregamento do jogo, será executada antes de todas as outras
    },
    create: function(){
        //função de criação, será executada uma vez antes de iniciar a atualização
    },
    update: function(){
        //função de atualização, será executada 60 vezes por segundo
    }
};

//adiciona o novo estado ao jogo com o nome de "novoEstado"
this.game.state.add('novoEstado', novo_estado);

//executa o novo estado adicionado ao jogo
this.game.state.start('novoEstado');
```

Foram criados três estados de jogo no protótipo: *menu*, tutorial e *play*. Eles apresentam o *menu*, o tutorial e o *gameplay* do jogo, representados pelas figuras 12 a 14 respectivamente.

Figura 11 - Menu

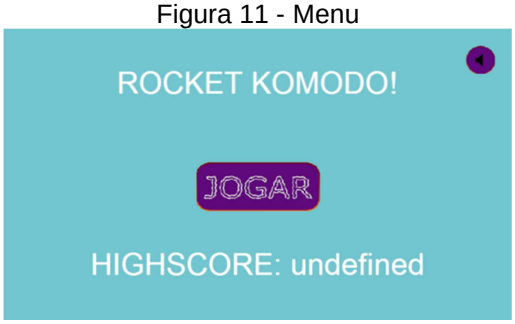


Figura 12 - Tutorial

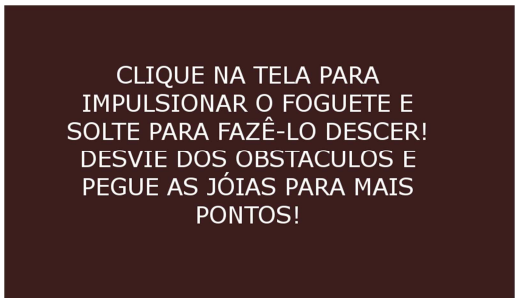
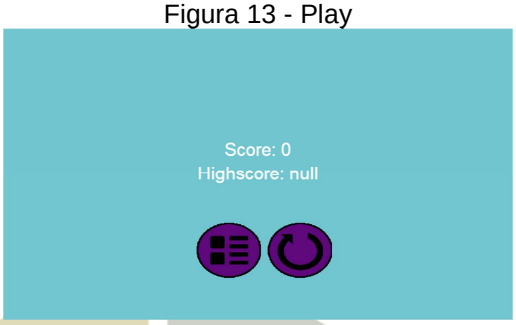
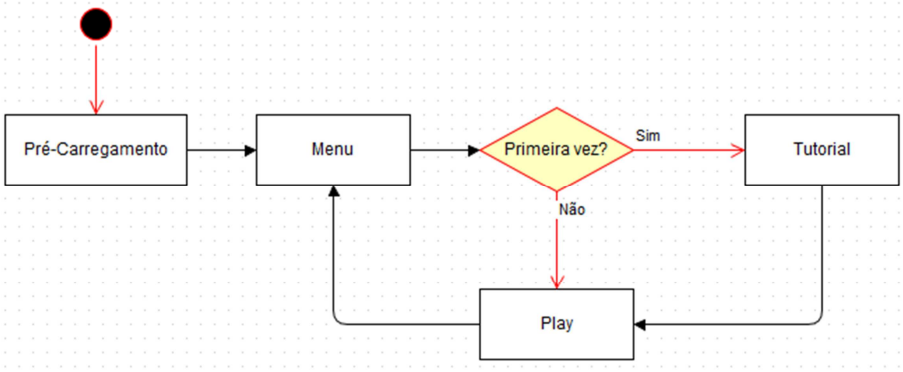


Figura 13 - Play



O diagrama de estados na figura 15 representa os estados que o protótipo possui e como eles funcionarão:

Figura 14 - Diagrama de estados



Com o protótipo desenvolvido foi possível iniciar os testes de usabilidade e se assegurar que o fluxo desejado fora atingido no desenvolvimento. Como o jogo possui apenas três estados diferentes, os testes em relação ao fluxo foram realizados de forma rápida em todos os dispositivos e, através de poucos cenários de teste, foi possível constatar que o protótipo funcionará como esperado.

Nos *smartphones*, que possuem uma tela menor, foi identificado que os botões e letras do jogo estavam muito pequenos e quase ilegíveis, principalmente em aparelhos com telas menores que três polegadas. Este problema afeta diretamente a experiência do jogador, que pode não saber o que fazer no jogo por não conseguir entender o que foi explicado, ou não conseguir entender o que um botão significa. Assim que identificado, o problema foi anotado e definido como prioridade no início do desenvolvimento do jogo.

3 DESENVOLVIMENTO E CONSIDERAÇÕES

Conforme descrito por NOVAK (2011) em seu livro *Game Development Essencials: an introduction,* o processo interativo de desenvolvimento — usado no desenvolvimento de *software* e W*eb* — também funciona bem para o desenvolvimento de jogos. Esse modelo incorpora um processo circular de três estágios: *design*, protótipo e avaliação.

O processo interativo ocorre de forma circular até o fim do desenvolvimento do jogo. Primeiro é definido um *design* que será aplicativo praticamente em um protótipo que será avaliado e, caso aprovado,

será aprimorado através de outro *design* e assim sucessivamente, até que o protótipo se transforme em uma versão *gold*[11]. Esse modelo funciona com facilidade no desenvolvimento de jogos multiplataforma em HTML5, pois o código-fonte do jogo desenvolvido não necessita de adaptações para serem executados em outros dispositivos, em consequência os protótipos desenvolvidos podem ser evoluídos de acordo com a necessidade de todas as plataformas que o jogo será lançado.

Dessa forma, o protótipo (beta) deve ser testado em todas as plataformas que o jogo será lançado, principalmente nos dispositivos *mobile* que não possuem especificações tão robustas quanto os computadores, pois, como o código-fonte do jogo será o mesmo para todas as plataformas desenvolvidas, é necessário fazer com que o jogo seja executado com eficiência em todas elas. Através dos testes é possível detectar queda de desempenho ou o mau funcionamento de alguma função do jogo e corrigi-los.

Considerando o *hardware* limitado e requisitos únicos do sistema operacional de telefones móveis, jogos neles são diferentes de jogos na maioria de outros dispositivos. Portanto, o desenvolvimento do jogo deve ser ajustado para as necessidades especificas destes dispositivos (MORRISON, 2004, tradução nossa).

Um dos grandes desafios para o desenvolvimento do projeto prático esteve em manter o desempenho estável nas plataformas móveis e, para isso, foram utilizadas algumas funções disponíveis na *Phaser Game Engine*, como por exemplo a função de pré-carregamento de componentes, observado na figura 16.

Figura 15 - Função de pré-carregamento

```
preload: function(){
//carrega uma imagem {parâmetros: id do arquivo, endereço do arquivo}
this.game.load.image('bird','assets/images/bird.png');
//carrega um audio {parâmetros: id do arquivo, endereço do arquivo}
this.game.load.audio('rocket','assets/sound/rocket1.ogg');
//carrega uma spritesheet {parâmetros: id do arquivo, endereço do arquivo, largura x do frame,
largura y do frame, número de frames}
this.game.load.spritesheet('medals', 'assets/images/medalhas.png', 40, 80, 9);
}
```

Através do carregamento prévio dos recursos do jogo é possível reduzir o consumo de memória durante a execução do jogo, evitando travamentos e queda de *frames* por segundo (FPS).

Além disso, para garantir ainda mais a execução sem quedas de FPS e de forma suave, foi realizada a criação de grupos para todos os inimigos do jogo. Na *Phaser*, um grupo é um *container* para guardar objetos que permite uma rápida reciclagem de elementos. Dessa forma, ao invés de realizar a criação de um novo inimigo a todo tempo durante a execução do jogo, são criados cinco inimigos no início do jogo que serão reciclados durante sua execução, poupando memória. A criação de um grupo e reciclagem de seus objetos é feita de uma forma bem simples, como demonstrado na figura 17.

Figura 16 - Criação de um grupo e reciclagem de objetos

```
create: function() {
  //cria um grupo chamado pedras
  this.pedras = game.add.group();
  //cria 5 elementos dentro do grupo “pedras” com a sprite cuja key é pedra (por padrão, todos os elementos do
grupos estarão mortos)
  this.pedras.createMultiple(5, 'pedra');
}

//função que ilustra a criação de uma pedra no meio da tela do jogo
adicionar_pedra: function() {
  //a variavél “pedra” será o primeiro elemento morto do grupo
  var pedra = this.rockets.getFirstDead();
  //define que, se caso a pedra saia da tela, ela morrerá
  pedra.outOfBoundsKill = true;
  //define a posição da pedra na tela
  pedra.reset(this.game.world.x/2, this.game.world.x/2);
}
```

[11] Uma vez que o jogo passou da fase beta, ele é considero *gold.* O jogo é enviado para ser manufaturado depois de intensamente testado e considerado ser aceitável. NOVAK (2011).

Além das questões de desempenho, outro ponto crítico no momento do desenvolvimento de um jogo cujo código será totalmente reaproveitado em todas as plataformas, é como o *input* (ou interações de entrada) do usuário será recebido em todas as plataformas em que o jogo será executado. No projeto desenvolvido, o único modo de *input* é através do *mouse* em computadores e através de *touch* em dispositivos móveis, dessa forma foi possível utilizar apenas a função apresentada na figura 18 para a detecção de entrada de dados.

Figura 17 - Função para a detecção de entrada de dados

```
//verifica se o ponteiro ativo está pressionado
if(this.game.input.activePointer.isDown)
   {
      //executa alguma função
   }
```

O ponteiro ativo sempre será dinâmico de acordo com a plataforma que o projeto está sendo executado, dessa forma não será necessário identificar através de código qual o dispositivo do usuário para a entrada de dados. Como é possível observar, as funções disponíveis através da *Phaser Game Engine* auxiliam de forma considerável na facilidade e velocidade do desenvolvimento de jogos multiplataforma.

3.1 Aspectos positivos do desenvolvimento com Phaser

As novas tecnologias introduzidas com o HTML5 permitiram desenvolvedores a criar jogos e páginas *Web* muito mais complexas e visualmente atrativas para os usuários. No livro *HTML5: Up and Running*, PILGRIM (2010) explora todas as novas características do HTML5, como a nova semântica, o *Canvas* e o armazenamento local de dados, e como essas tecnologias devem ser utilizadas nas mais diversas situações.

Porém, apesar de todas as melhorias, a combinação HTML5 e Javascript sem a utilização de nenhuma *engine* não é a melhor opção para a criação de um jogo com complexidade média ou alta. A ausência de algumas funções básicas para o desenvolvimento de um jogo irá exigir mais tempo de desenvolvimento e aumentará a complexidade do projeto. A *Phaser* contempla uma diversidade de funções para facilitar o desenvolvimento de um jogo exigindo ao desenvolvedor apenas a preocupação com o desenvolvimento das mecânicas de seu jogo.

É possível realizar o pré-carregamento de requisitos com apenas HTML5 e Javascript, porém é necessário criar uma função que realize o carregamento dos componentes e calcule a porcentagem estimada do processo. Ao utilizar a *Phaser*, apenas uma linha de código é necessária para o pré-carregamento de um componente, como já demonstrado na figura 16, e as funções para gerenciar o carregamento já estão prontas para o uso. O pré-carregamento de componentes é muito importante para evitar o consumo de memória durante a execução do jogo, momento em que ele estará executando diversas outras funções a cada segundo.

De acordo com a proposta de um jogo, a física pode ser um componente muito importante. No cenário de mercado atual, onde "[...] consumidores são inflexíveis e querem mais realismo físico." (EBERLY; SHOEMAKE, 2004), criar funções que simulam a física podem se tornar demasiadamente complexo e dificultar o desenvolvimento de um jogo. A *Phaser* conta com três sistemas de física, todos com gravidade e funções de colisão (descritos abaixo), já integrados com a *engine* para o desenvolvimento de jogos que necessitem de física.

- **ARCADE**: é o sistema de física mais simples dos três, todo corpo que utiliza o sistema de física arcade possuirá um corpo retangular sem rotação, é geralmente utilizado para colisões rápidas ou colisões sem complexidade. Por ser a mais simples, é a que menos consume memória;
- **NINJA**: é semelhante à ARCADE, porém possui suporte a corpos circulares e rotação. Se necessário, é possível calcular o tamanho exato de um corpo com o sistema NINJA. Por ser o mais preciso, é o mais lento; e,
- **P2**: é o sistema de física mais completo da *Phaser*. Conta com suporte total a corpos físicos e materiais (elementos que irão afetar o modo que um corpo reage a outro corpo). Atualmente é o que mais recebe atualizações e suporte.

Além da classe responsável por elementos da física, a *Phaser* também conta com uma classe completa de áudio que facilita consideravelmente a utilização de sons e músicas (elementos extremamente importantes) em jogos. Apesar da implementação de áudio não ser complexa sem a utilização da *engine*, sua utilização traz diversas vantagens como: facilidade de adição de sons, controle de volume, tocar sons em *loop*, execução de funções ao término de algum som, pausar sons, entre outros.

Todas essas funcionalidades comprovam o quanto a *Phaser* está preparada para o desenvolvimento de diversos tipos de jogos para a *Web*. Ao contrário de *engines* como *Construct 2* que possuem

características semelhantes com interface amigável e que não precisam de programação (e são limitadas exatamente por esse motivo), a *Phaser* proporciona total liberdade ao desenvolvedor e facilidade na execução de tarefas que sem sua utilização seriam complexas.

3.2 Aspectos negativos do desenvolvimento com *Phaser*

Mesmo apresentando diversas vantagens, o desenvolvimento com a *Phaser* também possui alguns aspectos negativos, um deles ocasionado por um problema antigo no desenvolvimento *Web*: a incompatibilidade de funcionalidades em determinados navegadores. De acordo com FLANAGAN (2006), a programação no *cliente-side* através de JavaScript foi sempre um combate com a incompatibilidade.

Com o surgimento de novos dispositivos móveis, constantemente novas versões de navegadores são desenvolvidas aumentando a gama de dispositivos disponíveis no mercado e, consequentemente, agravando o problema de compatibilidade já existente. Além disso, é importante destacar que os navegadores de dispositivos se comportam de forma diferente comparado ao *desktop:*

Por exemplo, com um navegador *desktop* o desempenho se comparará à velocidade do computador; em um celular móvel, no entanto, seus recursos são estritamente limitados. Coisas como transições com CSS3, que são uma ótima maneira de apresentar a navegação entre páginas, requerem uma grande quantidade de memória para processar. Preste atenção para a alocação de memória; é possível que sua aplicação *Web* vá travar em alguns navegadores. (AVOLA; RAASCH, 2012, tradução nossa).

Considerando todos esses fatores, é seguro afirmar que nem todas as funcionalidades disponíveis na *Phaser Game Engine* funcionam completamente em navegadores de dispositivos móveis, apesar de existir um trabalho contínuo para extinguir esses problemas. Segundo a *Photon Storm*, empresa que criou a *Phaser*, antes de uma funcionalidade ser adicionada à *engine* sempre são realizados diversos testes na maior variedade de navegadores possíveis, porém com ascensão do mercado no ritmo atual, é difícil manter o controle sobre a compatibilidade em todos os navegadores disponíveis.

Ao citar a incompatibilidade de determinadas tecnologias em alguns navegadores, é imprescindível falar sobre o desafio das principais *engines* disponíveis no mercado em relação a jogos na *Web*: facilitar o desenvolvimento de jogos 3D jogáveis diretamente no navegador. A *engine Unity* já consegue realizar essa tarefa, porém é necessário que o usuário faça *download* de um *plug-in*, enquanto a *PlayCanvas* promete jogos 3D com melhor desempenho do que a *Phaser* através da tecnologia *WebGL* sem o uso de *plug-ins*.

Apesar de ser uma excelente ferramenta para o desenvolvimento de jogos multiplataforma 2D, a Phaser não está totalmente preparada para a execução de jogos completos em 3D. É possível realizar a renderização de elementos na tela, porém a *engine* não está preparada para realizar testes de física nos mesmos, por exemplo. Parte disso se deve ao foco nos navegadores de dispositivos móveis, cuja necessidade de suporte ao *WebGL* não é prioridade.

Na tabela 1 a seguir é possível observar uma comparação entre os aspectos positivos e os aspectos negativos presentes no desenvolvimento de jogos com a Phaser:

Tabela 1 – Aspectos positivos x aspectos negativos

Funcionalidade	Positivo/Negativo	Considerações
Física integrada	Positivo	Auxilia no desenvolvimento de jogos e dá liberdade ao desenvolvedor
Pré-carregamento de Componentes	Positivo	Auxilia na melhoria do desempenho e sua utilização é simples
Manipulação de Áudio	Positivo	Fácil de utilizar e bastante controlável
Incompatibilidade de funções	Negativo	Alguns navegadores não aceitam as funções da Phaser, dificultando o desenvolvimento
Pouco suporte aos jogos 3D	Negativo	Com o WebGL disponível para o HTML5, se espera que a *engine* possua um suporte à essa tecnologia

Fonte: Próprio autor

3.3 Principais funções desenvolvidas no projeto prático

Durante o desenvolvimento do projeto proposto neste trabalho, foram criadas diversas funções utilizando várias funcionalidades disponibilizadas pela *Phaser*. A utilização desta *engine* tornou o desenvolvimento mais fácil e, consequentemente, mais rápido. Uma das principais funções desenvolvidas foi a *gas,* representada na figura 19, cuja responsabilidade é controlar a movimentação do jogador toda vez que for realizado um toque ou clique na tela.

Figura 18 - Função *gas*

```
gas: function() {
        //se o player não estiver vivo, não faz nada
        if (this.player.alive == false)
            return;

        //se estiver, diminiu a aceleração por 10
        this.playerAcceleration -= 10;
        //aumenta a velocidade Y (vertical) em 200 mais a aceleração
        this.player.body.velocity.y = -200 + this.playerAcceleration;
        //rotaciona o player para -30 graus em 200 milisegundos
        this.game.add.tween(this.player).to({angle: -30}, 200).start();
    }
```

A variável *this.playerAcceleration* foi criada para dar mais naturalidade ao movimento do jogador, fazendo com que ele se mova rápido no início do movimento e diminua a velocidade conforme o movimento continuar. Dessa forma, foi possível fazer com que o jogador desviasse dos objetos que vem em sua direção, porém era necessário que esses obstáculos fossem adicionados de forma aleatória ao jogo. Para isso foi criada a função *add_hazard*, que pode ser observada na figura 20.

Figura 19 - Função *add_hazard*

```
add_hazard: function(){
        //calcula um número aleatório entre 1 e 50
        this.rand = this.game.rnd.integerInRange(1,50);
        //adiciona pontos ou power ups
        this.add_rock_and_power_ups();

        //se o jogo não acabou, adiciona um ponto na pontuação do jogador e atualiza o texto na tela
        if(!gameOver)
        {
            score++;
            this.label_score.setText(score);
        }

        //de acordo com o número gerado, o jogo irá adicionar um obstáculo
        if(this.rand <= 10)
        {
            this.add_mountain();
        }
        else if (this.rand > 10 && this.rand <= 20)
        {
            this.add_inverted_mountain();
        }
        else if (this.rand > 20 && this.rand <= 30)
        {
            this.add_bouncing_guy();
        }
        else if (this.rand > 30 && this.rand <= 40)
        {
            this.add_three_rockets();
        }
        else
        {
            this.add_fire_trail();
        }
```

```
    }
```

Essa função basicamente gera um número aleatório e, de acordo com esse número, chama uma função para criar um obstáculo ao jogador. Uma dessas funções pode ser observada na figura 21, onde a função *add_mountain* é apresentada.

Figura 20 - Função *add_mountain*

```
add_mountain: function() {
    //pega o primeiro elemento "morto" do grupo "mountains"
    var mountain = this.mountains.getFirstDead();
    //adiciona a montanha fora da tela (posição x = 840) com uma altura aleatória entre 130 e 180 (posição y)
    mountain.reset(840,this.game.rnd.integerInRange(130, 180));
    //define a velocidade x como -200 (irá andar da direita para a esquerda)
    mountain.body.velocity.x = -200;
    //define que o objeto será destruido se sair da tela
    mountain.outOfBoundsKill = true;
    //define que o objeto é imóvel (não sofre reação em uma colisão)
    mountain.body.immovable = true;
  }
```

Como pode ser observado, essa função realiza a reciclagem de objetos que já foram criados previamente através da função *getFirstDead*. Essa função é exclusiva para grupos e sempre retornará o primeiro elemento "morto" do grupo, tornando a reciclagem de recursos bem simples. Na função *add_mountain*, uma "montanha" é criada para o jogador desviar, porém se executássemos o jogo apenas com o que foi apresentado, os objetos não iriam colidir com o jogador. Para que a colisão fosse verificada pelo jogo foi adicionada a seguinte condição (figura 22) na função *update*[12] do jogo.

Figura 21 - Verificação de colisão

```
//função executada a todo frame durante a execução do jogo

//se o player não estiver invencivel, faz a verificação de colisão com todos os grupos existentes
if(this.player.invencible == false)
    {
        this.game.physics.arcade.collide(this.player, this.rockets, this.hit_object, null, this);
        this.game.physics.arcade.collide(this.player, this.fireTrail, this.hit_object, null, this);
        this.game.physics.arcade.collide(this.player, this.invertedFireTrail, this.hit_object, null, this);
        this.game.physics.arcade.collide(this.player, this.bouncingGuys, this.hit_object, null, this);
        this.game.physics.arcade.collide(this.player, this.mountains, this.hit_object, null, this);
        this.game.physics.arcade.collide(this.player, this.invertedMountains, this.hit_object, null, this);
        this.game.physics.arcade.collide(this.player, this.engines, this.hit_object, null, this);
        this.game.physics.arcade.collide(this.player, this.invertedEngines, this.hit_object, null, this);
    }

//essas colisões são verificadas mesmo com o jogador invencivel pois são as colisões com os pontos e power ups
    this.game.physics.arcade.overlap(this.player, this.rarePoints, this.add_points, null, this);
    this.game.physics.arcade.overlap(this.player, this.mediumRarePoints, this.add_points, null, this);
    this.game.physics.arcade.overlap(this.player, this.points, this.add_points, null, this);
    this.game.physics.arcade.overlap(this.player, this.shields, this.add_powerUp, null, this);
    this.game.physics.arcade.overlap(this.player, this.bolts, this.add_powerUp, null, this);
```

Para a verificação da colisão foi utilizada a função *physics.arcade.collide* que possui os seguintes parâmetros: primeiro objeto ou grupo para verificar a colisão, segundo objeto ou grupo para verificar a colisão, função que será executada ao colidir, função de verificação que retornará verdadeiro caso a colisão possa ocorrer e falso caso contrário, contexto que as funções anteriores serão executadas. Com essa função foi possível identificar a colisão do jogador com obstáculos adicionados, mas sem a função de retorno nada iria ocorrer.

[12] Função presente na maioria das *engines* atuais. Tudo o que está dentro dessa função é executado a cada *frame* pelo jogo.

Como observado na figura 22, a função *hit_object* é chamada toda vez que o jogador colide com algum dos grupos de objetos do jogo e é nela em que toda a lógica de fim de jogo acontece (figura 23)

.

Figura 22 - Função *hit_object*

```
hit_object: function() {
        //se o player não estiver vivo, não faz nada
        if (this.player.alive == false)
            return;

        //pausa o som do foguete
        this.rocketSound.pause();
        //define o jogador como "morto"
        this.player.alive = false;
        //remove o timer que adiciona os obstaculos a cada 4 segundos
        this.game.time.events.remove(this.timer);
    }
```

Com o jogador morto nenhuma entrada de dados para controlar o personagem será reconhecida e os botões para reiniciar o jogo serão apresentados, forçando o reinício da partida. Dessa forma, todo o ciclo do jogo já estava fechado, porém foi adicionada mais uma função para deixar a experiência de jogo mais dinâmica. A função *add_powerUp* altera as características do jogador ou do ambiente para facilitar o jogo, seu código pode ser observado na figura 24.

Figura 23 - Função *add_powerUp*

```
add_powerUp: function(player, powerUp){
        //se o tipo do power up for shield, deixa o jogador invencivel
        if(powerUp.tipo=='shield')
        {
            //destroi o power up
            powerUp.kill();
            //define o jogador como invencivel
            this.player.invencible = true;
            //adiciona um timer que irá executar a função que removerá o atributo "invencivel" do jogador
            this.shieldTimer = this.game.time.events.add(10000, this.resetShield, this);
        }
        //se o tipo do power up for bolt, destrói todos os objetos na tela
        else if(powerUp.tipo=='bolt')
        {
            //destrói o power up
            powerUp.kill();
            //adiciona a sprite claridade na tela. Essa sprite é do tamanho de toda a tela e tem a animação de
clarear a tela por completo
            this.claridade = this.game.add.sprite(0,0,'claridade');
            //adiciona um timer que irá destruir a sprite Claridade em 100 milisegundos
            this.boltTimer = this.game.time.events.add(100, function(){this.claridade.destroy();}, this);

            //para cada elemento "vivo" do grupo, a Phaser irá executar a função abaixo
            this.rockets.forEachAlive(function(item){
                //destrói o elemento
                item.kill();
            });
            this.engines.forEachAlive(function(item){
                item.kill();
            });
            this.invertedEngines.forEachAlive(function(item){
                item.kill();
            });
            this.fireTrail.forEachAlive(function(item){
                item.kill();
            });
            this.invertedFireTrail.forEachAlive(function(item){
                item.kill();
            });
            this.bouncingGuys.forEachAlive(function(item){
```

```
            item.kill();
        });
        this.mountains.forEachAlive(function(item){
            item.kill();
        });
        this.invertedMountains.forEachAlive(function(item){
            item.kill();
        });
    }
}
```

A função exclusiva para grupos *forEachAlive* da Phaser auxiliou o desenvolvimento da função *add_powerUp*, pois através da primeira função é possível encontrar cada elemento "vivo" do grupo e realizar alguma ação com esse elemento. No caso do poder "raio" presente no jogo, é necessário destruir todos os obstáculos na tela no momento em que o jogador entra em contato com o raio, tarefa de fácil realização ao utilizar a função *forEachAlive*.

Ao fazer com que todas as funções trabalhassem em conjunto, foi criada a jogabilidade principal para o projeto prático. A programação destas funções e lógica do jogo fez com que o resultado esperado fosse atingido. O comportamento do jogo nas diferentes plataformas disponíveis será apresentado no capítulo seguinte.

4 DESEMPENHO MULTIPLATAFORMA

Um jogo multiplataforma deve ser executado com qualidade em todas as plataformas que será lançado. Por esse motivo, o jogo deve ser testado antes no maior número de dispositivos possíveis para identificar possíveis problemas de execução em plataformas especificas e corrigi-los antes de seu lançamento. Esta verificação pode ser feita através de simuladores de ambientes.

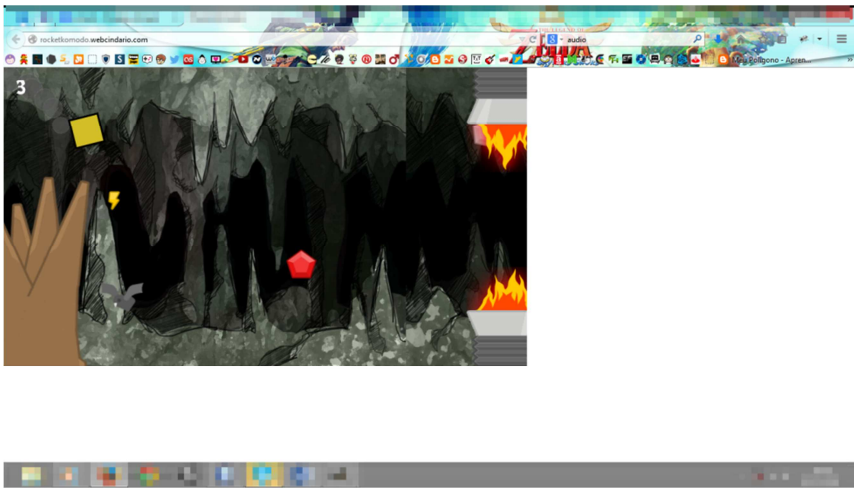
Para o desempenho do jogo ser considerado "bom" na plataforma analisada, ele precisa cumprir dois requerimentos: possuir nenhuma ou pouca queda de FPS durante sua execução e apresentar nitidamente os elementos do jogo para o jogador. No caso de celulares, principalmente, é possível observar que muitos não conseguem atingir esses requerimentos por possuir um *hardware* não muito robusto e/ou possuir uma tela muito pequena com pouca resolução.

Durante os testes de desempenho foram avaliados três tipos de dispositivos: computadores, *tablets* e *smartphones*. Nos computadores foi avaliado apenas o desempenho através do navegador, enquanto nos *smartphones* e *tablets* foi analisado o desempenho tanto através do navegador quanto com o jogo instalado como aplicativo nativo.

O primeiro dispositivo testado foi um computador *Windows* 8 64 bits, com processador *Core* i3-2330M de 2.20 GHZ e memória RAM de 6GB. O teste foi realizado utilizando três navegadores diferentes: *Mozilla Firefox, Google Chrome e Internet Explorer*. Os três executaram o jogo com pouquissimas quedas de FPS e reconheceram o *input* do jogador sem problemas, porém o Internet Explorer foi o único navegador que não reconheceu o áudio do jogo.

A resolução do jogo ficou no tamanho original devido ao tamanho da monitor do computador, evitando distorções na imagem. Devido a isso, todos os elementos do jogo foram visualizados sem problemas pelos jogadores. Uma simulação de como o jogo ficou no *notebook* pode ser visualizada na figura 25.

Figura 24 - Jogo no Firefox



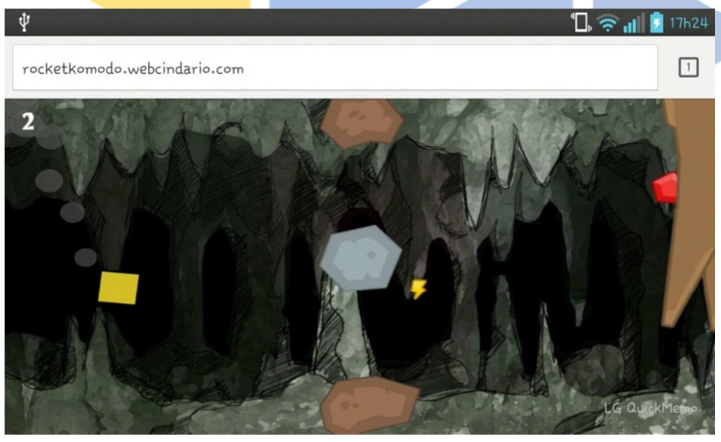
O segundo dispositivo testado foi um *smartphone* não tão robusto lançado em 2012: *Samsung Galaxy Y Duos*. O aparelho conta com o sistema operacional *Android* 2.3 e um processador de apenas um núcleo de 832 MHZ de potência. Sua tela é de 3,14 polegadas com 320x240 de resolução.

Nesse dispositivo o jogo encontrou diversos problemas para ser executado. No navegador nativo do aparelho, o jogo carregou bem lentamente e depois não aceitou nenhum *input* depois de carregado. Além disso, o som não foi reconhecido pelo navegador. Como um aplicativo, o comportamento do jogo foi um pouco diferente: ele abriu e o som foi reconhecido, porém devido ao *hardware* antigo e pouca resolução, a organização dos elementos na tela ficou distorcida e ele não reconheceu nenhum *input*.

É importante ressaltar que o segundo aparelho testado é um aparelho antigo e que não possui um bom desempenho com diversos aplicativos modernos. Já o terceiro dispositivo analisado não apresentou muitos problemas. O dispositivo em questão é o *LG Optimus G*, que possui o sistema operacional *Android* 4.0 e processador *Quad-Core* de 1.5 GHZ de potência, além de uma tela de 4.7 polegadas com 1280x768 de resolução.

Através do *Google Chrome,* o navegador não apresentou nenhum problema ao executar o jogo. O som funcionou muito bem, os *inputs* foram reconhecidos sem problemas, a taxa de FPS ficou bem estável durante toda a jogatina e a resolução do jogo, que é menor do que a resolução do aparelho, mesmo com todos os elementos "esticados" ficou muito semelhante à versão dos navegadores *desktop*, conforme apresentado na figura 26.

Figura 25 - *Printscreen* do jogo sendo executado no LG Optimus G



Como um aplicativo nativo, o jogo não possuiu problemas durante sua execução. Assim como na versão W*eb*, todos os elementos foram reconhecidos e funcionaram de acordo com o esperado. A taxa de FPS foi ainda melhor do que quando executado através do *Google Chrome*.

Finalizando os testes, foram realizados testes com um *tablet Philco* com *Android* 4.0 e processador *ARM Cortex* de 1.0 GHZ de potência, com tela de 7 polegadas e 800x480 de resolução.

No navegador nativo do aparelho o jogo ficou extremamente lento, ocorreram diversas quedas de FPS, se tornando impossível de jogar. O som também não foi reconhecido e os *inputs* eram reconhecidos com lentidão pelo aparelho. Por outro lado, a resolução apresentou bom desempenho principalmente por se aproximar à resolução original do jogo. Como um aplicativo instalado, o desempenho do jogo melhorou bastante, sem quedas constantes de FPS e com o som funcionando como esperado.

Como pode ser observado, nenhum aparelho de última geração foi utilizado para realizar os testes e a maioria dos aparelhos conseguiu executar o jogo conforme esperado, tornando-o um jogo que terá um alcance muito maior do que um jogo "pesado". Com um polimento maior do jogo é possível obter um desempenho ainda melhor, garantindo uma gama maior de dispositivos para lançar o jogo.

É possível observar de forma resumida o desempenho nos diversos navegadores testados através da tabela 2:

Tabela 2 - Desempenho multiplataforma nos navegadores

Dispositivo	Desempenho	Quedas de FPS	Som	Resolução
Internet Explorer (Notebook)	Satisfatório	Poucas quedas	Não funcionou	Original (840x480)
Mozilla Firefox (Notebook)	Satisfatório	Poucas quedas	Funcionou	Original (840x480)
Google Chrome (Notebook)	Satisfatório	Nenhuma queda	Funcionou	Original (840x480)
Navegador Android Nativo (Samsung Galaxy Y Duos)	Não satisfatório	Jogo não executou	Não funcionou	Elementos distorcidos (320x240)
Google Chrome (LG Optimus G)	Satisfatório	Poucas quedas	Funcionou	Elementos semelhantes à versão original (1280x768)
Navegador Android Nativo (Tablet Philco)	Não satisfatório	Muitas quedas	Não funcionou	Bem semelhante à versão original (800x480)

Fonte: Próprio autor

Na tabela 3, é possível observar de forma resumida, como o jogo se comportou como um aplicativo nativo nos diversos aparelhos:

Tabela 3 - Desempenho multiplataforma como aplicativo nativo

Dispositivo	Desempenho	Quedas de FPS	Som	Resolução
Smartphone - Samsung Galaxy Y Duos	Não satisfatório	Jogo não executou	Funcionou	Elementos distorcidos (320x240)
Smartphone - LG Optimus G	Satisfatório	Nenhuma queda	Funcionou	Elementos semelhantes à versão original (1280x768)
Tablet – Philco	Satisfatório	Poucas quedas	Funcionou	Bem semelhante à versão original (800x480)

Fonte: Próprio autor

5 CONCLUSÃO

Ao realizar essa pesquisa foi possível identificar o quanto as tecnologias *Web* evoluíram para permitir a criação de um conteúdo multiplataforma de forma mais fácil e acessível para os desenvolvedores. A independência de *plug-ins* externos possibilitou uma grande expansão ao HTML5, que devido a esse fato, consegue ser executado em dispositivos móveis sem problemas.

Conforme apresentado durante a pesquisa, o desenvolvimento de um jogo multiplataforma requer muito mais do que apenas o seu desenvolvimento e, para isso, a *Phaser Game Engine* conta com diversas funcionalidades que auxiliam o desenvolvedor a passar por todas as etapas necessárias com facilidade, tornando-a uma das melhores opções para o desenvolvimento de jogos multiplataforma na W*eb* da atualidade.

Com o cenário de jogos atualmente evoluindo constantemente é crítico que a *engine*, em um futuro próximo, suporte gráficos 3D para se manter como uma das melhores opções para se desenvolver jogos na W*eb.* Enquanto isso, sua simplicidade no uso e grande gama de possibilidades (comprovados durante a execução do projeto prático), permitindo a criação de diversos tipos de jogos, ainda a mantém nesse posto.

De acordo com a pesquisa realizada, é possível concluir que a *Phaser* não é apenas uma boa ferramenta para o desenvolvimento de jogos multiplataforma, mas também uma ferramenta muito útil durante a criação de protótipos e realização de testes de um projeto seja ele multiplataforma ou não.

Como trabalhos futuros, destaca-se a necessidade de pesquisas sobre a monetização de jogos com a *Phaser*, além de pesquisas sobre a evolução dos navegadores de dispositivos móveis em relação à grande quantidade de conteúdo multimídia disponível na W*eb* nos dias de hoje.

REFERÊNCIAS

ADOBE. **Flash to focus on PC browsing and mobile apps**: Adobe to more aggressively contribute to HTML5. Disponível em: <http://blogs.adobe.com/flashplayer/2011/11/flash-to-focus-on-pc-browsing-and-mobile-apps-adobe-to-more-aggressively-contribute-to-html5.html>. Acesso em 01 Junho 2014.

AVOLA, G. ; RAASCH, Jon. **Smashing mobile Web development.** Hoboken: John Wiley & Sons, 2012.

BUSKIRK, R. Van ; MORONEY, B. W. (2003). **Extending prototyping**. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5386825>. Acesso em 25 de Agosto de 2014.

DAVEY, Richard. **How to learn the Phaser HTML5 game engine**. Disponível em: <http://gamedevelopment.tutsplus.com/articles/how-to-learn-the-phaser-html5-game-engine--gamedev-13643>. Acesso em 01 de Junho de 2014.

EBERLY, David H. **Game physics**. 2.ed. New York: Elsevier, 2010.

FLANAGAN, David. **JavaScript**: the definitive guide. Sebastopol: O'Reilly Media, 2006.

FILHO, Marisardo Medeiros ; BENICIO, Ian V. ; CAMPOS, Fábio, NEVES, André M. M. **A importância da prototipação no *design* de games**. Disponível em: <http://www.sbgames.org/sbgames2013/proceedings/arte*design*/37-dt-paper.pdf>. Acesso em 26 de agosto de 2014.

FONSECA, João José Saraiva da. **Metodologia da pesquisa científica**. Fortaleza: UEC, 2002. Apostila.

HOM, James. **The usability methods toolbox handbook** (1998). Disponível em: <http://www.idemployee.id.tue.nl/g.w.m.rauterberg/>. Acesso em 25 de Agosto de 2014.

IDC BRASIL. **Estudo da IDC aponta recorde de vendas de smartphones no Brasil no segundo trimestre de 2014**. Disponível em: <http://br.idclatin.com/releases/news.aspx?id=1713>. Acesso em 09 de Novembro de 2014.

MEYER, Jeanine. **The essential guide to HTML5**: using games to learn HTML5 and JavaScript. London: Apress - ex-Peer Information, 2010.

MORRISON, Michael. **Beginning mobile phone game programming**. New York: Sams, 2004.

MICROSOFT. **Microsoft Internet Explorer 3.0:** beta now available. Disponível em: <http://www.microsoft.com/en-us/news/press/1996/may96/ie3btapr.aspx>. Acesso em 01 Junho 2014.

MOZILLA. **Plugins**: para que servem e como instalar? Disponível em: <http://br.mozdev.org/firefox/plugin>. Acesso em 26 de Outubro de 2014.

NINTENDO. **Nintendo demonstrates indie game support at game developers** Conference. Disponível em: <http://press.nintendo.com/articles.jsp?id=41311>. Acesso em 01 Junho 2014.

NOVAK, Jeannie. **Game development essencials**: an introduction. 3.ed. New York:Cengage Learning, 2011.

PILGRIM, Mark. **HTML5**: up and running. Sebastopol: O'Reilly Media, 2010.

ROVIO. **Angry Birds**. Disponível em: <http://www.rovio.com/en/our-work/games/view/1/angry-birds>. Acesso em 30 de Agosto de 2014.

ROCKSTAR. **Grand theft auto.** San Andreas: Rockstar, 2013. Mídia digital.

SCHELL, J. **The art of game d*esign***: a book of lenses. Boca Raton: CRC, 2008.

THORN, Alan. **Game engine d*esign* and implementation**. Burlington: Jones & Bartlett Learning, 2011.

WARFEL, Todd Zaki. **Prototyping**: a practitioner's guide. New York: Rosenfeld Media, 2009.

W3C. **HTML5 Introduction**. Disponível em: <http://www.w3schools.com/html/html5_intro.asp>. Acesso em 01 Junho 2014.

W3C. **Plan 2014**. Disponível em: <http://dev.w3.org/html5/decision-policy/html5-2014-plan.html>. Acesso em 01 Junho 2014.

YACOUB, Sherif ; AMMAR, Hany. **Pattern-Oriented analysis and de*sign***: composing patterns to d*esign* software systems. London: Pearson Education, 2004.

Tiago Zanchi de Paula

Atualmente é estudante no curso de Jogos Digitais na Faculdade de Tecnologia de Americana. Graduou-se em Técnico em Informática (ênfase em programação) em 2011 pelo Colégio Network. Possui experiência profissional desde 2012 com desenvolvimento e manutenção de sistemas web e manutenção de banco de dados

Contato: tiagozdp@hotmail.com

Fonte: CNPQ – Curriculo Lates

Aline Bossi Pereira da Silva

É aluna do programa de Mestrado em Tecnologia e Inovação (Faculdade de Tecnologia - Unicamp) .Especializou-se em Docência no Ensino Superior (Universidade Metodista de Piracicaba). Graduou-se em Análise de Sistemas e Tecnologia da Informação (ênfase em Jogos Digitais) pela FATEC - Faculdade de Tecnologia de Americana - com conclusão em 2011. Possui experiência profissional desde 2008 com desenvolvimento de sistemas web e websites junto a testes de usabilidade e acessibilidade de software em ambiente web. Possui experiência em docência no ensino superior desde 2014.

Contato: alinebossi@hotmail.com

Fonte: CNPQ – Curriculo Lates