**CENTRO PAULA SOUZA**

**FACULDADE DE TECNOLOGIA DE MOGI-MIRIM**

Diego Victor Gonzaga

Kauê Rocha

Tomas Coradi Lino

Vinícius de Moraes da Silva Pinto

**TO INFINTY AND BEYOND**

JOGO PARA A PLATAFORMA *WEB*, DESENVOLVIDO COM *HTML*5 *CANVAS* E *JAVASCRIPT*

**MOGI-MIRIM-SP**

**2018**

**Diego Victor Gonzaga**

**Kauê Rocha**

**Tomas Coradi Lino**

**Vinícius de Moraes da Silva Pinto**

**TO INFINTY AND BEYOND**

JOGO PARA A PLATAFORMA *WEB*, DESENVOLVIDO COM *HTML*5 *CANVAS* E *JAVASCRIPT*

Trabalho de Graduação apresentado ao Curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Faculdade de Tecnologia de Mogi Mirim como pré-requisito para a obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

**Orientador: Thales de Tarsis Cezare**

**MOGI MIRIM**

**2018**

Diego Victor Gonzaga

Kauê Rocha

Tomas Coradi Lino

Vinícius de Moraes da Silva Pinto

**TO INFINTY AND BEYOND**

JOGO PARA A PLATAFORMA *WEB*, DESENVOLVIDO COM *HTML*5 *CANVAS* E *JAVASCRIPT*

Este Trabalho de Graduação foi julgado e aprovado para obtenção do título de Tecnólogo, no Curso de Análise e Desenvolvimento de Sistemas, da Faculdade de Tecnologia de Mogi-Mirim.

Mogi-Mirim, 05 de julho de 2018.

**Prof. Thales de Tarsis Cezare**

**BANCA EXAMINADORA**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Professor Orientador

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Professor

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Professor

RESUMO

Os jogos digitais estão cada vez melhores, mas para que ocorram melhorias, as tecnologias disponíveis precisam evoluir ou novas devem surgir. Recentemente a tecnologia *HTML*5 *Canvas* surgiu para trazer algumas facilidades e novas funcionalidades aos navegadores mais modernos o que possibilitou a criação de novos jogos com gráficos muito melhores, mais leves e menos complexos, tudo isso sem a necessidade de instalar ou utilizar programas terceiros.

Ao decorrer deste trabalho serão apresentados discussões e resultados de todas as fases do processo de desenvolvimento de um jogo, intitulado de *To Infinity and Beyond*, desde a fase de entrevista e planejamento até a fase de programação e implementação, dando ênfase em algumas etapas, como a de desenvolvimento da interface e do motor, ao final estarão anexados alguns documentos técnicos produzidos durante essas etapas.

Como decorrência de todo esse trabalho surgiu o resultado final, um jogo no estilo do antigo *Space Invaders*, porém com um visual melhorado e moderno, e jogabilidade mais interativa e interessante. O grande diferencial deste trabalho foi utilizar as novas tecnologias disponibilizadas pela Web moderna, a exemplo, a tecnologia *HTML5 Canvas* já citada.

**Palavras-chaves:** Digital; *JavaScript*; Jogo; Web

*ABSTRACT*

*Digital games are getting better, but for improvements to take place, the available technologies need to evolve or new ones must emerge. Recently HTML5 Canvas technology came to bring some features and new features to the most modern browsers which made it possible to create new games with much better graphics, lighter and less complex, without the need to install or use third-party programs.*

*During the course of this work, we will present discussions and results of all phases of the development process of a game called To Infinity and Beyond, from the interview and planning phase to the programming and implementation phase, emphasizing some steps, such as the development of the interface and the engine, at the end will be attached some technical documents produced during these stages.*

*As a result of all this work came the final result, a game in the style of the old Space Invaders, but with an improved and modern look and more interactive and interesting gameplay. The great difference of this work was to use the new technologies made available by the modern Web, for example, the HTML5 Canvas technology already mentioned.*

***Keywords:*** *Digital; Game; JavaScript; Web*

INDICE DE FIGURAS

[**Figura 2.1** – Console *Mattel Intellivision* 10](#_Toc520148897)

[**Figura 2.2** – Jogos mais baixados 2014 11](#_Toc520148898)

[**Figura 2.3** – *Wolfenstein* 3D (1992) 12](#_Toc520148899)

[**Figura 2.4** – Camadas da Engenharia de *Software* 14](#_Toc520148900)

[**Figura 2.5** – Fases do método em cascata 16](#_Toc520148901)

[**Figura 2.6** – O processo da Extreme *Programming* (XP) 19](#_Toc520148902)

[**Figura 2.7** – Exemplo de caso de uso 25](#_Toc520148903)

[**Figura 2.8** – Exemplo de Diagrama de Classe 26](#_Toc520148904)

[**Figura 2.9** – Exemplo de Diagrama de Atividades 27](#_Toc520148905)

[**Figura 2.10** – Função de um programa 30](#_Toc520148906)

[**Figura 2.11** – Exemplo de documento *HTML* 32](#_Toc520148907)

[**Figura 2.12** – Exemplo de código *Canvas* e sua renderização 33](#_Toc520148908)

[**Figura 2.13** – Exemplo de código *JavaScript* 36](#_Toc520148909)

[**Figura 2.14** – Jogo criado com Cocos2d-JS 37](#_Toc520148910)

[**Figura 2.15** – Jogo desenvolvido com Stage.js 38](#_Toc520148911)

[**Figura 4.1** – Protótipo de baixa fidelidade da tela de continuação 44](#_Toc520148912)

[**Figura 4.2** – Protótipo de baixa fidelidade da tela de fim de jogo 45](#_Toc520148913)

[**Figura 4.3** – Protótipo de baixa fidelidade das telas de pausa, inicial e carregamento 46](#_Toc520148914)

[**Figura 4.4** – Protótipos de alta fidelidade das telas de pausa, continuação, fim de jogo e carregamento 46](#_Toc520148915)

[**Figura 4.5** – Protótipo de alta fidelidade da tela inicial 47](#_Toc520148916)

[**Figura 4.6** – Tela de carregamento 49](#_Toc520148917)

[**Figura 4.7** – Menu de fim de jogo 51](#_Toc520148918)

[**Figura 4.8** – Tela de pausa 52](#_Toc520148919)

[**Figura 4.9** – Personagem principal do jogador 52](#_Toc520148920)

[**Figura 4.10** – Limite da tela em destaque 53](#_Toc520148921)

[**Figura 4.11** – Interface de usuário destacada 54](#_Toc520148922)

[**Figura 4.12** – Inimigos 55](#_Toc520148923)

[**Figura 4.13** – Itens lançados pelos inimigos 56](#_Toc520148924)

[**Figura 4.14** – A área preenchida exemplifica a intersecção 57](#_Toc520148925)

[**Figura 4.15** – Colisão 57](#_Toc520148926)

[**Figura 4.16** – Explosão em destaque 58](#_Toc520148927)

[**Figura 4.17** – Itens 59](#_Toc520148928)

[**Figura 4.18** – *Spritesheet* do jogo 60](#_Toc520148929)

[**Figura 4.19** – Pedaços da *spritesheet* em destaque 61](#_Toc520148930)

[**Figura 4.20** – Tela inicial sem seletor de opção de menu 62](#_Toc520148931)

INDICE DE ABREVIATURAS E SIGLAS

2D – Duas Dimensões

3D – Três Dimensões

API – Application Programming Interface

DSDM – Desenvolvimento de Sistemas Dinâmicos

GPS – Global Positioning System

HTML – HyperText Markup Language

HTML5 – HyperText Markup Language 5

HUD – Heads-up Display

IBGE – Instituto Brasileiro de Geografia e Estatística

ID – Identificador

IEEE – Instituto de Engenheiros Eletricistas e Eletrônicos

iOS – iPhone Operating System

JS – JavaScript

MDN – Mozilla Developer Network

OS – Operating System

POO – Programação Orientada a Objetos

RPG – Role-Playing Game

TV – Televisão

UFO – Unidentified Flying Object

UFSC – Universidade Federal de Santa Catarina

UML – Unified Modeling Language

XP – Extreme Programming

SUMÁRIO

[RESUMO III](#_Toc520749048)

[*ABSTRACT* IV](#_Toc520749049)

[INDICE DE FIGURAS V](#_Toc520749050)

[INDICE DE ABREVIATURAS E SIGLAS VI](#_Toc520749051)

[1. INTRODUÇÃO 9](#_Toc520749053)

[2. REVISÃO BIBLIOGRAFICA 11](#_Toc520749054)

[2.1. JOGOS E SUAS PLATAFORMAS 11](#_Toc520749057)

[2.1.1. CONSOLE 11](#_Toc520749058)

[2.1.2. DISPOSITIVOS MOVEIS 12](#_Toc520749059)

[2.1.3. COMPUTADORES 13](#_Toc520749060)

[2.1.4. WEB 14](#_Toc520749061)

[2.2. ENGENHARIA DE *SOFWARE* 15](#_Toc520749062)

[2.2.1. METODOLOGIAS TRADICIONAIS 17](#_Toc520749063)

[2.2.2. METODOLOGIAS ÁGEIS 19](#_Toc520749064)

[2.2.3. LEVANTAMENTO DE REQUISITOS 23](#_Toc520749065)

[2.2.4. UML 26](#_Toc520749066)

[2.3. PROJETO DE INTERFACE 29](#_Toc520749067)

[2.4. DESENVOLVIMENTO DE *SOFTWARE* 31](#_Toc520749068)

[2.4.1. PROGRAMAÇÃO ORIENTADA A OBJETOS (POO) 32](#_Toc520749069)

[2.4.2. PROGRAMAÇÃO BASEADA EM PROTÓTIPOS 33](#_Toc520749070)

[2.4.3. INTERATIVIDADE NA *WEB* 33](#_Toc520749071)

[2.4.4. TECNOLOGIAS 33](#_Toc520749072)

[2.4.5. ANATOMIA DE UM VIDEOGAME 35](#_Toc520749073)

[2.4.6. TÉCNICAS 36](#_Toc520749074)

[2.4.7. FERRAMENTAS DE DESENVOLVIMENTO 37](#_Toc520749075)

[3. METODOLOGIA 42](#_Toc520749076)

[3.1. INSTRUMENTO DE COLETA DE DADOS 42](#_Toc520749078)

[3.2. CRONOGRAMA 42](#_Toc520749079)

[3.3. ATIVIDADES 43](#_Toc520749080)

[4. DESENVOLVIMENTO 45](#_Toc520749081)

[4.1. ENTREVISTA 45](#_Toc520749083)

[4.2. LEVANTAMENTO DE REQUISITOS 45](#_Toc520749084)

[4.3. DESENVOLVIMENTO DA INTERFACE 45](#_Toc520749085)

[4.4. DESENVOLVIMENTO DO MOTOR DA APLICAÇÃO 49](#_Toc520749086)

[4.4.1. FRAMES 50](#_Toc520749087)

[4.4.2. RECURSOS EXTERNOS 50](#_Toc520749088)

[4.4.3. CONTROLES 52](#_Toc520749089)

[4.4.4. MENUS 52](#_Toc520749090)

[4.4.5. NAVE 54](#_Toc520749091)

[4.4.6. INTERFACE DE USUÁRIO 56](#_Toc520749092)

[4.4.7. INIMIGOS 57](#_Toc520749093)

[4.4.8. COLISÕES E EXPLOSÕES 59](#_Toc520749094)

[4.4.9. ITENS 61](#_Toc520749095)

[4.5. IMPLEMENTAÇÃO DAS TELAS 62](#_Toc520749096)

[5. IMPLEMENTAÇÕES FUTURAS 65](#_Toc520749097)

[6. RESULTADOS E DISCUSSÕES 67](#_Toc520749098)

[7. BIBLIOGRAFIA 69](#_Toc520749099)

[8. APÊNDICE A – Fichas de Requisito 73](#_Toc520749100)

[9. APÊNDICE B – ATA DAS REUNIÕES 101](#_Toc520749101)

[10. APÊNDICE C – CÓDIGO FONTE 107](#_Toc520749102)

[11. Apêndice D – DIAGRAMAS DE MODELAGEM 142](#_Toc520749103)

[12. APÊNDICE E – ENTREVISTA 165](#_Toc520749104)

# INTRODUÇÃO

Segundo notícia do G1 (G1 RIO, 2015), o Brasil é o quarto maior consumidor de jogos digitais do mundo, os empregos no setor cresceram 13 vezes mais que o mercado de trabalho em geral, mas vem sofrendo com a falta de mão de obra qualificada. A consequência disso é o crescimento das ferramentas e tecnologias de apoio para esta atividade, que estão se tornando mais poderosas e eficientes, e cada vez mais simples, ao ponto de não ser necessário ter conhecimento de nenhuma linguagem de programação, tornando tudo muito intuitivo. Alguns exemplos abordados ainda neste trabalho são, o *Cocos2d-JS*, o *Stage.js* e o *Isogenic Engine*.

Com o crescimento constante da tecnologia, hoje é possível através de um celular, ter acesso à Internet em qualquer lugar e a qualquer hora. Tal facilidade implica diretamente no mercado de jogos digitais, pois a alta taxa de pessoas com acesso à rede mundial de computadores, que segundo o IBGE (G1 RIO, 2016) é de mais de 50% dos domicílios brasileiros desde 2014, gerou um maior número de interessados neste segmento, tornando-o cada vez mais forte e atraindo mais empresas do ramo a investir no Brasil, o que possibilita um crescimento muito grande para aqueles que trabalham neste mercado.

As aplicações construídas para a plataforma *web* estão em maior evidência, prova disso é o fato de a rede pública de computadores tem milhares de acessos diariamente, exemplo disso é a informação compartilhada pelo Olhar Digital (REDAÇÃO OLHAR DIGITAL, 2016), empresa jornalística do ramo de informática, que destaca o site *Giphy* por ter alcançado o incrível número 100 milhões de usuários diários.

Há constantemente novas tecnologias surgindo para auxiliar no desenvolvimento dos jogos digitais, tecnologias essas que são emergentes, como por exemplo, o *HTML5* e o *Canvas* que são bem recentes, mas já são muito empregadas nos jogos, como, o *Sand Trap*[[1]](#footnote-1) e *CUBE[[2]](#footnote-2)*.

Uma vez que o acesso à Internet está cada vez mais fácil, a demanda de jogos para esta plataforma aumenta consequentemente, dados que comprovam isso são os apresentados pelo Estadão (RONCOLATO, 2011), que previa que o mercado de games sócias, jogos de *Facebook*, por exemplo, até 2015 ia quintuplicar chegando a impressionante marca de cinco bilhões de dólares. E com esse grande crescimento uma nova necessidade surge, que é a agilização do processo de construção de aplicações para esse mercado, para tentar suprir essa grande demanda procura-se ferramental especializado, que poderiam ser usadas para potencializar, facilitar e acelerar o desenvolvimento de aplicações neste segmento. Por ser um mercado novo, será que as ferramentas de modelagem de *software* convencionais estão preparadas? Este tipo de aplicação não exige um tipo de ferramenta mais especializada?

Levando em conta esses pontos, neste mundo completamente novo, tanto para os desenvolvedores quanto para as aplicações de apoio ao desenvolvimento, a viabilidade da aplicação se torna possível?

Este trabalho de graduação tem como objetivo, mostrar que mesmo com pouco ferramental especializado para o desenvolvimento neste ambiente, já é possível criar um jogo com recursos de qualidade, com boa usabilidade e capaz de concorrer com outros jogos da mesma área. Propõe-se um jogo simples o suficiente para funcionar até mesmo em um computador com baixo poder computacional, que pode atingir pessoas de todas as faixas de etárias. A ideia principal do jogo é propor partidas rápidas, e que os jogadores em um intervalo de tempo curto consigam se divertir sem compromisso.

# REVISÃO BIBLIOGRAFICA

Neste capitulo será apresentado todo o material teórico em que o trabalho foi embasado.



## JOGOS E SUAS PLATAFORMAS

Segundo Arruda (2014, p. 3):

A concepção de jogos digitais envolve um entendimento mais amplo do que apenas saber o que são videogames. O termo videogame historicamente esteve limitado aos jogos de console e às máquinas de fliperama. Já as tecnologias digitais são baseadas na microinformática, o que engloba jogos para computadores, consoles, fliperamas, smart*Phone*s, *tablets* e qualquer equipamento que venha a existir.

Com o passar dos anos, cada vez mais os jogos vêm ganhando espaço, em 2013 no Brasil, conforme apresentado por Arruda (2014, p. 4), uma pesquisa da *Newzoo* revelou que mais de 35 milhões de brasileiros se entretêm com jogos digitais, a pesquisa não aborda outras plataformas, como celulares e tablets, mas apresenta uma pequena ideia da grandeza que este mercado possui.

As diversas plataformas apresentam suas peculiaridades, porém de acordo com o ponto de vista vantagens podem virar desvantagens e vice-versa, afinal a escolha da plataforma tem muita influência dos fatores que o cada um leva em consideração, como, conforto, qualidade dos gráficos, desempenho, portabilidade, custos, etc. A seguir uma breve contextualização de cada uma delas.

### CONSOLE

De acordo com o artigo publicado por Gonzales Clua e Bittencourt (2005, p. 1319) os consoles ou videogames são jogos eletrônicos desenvolvidos para uso doméstico, que basicamente funcionam acoplados a aparelhos de televisão.

Em outro artigo, Machado Pinto, Coronel e Bresolin (2013, p. 46), complementam que um console tem por objetivo “executar *software* de jogos de vídeo e é caracterizado como um computador otimizado nos seus componentes internos e externos”, ou seja, foram projetados com *hardware* dedicado para executar jogos, de diversas naturezas.

Segundo Gonzales Clua e Bittencourt (2005, p. 1319-1327) os primeiros consoles executavam jogos muito simples, monocromáticos, com simples formas geométricas e sem som, mas isso mudou no decorrer dos anos, pois avanços de *hardware* permitiram aos consoles evoluírem a ponto de hoje executarem jogos 3D coloridos, multijogadores, com efeitos sonoros, músicas, acesso à Internet, e muitas outras novas funcionalidades.

**Figura 2.1** – Console *Mattel Intellivision*



**Fonte:** GONZALES CLUA e BITTENCOURT (2005, p. 1320)

O maior benefício apresentado pelos consoles é seu alto desempenho, uma vez que todo seu *hardware* é dedicado para jogos seu comportamento realizando esta atividade é superior ao das demais plataformas, entretanto, como todo o seu poder de processamento está direcionado para esta tarefa os consoles acabam por ter um conjunto limitado de funcionalidades se comparado a um computador comum. E em geral são relativamente caros se levado em conta esse limitado número de tarefas que podem realizar.

### DISPOSITIVOS MOVEIS

Segundo o artigo de Lemes, Tomaselli e Camarotti (2012, p. 2), publicado na *SBGames*, os dispositivos moveis são telefones celulares, tablets, e aparelhos pequenos e leves com capacidade de reproduzir jogos.

Em uma matéria da revista Veja (REDAÇÃO VEJA, 2016), a empresa de tecnologia Cisco projeta que o número de celulares no mudo chegará a 5,4 bilhões em 2020, e essa grande massa está a procura sempre de algo a mais, Lecheta (2013, p. 21) afirma, “usuários comuns estão procurando cada vez mais celulares com diversos recursos, como câmeras, músicas, *Bluetooth*, ótima interface visual, jogos, GPS, acesso à internet e e-mail”.

Os números apresentados pela Cisco (REDAÇÃO VEJA, 2016) deixam claro o poder de popularização desta plataforma, devido a esses números absurdos, e como citado no artigo da *SBGames* (LEMES, TOMASELLI e CAMAROTTI, 2012), os dispositivos moveis são uma plataforma portátil de grande alcance, mesmo em países de baixa renda, mostrando tanto potencial era de se esperar que os jogos também embarcassem nesse mercado.

Mas independente de todos esses fatores, o número de *downloads* de jogos para dispositivos moveis vêm crescendo cada vez mais, isso fica claro na matéria apresentada pelo G1 (GOMES, 2014), que lista os cinco jogos mais baixados no *Google Play*, loja de aplicativos do *Google*. Em primeiro e segundo lugar estão Pou e *Subway Surfers* respectivamente, mesmo sem números oficiais publicados é possível imaginar a popularidade atingida por esses aplicativos tendo em vista o grande sucesso do jogo Meu Tom Falante, terceiro colocado, que a *Outfit7*, desenvolvedora do jogo, afirma ter cerca de 230 milhões de *downloads*, seguido por Meu Malvado Favorito em quarto, e Perguntados em quinto.

**Figura 2.2** – Jogos mais baixados 2014



**Fonte:** (GOMES, 2014)

### COMPUTADORES

No passado, assim como outras plataformas, os computadores possuíam limitações de *hardware* que reduziam sua capacidade de processamento e consequentemente prejudicava a qualidade dos jogos. Com os avanços tecnológicos durante vários anos a eficiência dos componentes que constituíam os computadores se desenvolveu muito, o que implicou no surgimento de novas tecnologias relacionadas ao universo de desenvolvimento de jogos, como o 3D.

Gonzales Clua e Bittencourt (2005, p. 1326) em seu artigo destacam “Com as novas tecnologias de criação de universos tridimensionais e a redução dos preços de processadores mais potentes, os jogos para computador tornaram-se altamente imersivos” e hoje, existem até computadores com *hardware* dedicado para execução de jogos, construídos com componentes de alto desempenho e configurações específicas para tal atividade.

**Figura 2.3** – *Wolfenstein* 3D (1992)



**Fonte:** (GONZALES CLUA e BITTENCOURT, 2005, p. 1326)

### WEB

Antes de qualquer coisa é preciso lembrar que a Internet nem sempre foi o que ela é hoje. A sua primeira versão, a *Web* 1.0, segundo Silveira Ferreira (2010, p. 13, 19), os sites eram estáticos e possuíam baixa ou nenhuma interatividade, isto é, apresentavam muitas informações aos visitantes e estes não podiam modifica-las ou contribuir de alguma forma. Na *Web* 2.0 “ao contrário da *web* 1.0, uma informação pode ser atualizada por inúmeras pessoas, permitindo que os usuários sejam muito mais do que meros espectadores e façam parte do espetáculo” (SILVEIRA FERREIRA, 2010, p. 21).

A nova possibilidade de interação do usuário trouxe muitas possibilidades e inovações, alguns exemplos apresentados por Silveira Fereira (2010, p. 21) são, *Flickr*, *Facebook*, *Twitter* e *MySpace*, e, claro, os jogos podem entrar nessa lista.

A *Web* moderna se tornou r*API*damente uma plataforma viável não só para a criação de jogos de alta qualidade, mas, mais importante que isso, tornou-se um ótimo meio de distribuição desses jogos.

De acordo com uma página sobre desenvolvimento de jogos *Web* da *Mozilla* *Developer Network*, MDN (2017):

A gama de jogos que podem ser criados está a par com desktops e sistemas operacionais nativos. Com as tecnologias da *Web* moderna e um navegador recente, é perfeitamente possível fazer jogos impressionantes, de alto nível. Nada de jogos de cartas simples ou jogos sociais multijogadores de antigamente feitos usando *Flash*, mas sim jogos de ação em 3D, RPGs, e muito mais. Graças a grandes melhorias de desempenho na tecnologia do compilador *JavaScript*, e novas *APIs*, é possível construir jogos que rodam no navegador, ou em dispositivos com suporte a *HTML*5 como aqueles baseados no Firefox OS, sem fazer concessões.

As vantagens de desenvolver aplicações *web* são muitas, por exemplo, o alcance da Internet é enorme, está em toda parte, jogos construídos com *HTML*5 funcionam em smart*Phone*s, tablets, PCs e *Smart* TVs, o marketing e a popularização são maiores, a aplicação não fica limitada a ser publicada em somente uma loja de aplicativos, ao contrário, ela pode ser anunciada e publicada em toda a rede como também em outras formas de mídia, aproveitando a facilidade de compartilhamento que a Internet proporciona, e assim alcançando novos horizontes.

Uma notícia do G1 (2011) pode exemplificar o sucesso dos jogos desta plataforma, em 2011, o jogo *CityVille* tinha uma média de 75 milhões de usuários ativos por mês, enquanto o *The Sims* Social possuía uma média de 40 milhões, Empires & Allies e Zynga Poker da famosa empresa de jogos Zynga, contavam com 39 e 36 milhões respectivamente, todos estes jogos citados estão disponíveis no *Facebook*. Apenas estes quatro jogos somavam quase 200 milhões de usuários e isso no ano de 2011 e em 2013 o *Facebook* (G1, 2013) registrou mais de 250 milhões de usuários mensais jogando na rede, e com certeza o número continuará a crescer.

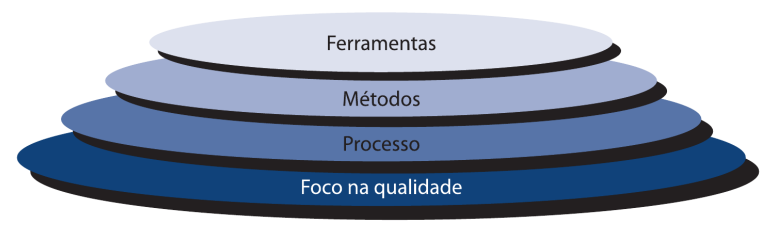
## ENGENHARIA DE *SOFWARE*

Pressman (2009, p. 39) e Hirama (2011, p. 7) lembram que Fritz Bauer propôs a definição, “Engenharia de *Software* é o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter *software* de maneira econômica, que seja confiável e funcione de forma eficiente em maquinas reais”, e logo Pressman (2009) apontou que essa definição “diz pouco sobre os aspectos técnicos da qualidade de *software*; ela não trata da necessidade de satisfação do cliente ou da entrega do produto dentro do prazo”, então para melhor definir a engenharia de *software* ele cita que a IEEE, Instituto de Engenheiros Eletricistas e Eletrônicos, desenvolveu uma definição mais abrangente ao afirmar que:

Engenharia de *software*: (1) A aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de *software*, isto é, a aplicação de engenharia ao *software*. (2) O estudo de abordagens como definido em (1)

Porém a abordagem varia muito de equipe para equipe, o importante é disciplina, mas sem perder muita adaptabilidade e agilidade. A Engenharia de *Software* é uma tecnologia dividida em camadas, conforme visto na figura 3.1.

**Figura 2.4** – Camadas da Engenharia de *Software*



**Fonte:** (PRESSMAN, 2009)

A base da engenharia de *software* é a camada de processos. O processo de engenharia de *software* é a liga que mantém as camadas de tecnologia coesas e possibilita o desenvolvimento de *software* de forma racional e dentro do prazo. (PRESSMAN, 2009, p. 40)

Hirama (2011, p. 9) diz que a:

Camada de métodos provê as abordagens e as atividades necessárias para a construção de um *software*. Os métodos abrangem um conjunto amplo de tarefas que incluem analise de requisitos, projeto, implementação, teste e manutenção.

Sobre a camada de ferramentas Pressman (2009) afirma que as ferramentas de engenharia de *software* devem fornecer suporte automatizado ou semi-automatizado para processos e métodos, de forma que quando integradas uma criei informação e que esta possa ser usada por outra.

Hirama (2011, p. 8) comenta que a “camada foco em qualidade, dá-se ênfase à preocupação de qualquer disciplina de engenharia que é a qualidade”.

Pode-se concluir então que a engenharia de *software* é uma ferramenta, uma técnica, que tem o objetivo de guiar, criar uma estrutura para o desenvolvimento de *software* de maneira a obter maior valor e qualidade possível ao processo e ao resultado final da atividade de desenvolvimento do produto.

### METODOLOGIAS TRADICIONAIS

As metodologias tradicionais são aquelas que foram as primeiras a serem pensadas para organizar os processos do desenvolvimento de *software* e foram muito utilizadas no passado, elas possuíam um custo muito alto de mudança, pois o acesso aos computadores era limitado.

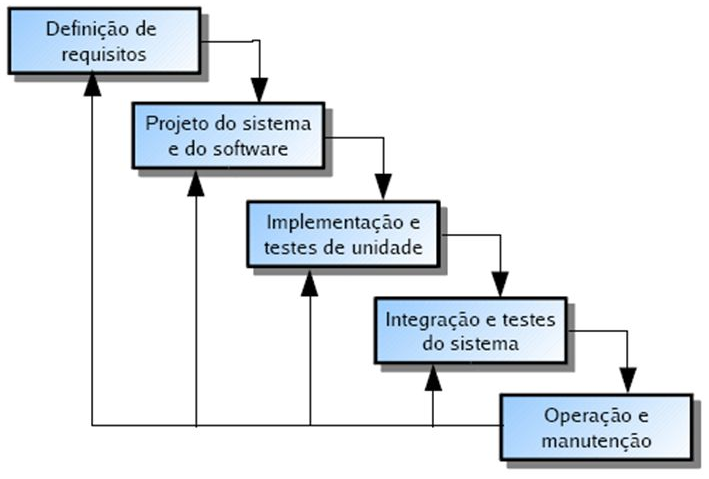
A história tem demonstrado que esses modelos tradicionais proporcionaram uma considerável contribuição quanto à estrutura utilizável no trabalho de engenharia de *software* e forneceram um roteiro razoavelmente eficaz para as equipes de *software*. Entretanto, o trabalho de engenharia de *software* e o seu produto permanecem “à beira do caos”. (PRESSMAN, 2009, p. 58, 59)

Sommerville (2003, p. 36) diz que “Esses modelos genéricos não são descrições definitivas de processos de *software*. Em vez disso, são abstrações úteis, que podem ser utilizadas para explicar diferentes abordagens do desenvolvimento de *software*”.

#### MÉTODO EM CASCATA

O modelo em cascata, conhecido também como ciclo de vida de *software* foi o primeiro modelo a ser desenvolvido e, basicamente, consistia em implementar engenharia em desenvolvimento de *software*, ou seja, desenvolver o *software* em fases, como mostrado na figura 2.5.

**Figura 2.5** – Fases do método em cascata



**Fonte:** (SOMMERVILLE, 2003)

Segundo Sommerville (2003, p. 38):

Devido aos custos de produção e aprovação de documento, as iterações são onerosas e envolvem um “retrabalho” significativo. Portanto, após um número pequeno de iterações, é normal suspender partes do desenvolvimento, como a especificação, e continuar com os estágios posteriores de desenvolvimento.

Posteriormente afirma:

Hoje em dia, o trabalho de *software* tem um ritmo acelerado e está sujeito a uma cadeia de mudanças intermináveis (em características, funções e conteúdo de informações). O modelo cascata é frequentemente inapropriado para tal trabalho. (SOMMERVILLE, 2003, p. 61)

#### MÉTODO DE DESENVOLVIMENTO EVOLUCIONÁRIO

O método de desenvolvimento evolucionário consiste em primeiramente, entregar um *software* básico ao usuário e, com base nas necessidades do cliente, ir evoluindo o *software*, tendo assim, várias versões.

Segundo Sommerville (2003, p. 39) “A abordagem evolucionária do desenvolvimento de *software*, muitas vezes, é mais eficaz do que a abordagem em cascata, no sentido de produzir sistemas que atendam às necessidades imediatas dos clientes”.

Esse método é muito interessante quando usado em sistemas pequenos ou até em sistemas médios, porém quando se pensa em implementar o método evolucionário em sistemas grandes, temos problemas, pois os sistemas são mal estruturados devido às mudanças e para definir o progresso no desenvolvimento de um *software* com este método é difícil.

Pressman (2009, p. 64) diz que “O segredo é definir as regras do jogo logo no início; isso significa que todos os envolvidos devem concordar que o protótipo é construído para servir como um mecanismo para a definição de requisitos”.

#### MÉTODO INCREMENTAL

No método incremental, após definir as funcionalidades do sistema, o cliente deve dizer quais funcionalidades são mais importantes ou menos importantes, já na parte do desenvolvimento, é desenvolvido e entregue primeiro as funcionalidades mais importantes e as outras são incrementadas depois, fazendo assim com que o cliente tenha acesso às funcionalidades que ele mais necessita em um tempo mais curto e após isso o sistema vai evoluindo e suprindo todas as necessidades.

Sommerville (2003, p. 44) diz que “Uma vez identificados os incrementos, os requisitos para as funções a serem entregues no primeiro incremento são definidos em detalhes, e esse incremento é desenvolvido [...]”.

O desenvolvimento incremental é particularmente útil nos casos em que não há pessoal disponível para uma completa implementação na época de vencimento do prazo estabelecido para o projeto. (PRESSMAN, 2009, p. 62)

### METODOLOGIAS ÁGEIS

Métodos ágeis são caracterizados por serem mais focados no trabalho cooperativo do que na documentação escrita em si, ou seja, são métodos mais práticos, reduzindo assim o tempo do desenvolvimento do *software*.

Segundo Pressman (2009, p. 82):

Na economia moderna é frequentemente difícil ou impossível prever como um sistema computacional (por exemplo, uma aplicação baseada na *Web*) irá evoluir com o tempo. As condições de mercado mudam r*API*damente, as necessidades dos usuários finais se alteram e novas ameaças competitivas emergem sem aviso. Em muitas situações, não se conseguirá definir completamente requisitos antes que se inicie o projeto. É preciso ser ágil o suficiente para dar uma resposta ao ambiente de fluido negócios.

Esses princípios foram reunidos por um grupo de metodologistas conhecidos em um documento chamado de **Manifesto Ágil**.

Neste documento, segundo Filho (2009, p. 102) são proclamados os seguintes valores:

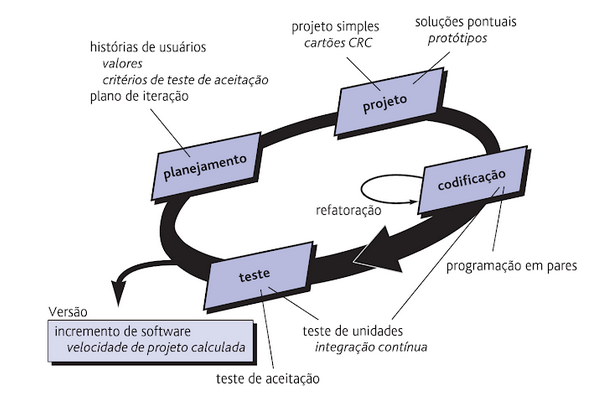
* Indivíduos e interações, em relação a processos e ferramentas.
* *Software* funcional, em relação a documentação abrangente
* Colaboração com o cliente, em relação a negociações contratuais.
* Resposta à mudança, em relação a seguir planos.

#### EXTREME PROGRAMMING (XP)

Segundo Pressman (2009, p. 88):

A Extreme *Programming* (programação extrema) emprega uma abordagem orientada a objetos (Apêndice 2) como seu paradigma de desenvolvimento preferido e envolve um conjunto de regras e práticas constantes no contexto de quatro atividades metodológicas: planejamento, projeto, codificação e testes.

**Figura 2.6** – O processo da Extreme *Programming* (XP)



**Fonte:** (PRESSMAN, 2009)

Filho (2009, p. 105) diz que “Provavelmente, o maior problema com muitas tentativas de aplicação do XP é o fato de que a metodologia não é aplicada de forma completa e consistente”.

O grupo utilizou uma forma modificada do método XP, pois com isso teríamos entregas constantes de partes operacionais, fazendo com que pudéssemos avaliar e ter um *feedback* de cada parte separadamente conforme o desenvolvimento, além do processo de desenvolvimento ser mais rápido.

#### SCRUM

Segundo Wazlawick (2013) o método *Scrum* segue também o Manifesto Ágil e tem algumas características estruturais, como: requisitos, análise, projeto, evolução e entrega. O método consiste em dividir o projeto em partes, chamadas *sprints* (o número de *sprints* em cada atividade varia dependendo de tamanho e complexidade) e todo dia ter as *daily scrums*, que são reuniões curtas para que toda a equipe esteja por dentro do que está sendo feito diariamente no projeto, na qual são feitas três perguntas-chave:

* O que você realizou desde a última reunião de equipe?
* Quais obstáculos estão encontrando?
* O que planeja realizar até a próxima reunião da equipe?

Segundo Pressman (2009, p. 96) “Um líder da equipe, chamado *Scrum* *master*, conduz a reunião e avalia as respostas de cada integrante. A reunião *Scrum*, realizada diariamente, ajuda a equipe a revelar problemas potenciais o mais cedo possível”.

Ao final do *sprint* deverá haver a *sprint review* meeting, para avaliar o produto do trabalho, e, eventualmente, a *sprint retrospective*, para avaliar os processos de trabalho. Assim, se aprovado, o produto (parcial ou final) poderá ser entregue ao cliente. (WAZLAWICK, 2013)

#### MÉTODO DE DESENVOLVIMENTO DE SISTEMAS DINÂMICOS (DSDM)

Wazlawick (2013) apresenta o método DSDM, consiste em trabalhar com a filosofia de que 80% de um *software* pode ser entregue em 20% do tempo que levaria para entregar 100% e utiliza prototipagem incremental, ou seja, faz-se vários protótipos, cada um sendo uma versão melhorada do anterior, esses 20% que faltam pode ser incrementado depois, quando os requisitos estiverem melhores definidos ou quando forem solicitadas alterações.

Segundo Pressman (2009, p. 97) O DSDM “[...] é uma abordagem para construir e manter sistemas que atendem restrições de prazo apertado através do uso da prototipagem incremental em um ambiente controlado”.

O DSDM Consortium é um grupo de empresas do mundo todo que mantém esse método e eles definiram um método ágil chamado ciclo de vida DSDM, onde há três ciclos iterativos: Iteração de modelos funcionais – produzir protótipos incrementais que demonstram funcionalidades – Iteração de projeto e desenvolvimento – revisita protótipos para validar a qualidade – Implementação – É quando se coloca a última versão do *software* no ambiente de trabalho (O incremento não pode estar completo e alterações ainda podem ser solicitadas na hora da implementação).

“O DSDM não é recomendável para projetos nos quais a segurança é um fator crítico, pois a necessidade de testes exaustivos desse tipo de sistema entra em conflito com os objetivos de custo e prazo do DSDM” (WAZLAWICK, 2013, p. 53)

### LEVANTAMENTO DE REQUISITOS

Dixon Moraes (2009) aborda o levantamento de requisitos da seguinte forma, um processo no desenvolvimento de *software* e a primeira atividade técnica. Uma das atividades mais importantes, é de vital importância para um bom entendimento sobre o *software* desejado pelo cliente. Ele deve definir os serviços e atividades que o sistema deve ou não realizar, como interface e restrições do sistema. O mapeamento dos processos ajuda o levantamento de requisitos para melhoria dos resultados, estabelecendo o que se deve fazer.

É metodologia de desenvolvimento e manutenção de sistemas modulares com as seguintes características: processo (roteiro) dinâmico, integrado e inteligente de soluções tecnológicas. (REZENDE, 2005, p. 90)

Conforme proposto por Somerville (2003, p. 83), um requisito é tratado como funcional quando descreve um serviço ou função que o sistema deve realizar. Paralelamente pode haver requisitos não-funcionais, que são restrições impostas, tanto ao sistema quanto ao desenvolvedor. Esse levantamento deve ser feito com o intuito de satisfazer as necessidades do cliente e funcionalidades propostas. Não se deve esquecer do especialista do domínio, profissional que trabalha na área em que o *software* será aplicado. Esse profissional possui experiência sobre os assuntos do ramo de mercado, ele não deve se apegar apenas a opinião da diretoria, mas também a dos usuários, mas é necessário tomar cuidado com a utilização dos profissionais. Sempre lembrando que essas práticas se bem-feitas, ajudam na garantia de qualidade se os processos não forem negligenciados.

#### ENTREVISTAS

A técnica da Entrevista também conhecida como Reunião, segundo Dixon Moraes (2009) é uma das mais conhecidas e mais simples de se utilizar no processo de levantamento de requisitos, justamente por ser fácil e simples de ser realizada essa técnica é comumente utilizada ao primeiro contato com o cliente e geralmente produz bons resultados nessa fase onde o principal objetivo é a obtenção de dados para assim entender as necessidades a serem vistas e desse modo trazer as possíveis soluções que se adequem bem ao desejo e satisfação do cliente. Antes de tudo é necessário que o entrevistador já tenha uma pauta para a sua entrevista, desse modo não indo totalmente a parte do assunto e dessa maneira não a entrevista tão longa e nem dispersa, podendo ser eficaz e com objetivo.

As seguintes diretrizes podem ser de grande auxilio na direção de entrevistas bem-sucedidas com o usuário: desenvolver um plano geral de entrevistas, certificar-se da autorização para falar com os usuários, planejar a entrevista para fazer uso eficiente do tempo, utilizar ferramentas automatizadas que sejam adequadas, tentar descobrir que informação o usuário está mais interessado e usar um estilo adequado ao entrevistar. (DIXON MORAES, 2009)

O planejamento de uma entrevista, segundo Dixon Moraes (2009) pode se dizer que é o pré-requisito da mesma, sendo assim nessa etapa do planejamento devem ser coletados e estudados todos os dados pertinentes a discussão dessa forma deixando os assuntos tratados na hora da execução mais contextualizados, fazendo assim a produtividade ser maior. Cabe ao analista ao fim da reunião realizar uma validação de todo aquele documento que foi feito, verificando se o conteúdo abordado vai atender as reais necessidades do cliente, se o cliente mudou de ideia em algum aspecto e se o mesmo está ciente e bem entendido sobre as questões abordadas e tratadas na entrevista.

Dixon Moraes (2009) afirma que o fator que decide o fracasso ou sucesso de uma entrevista é o comprometimento e maneira de como o entrevistador irá conduzi-la, é importante que o ao executar tenha a consciência de que o objetivo ali é ajudar o cliente e o atender da melhor maneira possível, soluções devem ser apresentadas e a opinião do entrevistado sempre deve ser escutada pois o perito no assunto é o próprio entrevistado e sua credibilidade jamais deve ser questionada, desse modo um bom entrevistador é aquele que saber usar as duas etapas da entrevista a de ouvir as necessidades e ideias e desse modo poder transmitir ao cliente algumas ideias e opiniões que agreguem e melhorem o todo.

Dixon Moraes (2009) concluí que a entrevista mesmo com o passar do tempo nunca perde seu valor e poder no quesito e sempre é bem-vinda, principalmente por estar sem dirigida diretamente com o cliente que é o responsável por todo o processo existir e desse modo poder satisfazê-lo da melhor maneira.

A atitude do analista em relação à entrevista é determinar seu fracasso ou sucesso. Uma entrevista não é uma competição, deve-se evitar o uso excessivo de termos técnicos e não conduzir a entrevista em uma tentativa de persuasão. O modo como o analista fala não deve ser muito alto, nem muito baixo, tampouco indiretamente, ou seja, utilizar os termos: ele disse isso ou aquilo na reunião para o outro entrevistado. (DIXON MORAES, 2009)

#### QUESTIONÁRIO

Dixon Moraes (2009) afirma, esta técnica é também muito utilizada e conhecida, o Questionário é uma técnica de levantamento de requisitos mais usados quando a informação a ser extraída é de um grande número de pessoas e os assuntos abordados são os mesmos, desta forma simplificando a tarefa e fazendo-a ser mais eficiente.

Deve ser desenvolvido um controle que identifique todas as pessoas que receberão os questionários. A distribuição deve ocorrer junto com instruções detalhadas sobre como preenchê-lo e ser indicado claramente o prazo para devolução do questionário. (DIXON MORAES, 2009)

Nessa técnica diferente da entrevista, segundo Dixon Moraes (2009) temos as questões sendo dirigidas por escrito ao grupo de participantes e o objetivo final é poder ser realizada uma análise sobre as diferentes opiniões e pontos de vista sobre o mesmo assunto, podendo ao juntar todas essas opiniões em uma forma só e realizando um produto que agrade de maneira geral.

Na fase de preparação do questionário deve ser indicado o tipo de informação que se deseja obter. Assim que os requisitos forem definidos o analista deve elaborar o questionário com questões de forma simples, clara e concisa, deixar espaço suficiente para as repostas que forem descritivas e agrupar as questões de tópicos específicos em um conjunto com um título especial (DIXON MORAES, 2009)

Segundo Dixon Moraes (2009) o questionário é uma boa técnica justamente por atingir um grande número de pessoas e dessa forma trazendo uma visão mais ampla, porém o analista deve ter cuidado, pois ao mesmo tempo essa quantidade o de opiniões podem se dispersar justamente por tratarem muitas vezes de pontos de vista diferente e interpretações diferentes para cada entrevistado, cabe ao analista juntar todos os dados coletos ao final e assim realizar o melhor padrão conforme as ideias vistas.

### UML

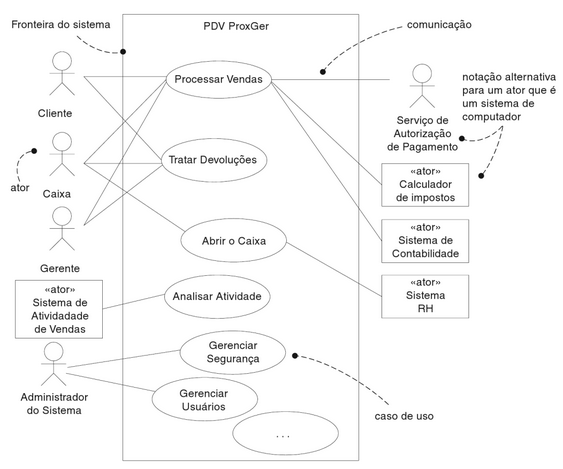
Larman (2007, p. 39) define UML, *Unified Modeling Language* traduzido como Linguagem de Modelagem Unificada, como uma notação diagramática padrão, usada para desenhar ou apresentar figuras relacionadas a *software*, Booch, Rumbaugh e Jacobson (2005, p. 14) definem como uma linguagem para elaboração da estrutura de projetos de *software* e diz que sempre será necessário utilizar vários modelos, referindo se aos diagramas, pois estes conectados tornam possível a compreensão do sistema. Unindo estas duas visões é possível concluir que UML é uma linguagem usada para representar graficamente os comportamentos, sequência de passos dos processos, como os usuários vão utilizar o sistema ou produto, e que para isso são usados diagramas esquemáticos, que serão descritos nos tópicos a seguir.

#### DIAGRAMA DE CASO DE USO

Segundo Pressman (2009, p. 137) “um caso de uso conta uma história estilizada sobre como o usuário final (desempenhando um de uma série de papeis possíveis) interage com o sistema sob um conjunto de circunstancias especificas” e Larman (2007, p. 20) complementa “um caso de uso é utilizado para estruturar o comportamento de itens em um modelo”.

O objetivo deste diagrama é representar graficamente o *software* de um ponto de vista diferente, utilizando de atores, que representam as diferentes pessoas ou dispositivos que usam o produto no contexto que será descrito, Pressman (2009, p. 137) define um ator como “qualquer coisa que se comunica com o sistema ou o produto e que é externa ao sistema em si”, esses atores podem interagir com o produto, executar tarefas, e essas ações que são representadas por elipses com uma curta e clara mensagem interna, que expõe a ideia principal da ação em questão e assim essa relação demonstra que o ator pode executar determinada ação.

**Figura 2.7** – Exemplo de caso de uso



**Fonte:** (LARMAN, 2007)

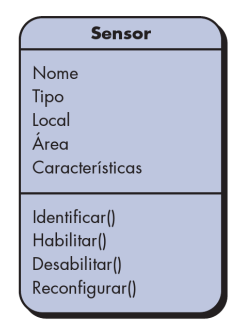
#### DIAGRAMA DE CLASSES

Pressman (2009, p. 166) afirma que a modelagem baseada em classes:

Representa objetos que o sistema irá manipular, as operações (também denominadas métodos ou serviços) que serão aplicadas aos objetos para efetuar a manipulação, os relacionamentos (alguns hierárquicos) entre os objetos e as colaborações que ocorrem entre as classes definidas.

O diagrama de classes visa representar graficamente esses objetos e suas operações, relacionamentos, hierárquicos ou não, e colaborações. Larman (2007, p. 18) descreve a representação gráfica de uma classe, “as classes são representadas por retângulos, geralmente incluindo seu nome, atributos e operações”.

**Figura 2.8** – Exemplo de Diagrama de Classe



**Fonte:** (PRESSMAN, 2009)

Por exemplo, na figura 2.8, existe uma classe Sensor, cujos atributos, também conhecidos como propriedades, são Nome, Tipo, Local, Área e Características, seus métodos, ou operações, Identificar, Habilitar, Desabilitar e Reconfigurar que acessam os atributos da classe se necessário para efetuar seus devidos propósitos.

#### DIAGRAMA DE ATIVIDADES

O diagrama de atividades é descrito de forma parecida por Larman (2007, p. 483), Booch, Rumbaugh e Jacobson (2005, p. 28), que dizem que este diagrama tem por objetivo representar o comportamento de um sistema descrevendo a sequência de ações em um processo, são parecidos com fluxogramas, pois mostram o fluxo entre as ações em uma atividade, estes podendo ser paralelos ou simultâneos.

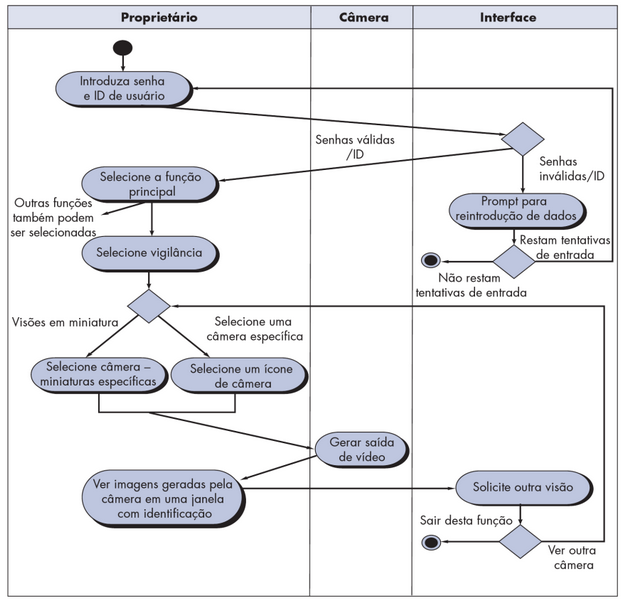
Pressman (2009, p. 161) acrescenta que este representa o “fluxo de interação em um cenário especifico” e o descreve:

Usa retângulos com cantos arredondados para representar determinada função do sistema, setas para representar o fluxo através do sistema, losangos de decisão com ramificação (cada seta saindo do losango é identificada) e as linhas horizontais cheias indicam as atividades paralelas que estão ocorrendo.

Porem graficamente é tão simples que acaba sendo muito intuitivo seu entendimento.

Para este trabalho foi optado por esquematizar o diagrama de atividades ao invés do diagrama de sequência, pois ambos têm objetivos muito semelhantes.

**Figura 2.9** – Exemplo de Diagrama de Atividades



**Fonte:** (PRESSMAN, 2009)

## PROJETO DE INTERFACE

O principal objetivo de projeto de interface segundo Souza, Leite, et al. (1999) é sensibilizar os engenheiros de *software* sobre algumas questões básicas importantes para o projeto de interface com o usuário. O projeto centrado no usuário é uma abordagem de projeto de interface onde a análise das atividades do usuário é primordial para o sucesso do projeto como um todo, projeto de interface sempre envolve o desenvolvimento de protótipos de interfaces.

Souza, Leite, et al. (1999) explica, o termo interface é aplicado normalmente àquilo que interliga dois sistemas. Tradicionalmente, considera-se que uma interface homem-máquina é a parte de um artefato que permite a um usuário controlar e avaliar o funcionamento deste artefato através de dispositivos sensíveis às suas ações e capazes de estimular sua percepção. No processo de interação usuário-sistema a interface é o combinado de *software* e *hardware* necessário para viabilizar e facilitar os processos de comunicação entre o usuário e a aplicação.

A interface entre usuários e sistemas computacionais diferencia-se das interfaces de máquinas convencionais por exigir dos usuários um maior esforço cognitivo em atividades de interpretação e expressão das informações que o sistema processa. (SOUZA, LEITE, *et al.*, 1999, p. 3)

Souza, Leite, et al. (1999) lembra que muitas vezes o designer deve identificar quais destes fatores têm prioridade sobre quais outros, uma vez que dificilmente se consegue alcançar todos de forma equivalente. As decisões do projetista determinam a forma de interação entre usuários e sistemas. Frequentemente designers definem a facilidade de uso como sendo o aspecto de usabilidade prioritário e, por vezes, acabam desenvolvendo sistemas em que os usuários não cometem erros, mas também não têm muita opção de ação ou decisão.

A meta do projeto de interface é definir um conjunto de objetos e ações de interface que permitam ao usuário realizar todas as tarefas estabelecidas, de um modo que satisfaça todas as metas de usabilidade definidas para o sistema. (PRESSMAN, 2009, p. 239)

Por mais que o levantamento de requisitos por si só levante uma infinidade de informações que deverão estar dispostas na interface, Souza, Leite, et al. Afirma que, é durante a interação real do usuário com o sistema que os detalhes realmente são percebidos. Para projetar a experiência perfeita, a utilização de protótipos se faz a forma mais eficaz de simular essa interação, devendo ser feita ainda no período de concepção do sistema, evitando gastos dispendiosos de tempo e monetários.

Segundo Souza, Leite, et al. (1999) explica os protótipos de baixa fidelidade, também chamados de rascunhos ou sketches, são concebidos ainda na fase inicial, durante a concepção do sistema. Desenhados geralmente à mão utilizando lápis, borracha e papel, essas representações são feitas de maneira rápida e superficial, apenas margeando a ideia do projeto e definindo superficialmente sua interação com o usuário, não se preocupando ainda com elementos de layout, cores, disposições, etc.

## DESENVOLVIMENTO DE *SOFTWARE*

Para entender o que é desenvolvimento de *software* é preciso saber o que é um *software*, Okuyama, Milleto e Nicolao (2014, p. 29) definem *software* como um programa que tem a finalidade de automatizar um processo ou resolver algum problema, Milleto e Bertagnolli (2014, p. 4) apresentam alguns exemplos, como, “um sistema que controla o metrô de uma cidade, um jogo de computador, um sistema de folha de pagamento de uma empresa”, entre outros.

Pressman e Maxim (2016, p. 4) utiliza de uma definição mais formal para explicar o que é *software*:

*Software* consiste em: (1) instruções (programas de computador) que, quando executadas, fornecem características, funções e desempenhos desejados; (2) estruturas de dados que possibilitam aos programas manipular informações adequadamente; e (3) informação descrita, tanto na forma impressa quanto virtual, descrevendo a operação e o uso do programa.

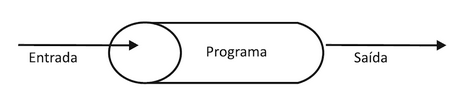
Milleto e Bertagnolli (2014, p. 4) afirmam que para “o desenvolvimento dessas aplicações, requer técnicas, métodos e mecanismos específicos para que elas funcionem conforme o esperado”. Essas técnicas, métodos e mecanismos são parte do processo de desenvolvimento de *software*, Milleto e Bertagnolli (2014, p. 21) afirmam que o uso de um processo é opcional, segundo Pressman e Maxim (2016, p. 17) o processo de desenvolvimento de *software* é “um conjunto de atividades, ações e tarefas realizadas na criação de um artefato”, mas lembrando de que um processo não é uma prescrição rígida de como desenvolver um *software*.

Portanto, desenvolvimento de *software* é, conforme citado por Milleto e Bertagnolli (2014, p. 21), um conjunto de atividades sistematizadas, que devem ser executadas em uma sequência pré-estabelecida de modo a agregar o máximo de valor possível a equipe envolvida no processo e ao produto que será produzindo com base na qualidade.

### PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)

Silva Filho (2011, p. 1) define programação como uma atividade que tem por objetivo o “desenvolvimento de programas que implementam determinadas funcionalidades” e capazes de produzir uma saída, ou seja, algum resultado.

**Figura 2.10** – Função de um programa



**Fonte:** (SILVA FILHO, 2011, p. 2)

Para Santos (2013) “Programação Orientada a Objetos, ou abreviadamente, POO é um paradigma de programação de computadores onde se usam classes e objetos” para “representar e processar dados usando programas de computadores”.

Silva Filho (2011, p. 11) explica, “uma classe serve como um padrão, modelo ou *template*”, ela “especifica quais dados e quais funções serão incluídos nos objetos daquela classe”, e lembra que definir uma classe não cria objetos, afinal, a classe é na realidade uma abstração para criar modelos baseados no mundo real.

A POO pode ser vista como o projeto de *software* utilizando uma coleção de objetos em cooperação, em oposição a uma vista tradicional, em que um programa pode ser visto como uma série de funções, ou simplesmente como uma lista de instruções para o computador. Em POO, cada objeto é capaz de receber mensagens, processar dados e enviar mensagens para outros objetos. Cada objeto pode ser visto como uma pequena máquina independente, com um papel ou responsabilidade distinta.

A POO se destina a promover uma maior flexibilidade e facilidade de manutenção na aplicação, e é muito popular em engenharia de *software*s de grande escala. Em virtude de sua forte ênfase na modularidade, código orientado a objetos destina-se a ser mais simples de desenvolver e mais fácil de entender mais tarde, prestando-se a uma análise mais direta, codificação e compreensão de situações e procedimentos mais complexos do que nos métodos de programação menos modulares. (TRASVIÑA, 2005)

### PROGRAMAÇÃO BASEADA EM PROTÓTIPOS

Segundo a Trasviña (2005), “Programação baseada em protótipos é um estilo de programação orientada a objetos na qual não existe a presença de classes” e lembra que “Este modelo também é conhecido como sem classes, orientado a protótipo, ou programação baseada em exemplares”. A relação de herança desse paradigma é realizada através do processo de decorar ou expandir objetos que servem como protótipo, ou seja, um objeto base, o protótipo, decora outro objeto, que permite que esse outro objeto utilize de seus recursos e funcionalidades.

O exemplo de uma linguagem baseada em protótipo citada por Trasviña (2005) é a linguagem de programação *Self* e logo afirma que este estilo de programação tem se tornado mais popular recentemente e vem sendo adotado por outros linguagens como *JavaScript, NewtonScript*, *Cecil*, entre outras.

### INTERATIVIDADE NA *WEB*

Nieradka (2014) lembra que o desenvolvimento *web* era puramente estático, mas que com o surgimento do *JavaScript*, ou JS, as páginas *web* passaram a ser interativas, possibilitando validação de formulários, caixas de diálogo, e muitos outros recursos.

Remoaldo (2008, p. 31) complementa, a interatividade é disponibilizada pela tecnologia *Flash* também, mas a diferença principal é que o *JavaScript* é integrado ao navegador e não precisa de um plug-in terceiro como o *Flash* que necessita do *Adobe Flash Player*.

Mas como é feita a interação do usuário com as páginas *web*? Cássio (2014, p. 49) afirma que a interação é dada por meio do teclado, mais precisamente, a leitura de eventos do teclado, e complementa que não é uma tarefa difícil com *JavaScript*.

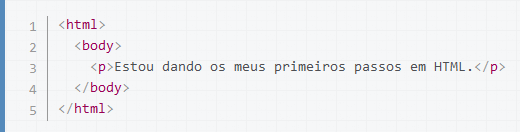
### TECNOLOGIAS

A seguir serão listadas e explanadas as tecnologias utilizadas para desenvolver este trabalho.

#### HTML5

Para Milleto e Bertagnolli (2014, p. 62) o *HTML*, ou *Hypertext Markup Language* traduzida como Linguagem de Marcação de Hipertexto, é “uma linguagem de marcação utilizada para criar páginas acessadas a partir de um navegador”. MDN (HTML, 2018) explica, o *HTML* é o bloco de construção mais básico de uma página *web*, é usado para criar e representar visualmente uma página. Ele determina o conteúdo de uma página, mas não sua funcionalidade.

**Figura 2.11** – Exemplo de documento *HTML*



**Fonte:** MDN (Introdução ao HTML)

A última versão estável do HTML, o HTML5 leva o HTML de uma marcação simples para estruturar um documento em uma plataforma de desenvolvimento de aplicativos completa. Entre outros recursos, o HTML5 inclui novos elementos e APIs JavaScript para aprimorar o acesso a armazenamento, multimídia e hardware. (MDN, 2015)

#### Canvas

Para Cássio (2014, p. 1) *Canvas* é “uma área retangular em uma página *web* onde podemos criar desenhos programaticamente, usando *JavaScript*”, a figura 2.12 mostra bem isso. Essa área desenhável pode ser usada, por exemplo, para desenhar gráficos, fazer composição de fotos, criar animações ou até mesmo fazer processamento de vídeo em tempo real.

**Figura 2.12** – Exemplo de código *Canvas* e sua renderização



**Fonte:** MDN (MDN)

Segudo a documentação da MDN (MDN) o *Canvas* foi originalmente criado pela Apple para seus widgets do *Dashboard*, mas desde então tem sido adotado pelos principais navegadores e agora faz parte do *HTML*5.

### ANATOMIA DE UM VIDEOGAME

A documentação apresentada pela MDN (Anatomia de um vídeo game) explica, o objetivo de cada jogo é apresentar ao usuário, uma situação, indentificar sua entrada, interpretar esses sinais em ações, e calcular uma nova situação resultante dos referidos atos. Os jogos estão constantemente em um ciclo através destes estágios, varias e varias vezes, até que alguma condição final ocorra, como, ganhar ou perder. Não surpreendentemente, este padrão corresponde a como o nucleo de jogo está programado.

Mais adiante a MDN (Anatomia de um vídeo game) explica, alguns jogos conduzem este ciclo através da entrada do usuário, por exemplo, um jogo dos sete erros. Este jogo apresenta duas imagens, aceita cliques ou toques na tela, interpreta a entrada como um sucesso, fracasso, pausa, interação de menus, etc. Finalmente, ele calcula uma cena atualizada resultante dessa entrada. O ciclo do jogo é avançado pela entrada do usuário e espera até que a proxima seja fornecida. Esta é uma abordagem baseada em turnos, que não exige uma atualização constante a cada segundo, somente quando o jogador reage.

Outros jogos exigem controle sobre a menor fatia de tempo possivel, afirma a MDN (Anatomia de um vídeo game) e segue, os mesmos princípios se aplicam como anteriormente citado com uma leve diferença: cada quadro de animação progride o ciclo e qualquer alteração na entrada do usuário é capturada no primeiro quadro disponível. Este modelo é implementado em algo chamado de laço principal. Se os ciclos do jogo forem baseados no tempo, então o jogo irá aderir a este modelo.

### TÉCNICAS

Existe uma vasta quantidade de técnicas que podem ser encontradas pela Internet, porem algumas são importantes demais para serem deixadas de lado, são essenciais para qualquer jogo digital em desenvolvimento. A seguir serão listadas as principais e mais conhecidas segundo a MDN (Técnicas para desenvolvimento de jogos).

#### ANIMAÇÕES EFICIENTES PARA JOGOS *WEB*

De acordo com a MDN (Efficient animation for web games) essa técnica aborda meios de criar animações eficientes para jogos *web*, com uma maior ênfase em dispositivos com poder de processamento inferior, como telefones celulares.

Algumas dessas técnicas podem ajudar a aplicação a consumir menos bateria e recursos de dispositivos de baixo desempenho, e assim suportando uma quantidade maior de dispositivos.

#### INICIALIZAÇÃO OTIMIZADA

Sobre esta tecnica a MDN (Optimizing startup performance) lembra que um aspecto muitas vezes esquecido no desenvolvimento de aplicações, mesmo entre aqueles com foco na otimização do desempenho é o desempenho de inicialização. Quanto tempo um aplicativo necessita para a inicialização? Parece que ele trava o dispositivo ou o navegador do usuário enquanto o aplicativo carrega?

Independentemente da plataforma, é sempre uma boa ideia iniciar o mais rápido possível e esta técnica oferece dicas e sugestões para garantir que a inicialização de uma aplicação ocorra rapidamente, sem travamentos e erros inesperados tanto ao escrever um novo aplicativo quanto ao portar para outra plataforma.

#### ÁUDIO PARA JOGOS *WEB*

Destacada pela MDN (Audio for Web games), a técnica de Áudio é uma parte importante de qualquer jogo, ele adiciona *feedback* e atmosfera. O áudio baseado em tecnologias *web* está amadurecendo rápido, mas ainda há muitas diferenças de navegador para navegador. Muitas vezes é necessário decidir quais áudios são essenciais para uma boa experiência de jogo e quais são agradáveis de ter, mas não essenciais, e elaborar uma estratégia nesse sentido.

De longe, as plataformas mais difíceis para fornecer suporte de áudio na *web* são as plataformas móveis. Infelizmente, estas são também as plataformas que as pessoas em sua maioria usam para jogar, e essa técnica tenta minimizar ao máximo estes problemas.

#### DETECÇÃO DE COLISÃO 2D

Conforme descrito pela MDN (2D collision detection) esta técnica consiste em utilizar algoritmos para detectar colisões 2D, independente do tipo de formas que podem colidir, por exemplo, entre retangulos, entre um retangulo e um círculo, entre cirulos, e ainda colisões entre poligonos irregulares, mais tarde afirma, esta tecnica apresenta os metodos mais comuns e de alto desempenho utilizados para proporcionar a detecção de colisão dos diversos tipos de formas.

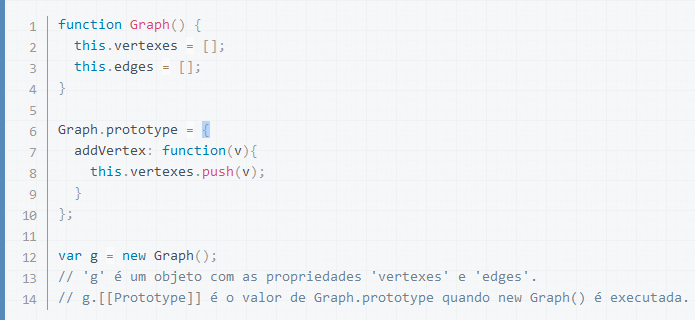
### FERRAMENTAS DE DESENVOLVIMENTO

Uma vez que a Internet evoluiu ao longo dos anos, as tecnologias por ela utilizadas também evoluíram, e isso permitiu que o número de ferramentas para o desenvolvimento de aplicações *web* acompanhasse esse crescimento, a seguir são apresentadas algumas ferramentas disponíveis, escolhidas pela MDN (Ferramentas para o desenvolvimento de jogos) para facilitar a criação de experiências de jogos na *web*, tais como frameworks e linguagens de programação *web*.

#### JavaScript

Remoaldo (2008, p. 35) explica, a linguagem *JavaScript* “é uma linguagem de *scripting* interpretada que está disponível de forma gratuita na maioria dos *browsers* modernos”. A MDN (Introdução) descreve o *JavaScript* como uma linguagem orientada a objetos, multiplataforma, pequena e leve, onde, dentro de um ambiente de *host*, por exemplo, um navegador *web*, pode ser ligado aos objetos deste ambiente para prover um controle programático sobre eles.

**Figura 2.13** – Exemplo de código *JavaScript*



**Fonte:** MDN (Herança e cadeia de protótipos (prototype chain))

Segundo documentação da MDN (Introdução), núcleo do *JavaScript* pode ser estendido para uma variedade de propósitos, complementando assim a linguagem:

O lado cliente do JavaScript estende-se do núcleo da linguagem, fornecendo objetos para controlar um navegador *web*. Por exemplo, as extensões do lado do cliente permitem que uma aplicação coloque elementos em um formulário *HTML* e responda a eventos do usuário, como cliques do mouse, entrada de formulário e de navegação da página.

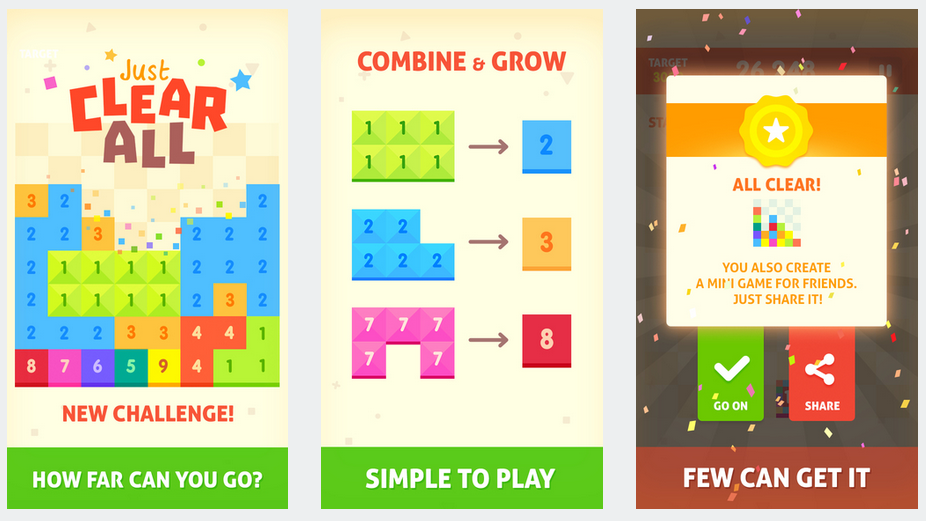
O lado do servidor do JavaScript estende-se do núcleo da linguagem, fornecendo objetos relevantes à execução do *JavaScript* em um servidor. Por exemplo, as extensões do lado do servidor permitem que uma aplicação se comunica com um banco de dados, garantindo a continuidade de informações de uma chamada para a outra da aplicação, ou executar manipulações de arquivos em um servidor.

O *JavaScript* foi escolhido para este trabalho porque não exige nenhuma extensão ou biblioteca terceira para o desenvolvimento de aplicações *web*. Como está presente em todos os navegadores modernos, por exemplo, Mozilla Firefox, Google Chrome, Opera e Microsoft Edge, e por ser uma linguagem multiplataforma, permite um alcance maior de usuários, e permiti ao programador maior liberdade ao estruturar e codificar a aplicação, ao contrário de um *framework*, que pode restringir o programador em diversos pontos.

#### Cocos2d-JS

Cocos2d-JS[[3]](#footnote-3) é um motor de código aberto para criar jogos *web* e nativos. Tem uma alta performance, é amigável com o usuário e suporta o desenvolvimento multiplataforma. As plataformas suportadas incluem *web*, *Android*, *iOS*, *Windows* *Phone* 8, *Mac*, *Windows*, etc. O *Cocos2d-JS* faz a programação de jogos 2D mais fácil e rápida. Ele simplifica os componentes principais da programação de jogos 2D sendo fácil de aprender e possui uma de fácil uso.

**Figura 2.14** – Jogo criado com Cocos2d-JS



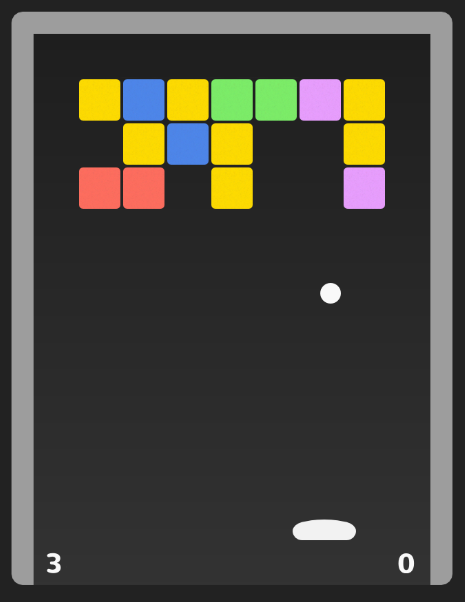
**Fonte:** http://cocos2d-x.org/games/3036

*Cocos2d-JS* é a versão *JavaScript* do *Cocos2d-x*, que também é um motor de jogos, de código aberto. Ele permite aos desenvolvedores explorar o seu conhecimento em C++, Lua e *JavaScript* com o desenvolvimento em plataforma cruzada para iOS, Android, *Windows* *Phone*, *Mac* OS X, *Windows* *Desktop* e Linux, poupando tempo, esforço e custo.

#### Stage.js

*Stage.js*[[4]](#footnote-4) é uma biblioteca 2D *HTML*5 escrita em *JavaScript* para desenvolvimento de jogos multiplataforma, ela é leve, rápida e de código aberto. Internamente gerencia ciclos de renderização e de desenho da aplicação, também processa e distribui para os respectivos responsáveis os eventos de clique e toque.

**Figura 2.15** – Jogo desenvolvido com Stage.js



**Fonte:** http://piqnt.com/stage.js/example/game-breakout/

#### Isogenic Engine

*Isogenic Engine*[[5]](#footnote-5) é um sistema *JavaScript*, executado do lado do cliente e também do servidor, que fornece um grafo de cena, com uma funcionalidade de multijogadores embutida. Simulações em execução no lado do servidor podem ser transmitidas automaticamente para todos os clientes ligados à rede ou seccionado em salas distintas em que apenas os clientes que estão participando recebem as atualizações.

O motor apoia completamente tanto renderização 2D quanto isométrica e pode ser executado com ou sem funcionalidade multijogador. Tem um sistema de construção incluído para analisar automaticamente o código final do jogo, que detecta as seções da *API* do *Isogenic* que tenham sido usadas e minifica o jogo e o Isogenic, para obter o menor tamanho possível do jogo já pronto para distribuição.

# METODOLOGIA

Pradanov e Freitas (2013, p. 14) descrevem metodologia como “uma disciplina que consiste em estudar, compreender e avaliar os vários métodos disponíveis para a realização de uma pesquisa acadêmica” e afirmam que seu objetivo é avaliar métodos e técnicas de pesquisa que possibilitam a coleta e o processamento de informações para solucionar problemas e/ou questões de investigação.

Silva e Menezes (2005) em um artigo da UFSC concluem que “Pesquisa é um conjunto de ações, propostas para encontrar a solução para um problema, que têm por base procedimentos racionais e sistemáticos” e é realizada “quando se tem um problema e não se têm informações para solucioná-lo”.

Segundo Gil (2002, p. 43-45) existe várias formas de classificar as pesquisas. Do ponto de vista dos procedimentos técnicos:

* Pesquisa Bibliográfica, que é elaborada com base em material já existente, como livros, artigos científicos, e disponibilizados na internet, o caso deste trabalho.
* Pesquisa Documental, desenvolvida a partir de materiais que não receberam tratamento analítico.
* Estudo de caso, quando é necessário o estudo profundo de maneira que se permita o seu amplo e detalhado conhecimento.



## INSTRUMENTO DE COLETA DE DADOS

Para a coleta das informações necessárias para o levantamento de requisitos foi aplicada a técnica de entrevista, os resultados obtidos podem ser visualizados na seção 4.1 deste trabalho.

## CRONOGRAMA

A seguir o cronograma com as atividades desenvolvidas durante todo o trabalho.

**Tabela 3.1** – Cronograma das atividades desenvolvidas

|  |  |
| --- | --- |
| **Meses** | **Atividade** |
| Janeiro de 2017 | Entrevista |
| Janeiro de 2017 | Levantamento de requisitos |
| Fevereiro de 2017 | Modelagem estática (Diagrama de classes e Casos de Uso) e dinâmica (Diagrama de Atividades) |
| Março de 2017 | Protótipos de tela de baixa e alta fidelidade (Mockups) |
| Março/Abril/Maio de 2017 | Desenvolvimento do motor da aplicação |
| Maio/Junho de 2017 | Implementação das telas |
| Julho/Agosto de 2017 | Teste de unidade e integração |
| Setembro de 2017 a Junho de 2018 | Documentação do trabalho |

**Fonte:** Autores deste trabalho

## ATIVIDADES

Neste tópico será apresentado, de forma resumida, o que aconteceu em cada passo do desenvolvimento do trabalho, já os resultados serão apresentados no próximo c*apí*tulo.

* Entrevista: Nesta etapa visava-se recolher o máximo de informações dos interessados no projeto de forma a garantir que todas as expectativas fossem atingidas ao final do trabalho, ou seja, que nenhum detalhe fosse deixado de lado.
* Levantamento de requisitos: Com base nos dados coletados na fase anterior foram criadas as fichas de requisitos, presentes no Apêndice A, que descrevem melhor cada uma das necessidades levantadas pelos interessados.
* Modelagem estática e dinâmica: Foram criados diagramas para representar os objetos do sistema e as relações entre eles, a sequência de passos executada durante os processos do jogo e as suas interações com os jogadores.
* Protótipos de tela de baixa e alta fidelidade: Tinha a finalidade de criar protótipo de telas do jogo, como tela inicial, de fim de jogo, a própria tela onde o jogo iria ocorrer, entre outras, onde o protótipo de baixa fidelidade era apenas um simples esboço que serviria de base para as telas de alta finalidade.
* Desenvolvimento do motor da aplicação: Etapa de pura programação, basicamente todos os dados criados até então se tornaram código e deram vida ao jogo.
* Implementação das telas: Com base nos protótipos de tela anteriormente elaborados foram codificadas as telas, ou seja, foram construídas programaticamente.
* Documentação do trabalho: Elaboração deste documento.

# DESENVOLVIMENTO

Nesta etapa são apresentados os resultados obtidos durante o desenvolvimento deste trabalho.



## ENTREVISTA

A entrevista foi feita entre os membros do grupo deste trabalho, onde alguns assumiram o papel de entrevistadores e outros de *stakeholders*, uma espécie de cliente que tem as informações mais pertinentes para o desenvolvimento da aplicação em questão.

O objetivo nesta etapa foi reunir o máximo de informações sobre a visão final que o cliente tinha do trabalho a ser executado, afinal é preciso saber o que o cliente quer para então desenvolver. A entrevista completa pode ser encontrada no Apêndice E deste trabalho.

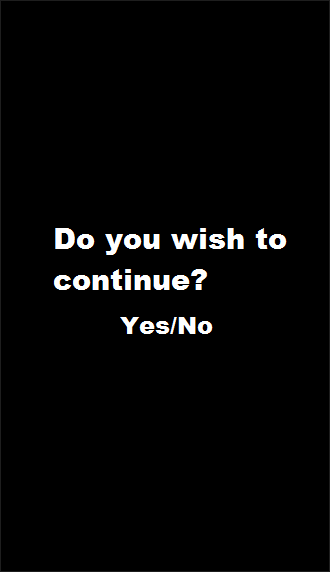
## LEVANTAMENTO DE REQUISITOS

Como resultados desta atividade, utilizando-se das informações levantadas na etapa de entrevista, foram elaboradas as fichas de requisitos que estão anexadas a este trabalho na forma de um apêndice, vide apêndice A.

## DESENVOLVIMENTO DA INTERFACE

Com base na entrevista e requisitos levantados na fase inicial do trabalho foram elaborados os protótipos de baixa e alta fidelidade. Todas as telas do jogo são muito simples, portanto, produzi-las não foi uma tarefa tão complicada, principalmente as telas de baixa fidelidade.

**Figura 4.1** – Protótipo de baixa fidelidade da tela de continuação



**Fonte:** Autores deste trabalho

A figura acima apresenta o protótipo da tela de continuação, que pergunta ao jogador se ele deseja continuar, ela será exibida sempre que o jogador perder um crédito a não ser que ele já tenha perdido todos, nesse caso será exibida a tela de fim de jogo, vide figura 4.2. Na tela de continuação o jogador pode escolher apenas entre duas opções, *yes,* sim e *no*, não.

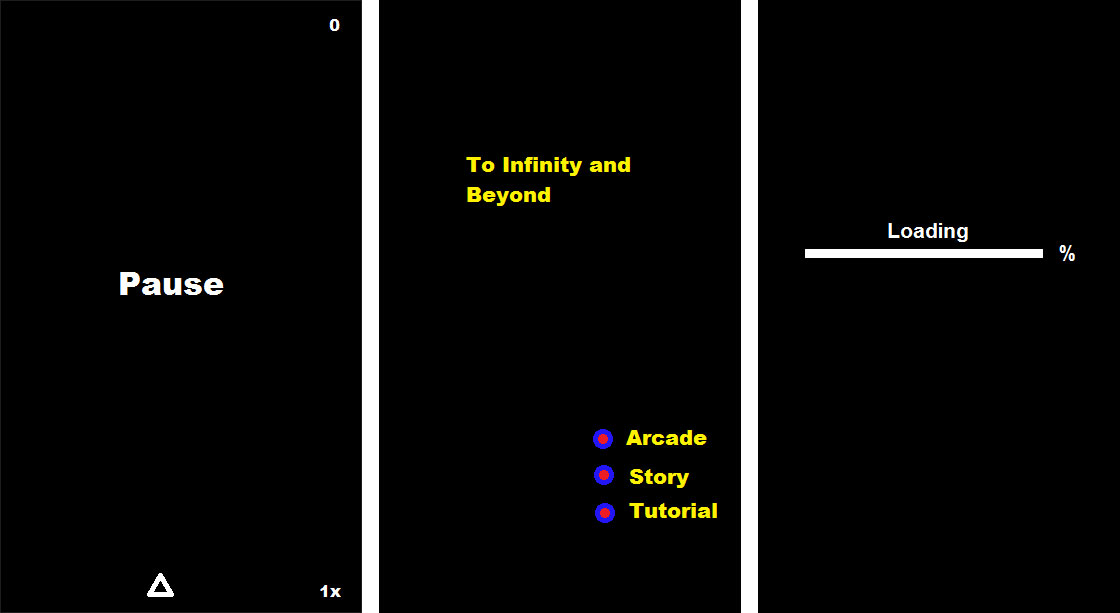
**Figura 4.2** – Protótipo de baixa fidelidade da tela de fim de jogo



**Fonte:** Autores deste trabalho

As telas pausa, inicial e carregamento são bem semelhantes, possuem apenas pequenos trechos de textos com poucas ou até mesmo sem opções:

**Figura 4.3** – Protótipo de baixa fidelidade das telas de pausa, inicial e carregamento



**Fonte:** Autores deste trabalho

Os protótipos de baixa fidelidade foram todos desenvolvidos no *Microsoft Paint* enquanto os protótipos de alta fidelidade foram desenvolvidos no *Adobe Photoshop CS6*.

As diferenças entre os protótipos de baixa e alta fidelidade das telas de pausa, continuação, fim de jogo e carregamento, resume se a definição de uma nova fonte para os textos e do fundo agora possui algumas estrelas, tentado dessa forma reproduzir o espaço.

**Figura 4.4** – Protótipos de alta fidelidade das telas de pausa, continuação, fim de jogo e carregamento



**Fonte:** Autores deste trabalho

E por fim o protótipo de alta fidelidade da tela inicial, que ganhou uma arte de fundo e fontes diferentes.

**Figura 4.5** – Protótipo de alta fidelidade da tela inicial



**Fonte:** Autores deste trabalho

## DESENVOLVIMENTO DO MOTOR DA APLICAÇÃO

O motor da aplicação foi desenvolvido, conforme citado anteriormente, utilizando a linguagem programação *web*, *JavaScript*, combinada com o paradigma de Orientação a Objetos.

Nenhuma ferramenta especifica foi utilizada, apenas um editor de texto moderno para acelerar e simplificar o processo de codificação foi escolhido o *Sublime Text 3*[[6]](#footnote-6).

O código fonte completo possui 1656 linhas, por ser muito extenso foi anexado a este trabalho na forma de um apêndice, apêndice C.

### FRAMES

Este é um jogo que está constantemente atualizando a tela do jogo, cada atualização acontece em um frame, uma fatia de tempo muito pequena, menor até mesmo que um segundo, e para que isso funcione corretamente existe um ciclo de instruções que precisão ser executadas apara controlar as atualizações. Um exemplo de atualização neste ciclo é o fundo do jogo que esta sempre em movimento, a cada frame ele é desenhado um pouco para baixo e dessa forma causando a impressão de que a nave do jogador está em movimento.

Outro item que vale a pena destacar, em meio as suas tarefas o ciclo faz algumas asserções e decide se deve criar e atualizar a posição de inimigos, itens, explosões, disparos, a interface do usuário e a nave do jogador:

Fica claro que este ciclo é encarregado de muitas tarefas, mas resumidamente sua função é verificar e atualizar o estado atual do jogo e redesenhar a tela resultante de acordo com o respectivo estado.

### RECURSOS EXTERNOS

Os gráficos e efeitos sonoros são arquivos externos armazenados nas suas respectivas pastas dentro da pasta da aplicação, e são carregados durante a inicialização do jogo.

Dentro do código da aplicação existe uma lista com os nomes dos recursos necessários, uma vez que o navegador inicia o carregamento do jogo o processo de leitura dos recursos externos é juntamente iniciado, quando um recurso acaba de ser carregado a barra de progresso é atualizada.

**Figura 4.6** – Tela de carregamento



**Fonte:** Autores deste trabalho

Uma curiosidade, a tela de carregamento também possui recursos que precisam ser carregados, como por exemplo, a imagem de fundo e a própria barra de carregamento, porém estes itens ganham prioridade no processo e se tornam os primeiros itens da lista, devido sua grande importância que é justamente informar o usuário sobre o estado atual de inicialização da aplicação.

O carregamento de recursos sonoros e gráficos são semelhantes em quase toda sua totalidade, diferenciando-se apenas na forma como o código lida com cada um posteriormente. Para o usuário, isso tudo é transparente, tudo que ele verá e precisa ver, é a barra de carregamento sendo preenchida.

### CONTROLES

O principal e único meio de interação da aplicação se da através do teclado. Para identificar as interações do jogador foram configurados eventos, um evento é uma espécie de gatilho, que dispara uma reação pré-definida quando uma determinada ação acontece.

Foram criados eventos para escutar, as setas que movimentam a nave nas quatro direções, direita, esquerda, cima e baixo, a tecla *Shift* que enquanto pressionada diminui a velocidade da nave, P para pausar o jogo, *Enter* para confirmar opções durante os menus do jogo e Z para efetuar disparos.

### MENUS

Durante o jogo são apresentadas algumas telas e não surpreendentemente entre elas existem menus, tela inicial, tela de continuação e na tela de *gameover*, novamente faz-se uso dos eventos para identificar as teclas pressionadas pelo jogador para alternar entre as opções dos menus e a tecla *Enter* para identificar a opção escolhida.

Decorrente da escolha do usuário em um menu outra tela poderá ser chamada, por exemplo, selecionar a opção *Retry* na tela de fim de jogo apenas reinicia a partida, porém sem sair da tela de jogo ao contrario quando selecionada a opção *Menu* que leva o jogador a tela inicial do jogo.

**Figura 4.7** – Menu de fim de jogo



**Fonte:** Autores deste trabalho

Existem ainda uma espécie de menu sem opções, a tela de pausa, se trata de um menu porque o jogador pode optar por pausar e continuar o jogo, ou seja, é uma seleção de opção indireta.

**Figura 4.8** – Tela de pausa



**Fonte:** Autores deste trabalho

### NAVE

Este é o personagem principal do jogador, comumente é uma pessoa, mas dependendo do tipo de jogo ele pode variar desde um carro de corrida a um time de futebol, neste caso uma nave representa o jogador.

**Figura 4.9** – Personagem principal do jogador

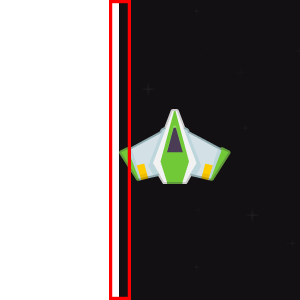


**Fonte:** Autores deste trabalho

Mesmo sendo uma nave, o personagem possui três vidas, podendo disparar *lasers* e acumular pontos. A nave é desenhada na tela utilizando os recursos nativos do navegador e é controlada pelo jogador através do teclado, por meio dos eventos citados anteriormente.

Existe um verificador para garantir que a nave nunca saia da tela, ou seja, quando uma das extremidades da nave alcança os limites da tela qualquer tentativa de mover a nave além daquele ponto é ignorada pelo jogo.

**Figura 4.10** – Limite da tela em destaque



**Fonte:** Autores deste trabalho

Os disparos da nave, controlados através da tecla Z, possuem um temporizador que regula o tempo entre os disparos para que não seja possível atirar descontroladamente. Acerca da movimentação dos disparos, existe também um controle que basicamente atualiza a posição do disparo utilizando uma velocidade pré-definida, que é constante, e assim em cada frame o disparo é desenhado mais a frente até que saia da tela de jogo ou atinja um inimigo e isso tudo acontece constantemente com cada disparo na tela.

Uma vez que os disparos saem da tela ou atingem um inimigo eles são removidos da tela, este comportamento é comum entre vários objetos desta aplicação, pois uma vez que uma destas situações aconteça o disparo ou o recurso em questão não está mais em jogo.

### INTERFACE DE USUÁRIO

Um dos itens mais comuns dos jogos é a interface de usuário, também conhecida com *heads-up display, HUD,* que exibe informações do jogador, conforme pode ser visto na Figura 4.8.

**Figura 4.11** – Interface de usuário destacada



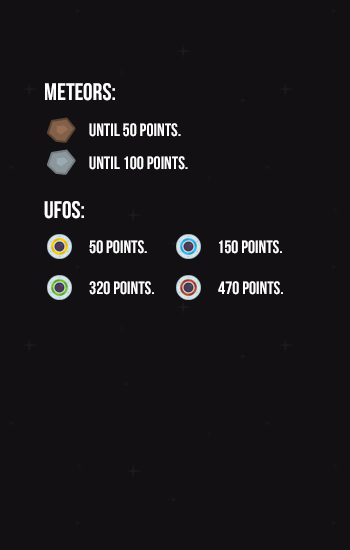
**Fonte:** Autores deste trabalho

Na imagem a cima é possível ver a interface de usuário destacada, onde, os raios representam as vidas do jogador, os números no canto superior direito a pontuação, e no canto inferior direito o multiplicador que deve multiplicar os pontos obtidos através da derrota de inimigos ou de pontos coletados, pelo valor do multiplicador acumulado e enfim somar a pontuação final.

### INIMIGOS

*UFO’s* e Meteoros são os dois tipos de inimigos que o jogador vai encontrar durante uma partida e existe ainda o disparo do *UFO* que pode também ser considerado um inimigo, a colisão com qualquer um desses é tratado pela aplicação como uma derrota, logo, uma vida é perdida e a tela de continuação é exibida e no caso de não existir mais vidas restantes a tela de fim de jogo é que será exibida.

**Figura 4.12** – Inimigos



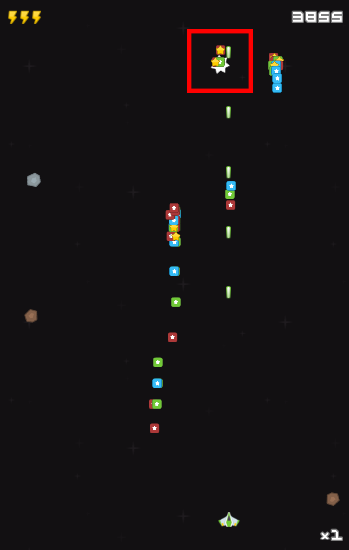
**Fonte:** Autores deste trabalho

Da mesma forma como os disparos da nave, a criação de novos inimigos é limitada por um temporizador para garantir que não haja um exagero grande demais no numero de inimigos na tela, e ao sair da tela também são removidos do jogo.

Cada inimigo tem uma variedade de tipo, os meteoros possuem duas variações, a única diferença entre elas é a pontuação e velocidade de movimento, porem os *UFO’s* tem até 4 formas diferentes, não se restringindo somente aos pontos, mas o comportamento e velocidade de seu movimento e tipo de disparos também mudam. Lembrando que ao cria um novo inimigo a escolha do tipo, variação e posicionamento inicial é totalmente aleatória.

Além dos pontos obtidos ao derrotar os inimigos, estes podem explodir e lançar itens na tela, a quantidade e tipo também variam de acordo com o inimigo e tipo, os itens serão melhor descritos nas próximas seções.

**Figura 4.13** – Itens lançados pelos inimigos



**Fonte:** Autores deste trabalho

Para cada tipo de UFO existe um tipo de disparo e cada um se movimenta de forma e em velocidade diferente o que implica em uma atualização de posicionamento bem delicada, porem trás uma diversidade maior para entreter o jogador com mais desafios.

### COLISÕES E EXPLOSÕES

Um jogo sem colisões não seria divertido afinal não haveria desafio para o jogador, por isso a aplicação possui um mecanismo para identificar colisões entre a nave do jogador, os inimigos e os disparos.

O código da aplicação identifica uma colisão verifica quando um objeto interseciona com outro objeto isso fica mais claro observando-se a Figura 4.14:

**Figura 4.14** – A área preenchida exemplifica a intersecção

**Fonte:** Autores deste trabalho

Todos os objetos do jogo que podem colidir tem uma configuração que indica a área de colisão, então quando outro objeto faz intersecção com esta área é identificada a colisão no jogo, na Figura 4.15 é possível observar a colisão de um meteoro e a nave do jogador, as áreas destacadas são as áreas de colisão e a área preenchida é o ponto de intersecção.

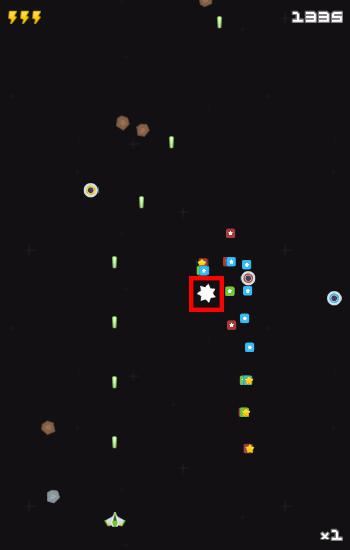
**Figura 4.15** – Colisão

**Fonte:** Autores deste trabalho

A identificação de colisão funciona desta forma, mas quando ela ocorre? A resposta é simples o tempo todo, no ciclo principal da aplicação, citado anteriormente, esse mecanismo para verificar colisões é chamado em cada iteração passando por todos os objetos, nave do jogador, disparos do jogador, meteoros, *UFO’s* e os disparos dos *UFO’s.* Por conta disso que é importante que quando os objetos deslocarem-se para além da tela estes sejam removidos, pois dessa forma diminui o numero itens a serem processados durante a colisão.

Quando a colisão é com a nave do jogador, uma vida é descontada e a tela de continuação é exibida, quando é com um inimigo o mesmo é removido da tela e uma animação de explosão, com efeito sonoro, é apresentada no seu lugar e se for o caso os itens são lançados.

**Figura 4.16** – Explosão em destaque

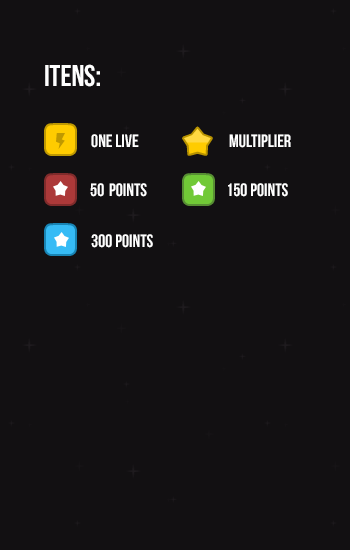


**Fonte:** Autores deste trabalho

### ITENS

Ao derrotar os inimigos itens podem ser lançados na tela, existem três tipos de itens, alguns apenas rendem pontos, outros incrementam o multiplicador e ainda existe o item que pode dar uma vida extra ao jogador.

**Figura 4.17** – Itens



**Fonte:** Autores deste trabalho

Uma vez que o item sair da tela, caso o jogador não o colete, este será removido do jogo sem qualquer outro efeito colateral, agora caso o jogador colete o item uma devida ação será tomada pela aplicação. No caso de itens de pontuação, o valor do item será multiplicado pelo valor do multiplicador acumulado e então somado a pontuação total, itens do tipo multiplicador apenas incrementaram o contador de multiplicador e algo semelhante acontece com o item de vida extra, o contado de vidas é incrementado quando o item é coletado, porém este último é muito raro.

Lembrando que a quantidade de cada item é aleatória, mas respeitando um limite pré-definido por tipo de inimigo, dessa forma inimigos mais fortes podem gerar mais itens valiosos do que outros mais fracos.

A coletada do item faz uso do verificador de colisão, mas neste caso a colisão da nave com o item é vista de forma diferente, a colisão significa que a nave coletou o item.

## IMPLEMENTAÇÃO DAS TELAS

A implementação da interface em sua maioria é desenhar pedaços de um *spritesheet*, uma imagem constituída de outras pequenas imagens. Uma vez que o recurso está carregado, basta usar as funcionalidades do *Canvas* para desenhar pedaços selecionados da *spritesheet*.

**Figura 4.18** – *Spritesheet* do jogo



**Fonte:** Autores deste trabalho

Para desenhar as partes da *spritesheet* é muito simples, o próprio *Canvas* disponibiliza meios para desenhar pedaços de uma imagem, basta informar as coordenadas de origem e destino e o tamanho da área que deseja desenha que ele trata de recortar e desenhar na tela.

Observando a Figura 4.18 e em seguida a Figura 4.19 é possível ver os elementos da *spritesheet* desenhadas na tela de jogo.

**Figura 4.19** – Pedaços da *spritesheet* em destaque



**Fonte:** Autores deste trabalho

No caso da tela inicial apenas uma imagem é utilizada e por cima dela é desenhada a bolinha ao lado da opção de menu atualmente selecionada e quando o jogador troca de opção a imagem é redesenhada e a bolinha também, porem agora em uma nova posição.

**Figura 4.20** – Tela inicial sem seletor de opção de menu



**Fonte:** Autores deste trabalho

# IMPLEMENTAÇÕES FUTURAS

Muitas ideias surgiram durante o desenvolvimento do trabalho, algumas factíveis e outras não.

Sem dúvidas que jogar com ou contra alguém é muito mais divertido, esta é uma funcionalidade que está presente em grande parte dos jogos de qualquer plataforma. A possibilidade de disputar partidas em modo cooperativo, dois jogadores se ajudando para cumprir um objetivo em comum, ou versus, um jogador contra o outro, é viável e abre a possibilidade para diversos novos cenários e combinações de partidas.

A construção de fases também é algo que foi pensado e discutido, pois permite aumentar a dificuldade em etapas, criando mais desafios e possibilitando um tipo de compromisso com o jogador. Foram levantas diversas ideias, como fases em que é preciso proteger um tipo de aliado, concluir algum objetivo em determinado tempo, coletar uma determinada quantidade de um item especifico, entre outras. Com a estrutura que o trabalho apresenta atualmente é possível implementar, porém exigiria uma demanda de tempo muito grande, portanto, decidiu-se que seria mais sensato adiar o desenvolvimento desta funcionalidade.

Um atrativo para os jogadores é a capacidade de personalização de diversos aspectos do jogo, como, controles, musica, visual do personagem, tudo isso para proporcionar melhor aceitação e familiaridade. A personalização pode ir um pouco além do aspecto visual, permitindo personalizar atributos que podem influenciar diretamente no jogo. Também como em grandes jogos, existem as famosas lojinhas de melhorias, onde seria possível comprar, utilizando apenas recursos coletados através do próprio jogo, novas naves, poderes, itens raros ou eventuais, vidas extras, ou seja, uma grandiosa quantidade de diferentes itens.

Outra possibilidade, não menos importante, seria tornar o tutorial do jogo interativo, isto é, conforme o jogo vai se desenvolvendo mensagens explicativas de como funciona e o objetivo de determinada ação aparecem na tela para ensinar o jogador.

E não podia faltar um ranque geral de maiores pontuações, criando assim uma disputa entre os jogares para ser o melhor no jogo. Eventualmente seria ofertado um prêmio para os três melhores do mês, algum tipo de item único.

Muitas ideias surgiram durante todo o processo, muitas ruins, algumas boas e outras não viáveis ou muito onerosas. As melhorias comentadas foram as mais discutidas e consideradas como as com mais chances de sair do papel e serem desenvolvidas e implementadas.

# APÊNDICES

Todos os apêndices citados no decorrer deste trabalho encontram-se hospedados em um repositório na internet, para acessa-los acesse <https://github.com/DiegoVictor/TCC/tree/master/Apêndices>.

# RESULTADOS E DISCUSSÕES

Durante cada etapa do desenvolvimento varias informações foram processadas e consequentemente resultados foram obtidos, entre eles, fichas de requisitos, diagramas de caso de uso, atividades e classe, protótipos de tela e o próprio código fonte, todos estão presentes nos respectivos apêndices deste trabalho.

Retornando a uma adversidade muito importante levantada inicialmente e que agora pode ser discutida, questionava-se as ferramentas convencionais de apoio e aceleração ao desenvolvimento de software, seriam elas capazes de realmente auxiliar a criação de jogos *web*? Indo um pouco mais além, seriam capazes de amparar o desenvolvimento de jogos que utilizam apenas de recursos nativos dos navegadores modernos?

Estas questões tornavam a viabilidade da aplicação no mínimo duvidosa, mas agora com todos os resultados em mãos é fácil desmistificar esse questionamento, entretanto, uma ferramenta mais especializada permitiria um desempenho e resultados superiores. Porque? O diferencial destas ferramentas é a automatização de tarefas corriqueiras, não implicando que sejam simples, por exemplo, colisão entre objetos irregulares, conceitos de física, como gravidade, aceleração, construção de cenários mais complexos e mais longos e tudo isso mantendo um bom funcionamento da aplicação, com muitas dessas tarefas prontas seria possível produzir mais com menos tempo e com segurança e consistência.

Existem sim ferramentas mais especializadas, porém são poucas as opções, e em alguns casos podendo até limitar o desenvolvimento em alguns aspectos, mas claro, isso é característico de uma ferramenta para outra.

Com a grade variedade de jogos disponíveis na rede, para que uma aplicação se destaque é muito importante que ela tenha algo de diferente, e mesmo utilizando apenas as ferramentas convencionais foi possível destacar-se em relação às demais em alguns elementos. Ainda são poucos os jogos que utilizam apenas tecnologias nativas disponibilizadas pelos navegadores mais modernos, ou seja, mais pessoas terão acesso e sem a necessidade de qualquer tecnologia terceira.

Um exemplo clássico, os jogos disponíveis em redes sociais como o *Facebook*, em sua grande maioria são desenvolvidos utilizando o Adobe Flash Player, um reprodutor de multimídia e aplicações amplamente distribuído, é instalado como um pacote extra nos navegadores, sem esse pacote instalado o jogo não pode ser executado, diminuindo a acessibilidade da aplicação, pois infelizmente ainda existem dispositivos que podem não ter suporte ou simplesmente não permitir a instalação de pacotes de terceiros.

Em relação a aplicações que também foram construídas com essas mesmas tecnologias *web* nativas, algumas diferenças que podem ser citadas, partidas pequenas e rápidas, nada de ficar esperando algo terminar ou acontecer para não perder progresso, nenhuma restrição etária e vale destacar, atualmente existem poucos jogos neste estilo, mas o grande destacamento de *To Infinity And Beyond* é seu estilo de jogo, rolagem vertical com naves, espaciais e alienígenas, lembrando o inesquecível *Space Invaders* que fez muito sucesso no passado, porém com a qualidade dos recursos gráficos e sonoros modernos.

# BIBLIOGRAFIA

ARRUDA, E. P. **Fundamentos para o Desenvolvimento de Jogos Digitais**. Porto Alegre: Bookman, 2014.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML. Guia do Usuário.** 2ª. ed. Rio de Janeiro: Elsevier, 2005.

CÁSSIO, É. **Desenvolva Jogos com HTML5 Canvas e JavaScript**. [S.l.]: Casa do Código, 2014.

DIXON MORAES, J. B. Técnicas para levantamento de Requisitos. **DevMedia**, 2009. Disponivel em: <https://www.devmedia.com.br/tecnicas-para-levantamento-de-requisitos/9151>. Acesso em: 01 jan. 2018.

FILHO, W. D. P. P. **Engenharia de Software Fundamentos, Métodos e Padrões**. 3ª. ed. [S.l.]: LTC, 2009.

G1. Conheça os 10 games sociais mais populares do Facebook. **G1**, 2011. Disponivel em: <http://g1.globo.com/tecnologia/noticia/2011/09/conheca-os-10-games-sociais-mais-populares-do-facebook.html>. Acesso em: 24 maio 2017.

G1. Por mês, mais de 250 milhões de usuários jogam no Facebook. **G1**, 2013. Disponivel em: <http://g1.globo.com/tecnologia/noticia/2013/03/por-mes-mais-de-250-milhoes-de-usuarios-jogam-no-facebook.html>. Acesso em: 24 maio 2017.

G1 RIO. Brasil é o 4º consumidor de games, mas mercado carece de mão de obra. **G1**, 2015. Disponivel em: <http://g1.globo.com/rio-de-janeiro/noticia/2015/11/brasil-e-o-4-consumidor-de-games-mas-mercado-carece-de-mao-de-obra.html>. Acesso em: 19 fev. 2017.

G1 RIO. Internet chega pela 1ª vez a mais de 50% das casas no Brasil, mostra IBGE. **G1**, 2016. Disponivel em: <http://g1.globo.com/tecnologia/noticia/2016/04/internet-chega-pela-1-vez-mais-de-50-das-casas-no-brasil-mostra-ibge.html>. Acesso em: 19 fev. 2017.

GIL, A. C. **Como Elaborar Projetos de Pesquisa**. 4ª. ed. São Paulo: Atlas, 2002.

GOMES, H. S. Jogos simples são os mais baixados para celular no Brasil em 2014. **G1**, 2014. Disponivel em: <http://g1.globo.com/tecnologia/games/noticia/2014/12/jogos-simples-sao-os-mais-baixados-para-celular-no-brasil-em-2014-veja.html>. Acesso em: 02 mar. 2017.

GONZALES CLUA, E. ; BITTENCOURT, J. R. Desenvolvimento de Jogos 3D: Concepção, Design e Programação, São Leopoldo, jul. 2005.

HIRAMA, K. **Engenharia de Software Qualidade e Produtividade com Tecnologia**. Rio Janeiro: Elsevier, 2011.

LARMAN, C. **Utilizando UML e padrões. Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo**. 3ª. ed. São Paulo: Bookman, 2007.

LECHETA, R. R. **Google Android - 3ª Edição:** Aprenda a criar aplicações para dispositivos móveis com o Android SDK. 3ª. ed. [S.l.]: Novatec, 2013.

LEMES, D. O.; TOMASELLI, F. C.; CAMAROTTI, S. A economia digital e o mercado de jogos moveis. **SBC**, Brasilia, November 2012.

MACHADO PINTO, N. ; CORONEL, D. A.; BRESOLIN, R. P. Análise Comparativa da Evolução das Vendas e do PIB. **Revista de Administração, Contabilidade e Sustentabilidade**, v. 3, n. Especial, p. 44-60, 15 set. 2013.

MDN. HTML5. **Mozilla Developer Network**, 2015. Disponivel em: <https://developer.mozilla.org/en-US/docs/Glossary/HTML5>. Acesso em: 17 set. 2016.

MDN. Introdução ao desenvolvimento de jogos para a Web. **Mozilla Developer Network**, 2017. Disponivel em: <https://developer.mozilla.org/pt-BR/docs/Games/Introduction>. Acesso em: 25 fev. 2018.

MDN. HTML. **Mozilla Developer Network**, 2018. Disponivel em: <https://developer.mozilla.org/en-US/docs/Web/HTML>. Acesso em: 17 set. 2016.

MDN. 2D collision detection. **Mozilla Developer Network**. Disponivel em: <https://developer.mozilla.org/en-US/docs/Games/Techniques/2D\_collision\_detection>. Acesso em: 2016 abr. 22.

MDN. Anatomia de um vídeo game. **Mozilla Developer Network**. Disponivel em: <https://developer.mozilla.org/pt-BR/docs/Games/Anatomy>. Acesso em: 17 set. 2016.

MDN. Audio for Web games. **Mozilla Developer Network**. Disponivel em: <https://developer.mozilla.org/en-US/docs/Games/Techniques/Audio\_for\_Web\_Games>. Acesso em: 22 abr. 2016.

MDN. Canvas API. **Mozilla Developer Network**. Disponivel em: <https://developer.mozilla.org/en-US/docs/Web/API/Canvas\_API>. Acesso em: 17 set. 2016.

MDN. Efficient animation for web games. **Mozilla Developer Network**. Disponivel em: <https://developer.mozilla.org/en-US/docs/Games/Techniques/Efficient\_animation\_for\_web\_games>. Acesso em: 22 abr. 2016.

MDN. Ferramentas para o desenvolvimento de jogos. **Mozilla Developer Network**. Disponivel em: <https://developer.mozilla.org/pt-BR/docs/Games/Tools>. Acesso em: 22 abr. 2016.

MDN. Herança e cadeia de protótipos (prototype chain). **Mozilla Developer Network**. Disponivel em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Inheritance\_and\_the\_prototype\_chain>. Acesso em: 17 set. 2016.

MDN. Introdução. **Mozilla Developer Network**. Disponivel em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Introduction>. Acesso em: 17 set. 2016.

MDN. Introdução ao HTML. **Mozilla Developer Network**. Disponivel em: <https://developer.mozilla.org/pt-BR/docs/Aprender/HTML/Introducao\_ao\_HTML>. Acesso em: 17 set. 2016.

MDN. Optimizing startup performance. **Mozilla Developer Network**. Disponivel em: <https://developer.mozilla.org/en-US/Apps/Fundamentals/Performance/Optimizing\_startup\_performance>. Acesso em: 22 abr. 2016.

MDN. Técnicas para desenvolvimento de jogos. **Mozilla Developer Network**. Disponivel em: <https://developer.mozilla.org/pt-BR/docs/Games/Techniques>. Acesso em: 17 set. 2016.

MILETTO, E. M.; BERTAGNOLLI, S. D. C. **Desenvolvimento de Software II.Introdução ao Desenvolvimento Web com HTML, CSS, Javascript e PHP**. Porto Alegre: Bookman, 2014.

NIERADKA, I. P. **Javascript + CSS + DOM Desenvolvimento Para Web**. [S.l.]: Novaterra, 2014.

OKUYAMA, F. Y.; MILETTO, E. M.; NICOLAO, M. **Desenvolvimento de Software I. Conceitos Basicos**. Porto Alegre: Bookman, 2014.

PRADANOV, C. C.; FREITAS, E. C. **Metodologia do Trabalho Científico:** Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico. 2ª. ed. Novo Hamburgo: Feevale, 2013.

PRESSMAN, R. R. **Engenharia de Software. Uma Abordagem Profissional**. 7ª. ed. São Paulo: McGraw Hill Brasil, 2009.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software. Uma Abordagem Profissional**. 8ª. ed. Porto Alegre: AMGH, 2016.

REDAÇÃO OLHAR DIGITAL. Giphy atinge mais de 100 milhões de usuários diários. **Olhar Digital**, 2016. Disponivel em: <https://olhardigital.com.br/noticia/giphy-atinge-mais-de-100-milhoes-de-usuarios-diarios/63433>. Acesso em: 19 fev. 2017.

REDAÇÃO VEJA. Em 2020, haverá mais pessoas no mundo com celulares que com eletricidade, água e carros. **Veja**, 2016. Disponivel em: <https://veja.abril.com.br/economia/em-2020-havera-mais-pessoas-no-mundo-com-celulares-que-com-eletricidade-agua-e-carros/>. Acesso em: 01 mar. 2017.

REMOALDO, P. **O Guia Pratico do Dreamweaver CS3 com PHP, JavaScript e Ajax**. 1ª. ed. [S.l.]: Centro Atlântico, 2008.

REZENDE, D. A. **Engenharia de Software e Sistemas da Informação**. 3ª. ed. Rio Janeiro: BRASPORT, 2005.

RONCOLATO, M. Games sociais; um mercado em alta. **Estadão**, 2011. Disponivel em: <https://link.estadao.com.br/noticias/geral,games-sociais-um-mercado-em-alta,10000039840>. Acesso em: 19 fev. 2017.

SANTOS, R. **Introdução à Programação Orientada a Objetos. Java**. 2ª. ed. Rio de Janeiro: Elsevier, 2013.

SILVA FILHO, A. M. **Introdução à Programação Orientada a Objetos com C++**. Rio de Janeiro: Elsevier, 2011.

SILVA, E. L.; MENEZES, E. M. **Metodologia da Pesquisa e Elaboração de Dissertação**. 4ª. ed. Florianópolis: [s.n.], 2005.

SILVEIRA FERREIRA, J. S. **A WEB SEMÂNTICA**. UNIVERSIDADE DO SUL DE SANTA CATARINA. Palhoça. 2010.

SOMMERVILLE, I. **Engenharia de Software**. 6ª. ed. [S.l.]: Addison Wesley Longman Limited , 2003.

SOUZA, C. S. et al. Projeto de Interfaces de Usuário. Perspectivas Cognitivas e Semióticas, Rio de Janeiro, Julho 1999. Disponivel em: <https://www.dimap.ufrn.br/~jair/piu/JAI\_Apostila.pdf>. Acesso em: 01 jan. 2018.

TRASVIÑA, F. Introdução ao JavaScript Orientado a Objeto. **Mozilla Developers Network (MDN)**, 2005. Disponivel em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Introduction\_to\_Object-Oriented\_JavaScript>. Acesso em: 16 set. 2016.

WAZLAWICK, R. S. **Engenharia de Software. Conceitos e Práticas**. [S.l.]: Elsevier Brasil, 2013.

# APÊNDICE A – Fichas de Requisito

Neste anexo estão presentes as fichas de requisitos geradas durante o processo de levantamento de requisitos.

# APÊNDICE B – ATA DAS REUNIÕES

# APÊNDICE C – CÓDIGO FONTE

# APÊNDICE D – DIAGRAMAS DE MODELAGEM

# APÊNDICE E – ENTREVISTA

1. Disponível em <<http://gopherwoodstudios.com/sandtrap/sand-trap.htm>> Acesso em: 19/02/2017 [↑](#footnote-ref-1)
2. Disponível em <<http://www.playmapscube.com/>> Acesso em: 19/02/2017 [↑](#footnote-ref-2)
3. Disponível em <http://cocos2d-x.org> Acesso em: 22/04/2016 [↑](#footnote-ref-3)
4. Disponível em <http://piqnt.com/stage.js/> Acesso em: 22/04/2016 [↑](#footnote-ref-4)
5. Disponível em <http://www.isogenicengine.com/> Acesso em: 22/04/2016 [↑](#footnote-ref-5)
6. Disponível em <https://www.sublimetext.com/> Acesso em: 01/06/2018 [↑](#footnote-ref-6)