

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA
E TECNOLOGIA DO RIO GRANDE DO SUL
CAMPUS BENTO GONÇALVES

LIMITAÇÕES DO HTML
NO DESENVOLVIMENTO DE JOGOS MULTIPLATAFORMA

JEAN CARLO MACHADO

Bento Gonçalves, dezembro de 2015.

JEAN CARLO MACHADO

LIMITAÇÕES DO HTML
NO DESENVOLVIMENTO DE JOGOS MULTIPLATAFORMA

Monografia apresentada junto ao Curso de Tecnologia em Análise e Desenvolvimento de Sistemas no Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - Campus Bento Gonçalves, como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Esp. Rafael Jaques

Bento Gonçalves, dezembro de 2015.

RESUMO

Jogos estão cada vez mais diversos, sociais e presentes no nosso dia a dia e a miríade de dispositivos que podem comportá-los oferece desafios e oportunidades. O HTML é uma ferramenta que possibilita a construção de jogos para múltiplas plataformas; entretanto, as tecnologias do HTML mudam constantemente e seu suporte é variado. Neste contexto, este trabalho busca estudar as limitações do HTML quando aplicado ao desenvolvimento de jogos multiplataforma. Para tal, foi elaborada uma revisão bibliográfica do assunto através de artigos científicos, livros e testes. E desenvolveu-se um protótipo de jogo de matemática, onde o usuário pode escolher a veracidade de data equação. Com a experiência adquirida e os dados coletados na revisão procedeu-se com a criação de uma lista de limitações presentes no atual estado do HTML.

Palavras-chave: Jogos, HTML, Limitações, Multiplataforma

ABSTRACT

Games are increasingly diverse, social and present in our every day lives. The great amount of devices that can handle this games offers challenges and opportunities. HTML is a set of tools that enables the construction of multiplatform games; nevertheless, the set of tools HTML is composed of changes rapidly and their support is miscellaneous. Given this situation, this work aims to study the limitations of HTML when applied to multiplatform game development. For that, we elaborated a bibliographic research and a game prototype was developed. With the expertise we got from it, and the data we collected, we composed a list of limitations that are present currently on HTML.

Palavras-chave: Games, HTML, Limitations, Multiplatform

LISTA DE FIGURAS

Figura 1:	Suíte HTML	24
Figura 2:	Os módulos do CSS	26
Figura 3:	Exemplo de Media Query	27
Figura 4:	Exemplos de media queries customizadas	27
Figura 5:	Exemplo de transição	28
Figura 6:	Exemplo de transformação	28
Figura 7:	Propostas do ECMA 7	32
Figura 8:	Exemplo de utilização de Web Animations	34
Figura 9:	Suporte das especificações do HTML nos navegadores	36
Figura 10:	Círculo em SVG.	39
Figura 11:	Canvas	40
Figura 12:	Comparação de codecs de áudio	44
Figura 13:	Exemplo de utilização da tag áudio	44
Figura 14:	Exemplo de utilização de vídeo	46
Figura 15:	Web Storage na prática	48
Figura 16:	Adicionando um cliente em IndexedDB.	49
Figura 17:	Exemplo de arquivo de manifesto offline	50
Figura 18:	Utilização dos eventos do teclado	51
Figura 19:	Objetos de um Gamepad	52
Figura 20:	Protótipo no PhoneGap Build	55
Figura 21:	Interface requerida	58
Figura 22:	Diagrama de classes simplificado	59
Figura 23:	Interface do jogo com equação sendo apresentada	61
Figura 24:	Interface do jogo com equação sendo apresentada no desktop (1440x900)	61
Figura 25:	Cálculo da pontuação do jogador	62
Figura 26:	Resultado de uma partida	63
Figura 27:	Configurações do jogo	64
Figura 28:	Contador em Canvas	65
Figura 29:	Exemplo de função imediatamente invocada.	68
Figura 30:	Função para testar tipos que funciona como o esperado.	72
Figura 31:	Teste de tela cheia	73

Figura 32:	Diagrama de classes completo	92
------------	--	----

SUMÁRIO

1	INTRODUÇÃO	11
1.1	PROBLEMA	12
1.2	OBJETIVOS	12
1.2.1	Objetivo Geral	12
1.2.2	Objetivos Específicos	12
1.3	JUSTIFICATIVA	13
1.4	METODOLOGIA	14
1.5	TRABALHOS RELACIONADOS	14
2	REVISÃO BIBLIOGRÁFICA	16
2.1	JOGOS DIGITAIS	16
2.1.1	Benefícios	16
2.1.2	Mecânica	17
2.1.3	Laço	17
2.2	JOGOS MULTIPLATAFORMA	18
2.2.1	Jogos Web	18
2.2.2	Desenvolvimento de Jogos Nativos	19
2.2.3	Jogos Híbridos	20
2.3	WEB	20
2.3.1	Open Web	21
2.4	HTML	22
2.4.1	DOM	24
2.5	CSS	25
2.5.1	Media Queries	26
2.5.2	Transições	27
2.5.3	Transformações 3D	28
2.5.4	CSS 4	29
2.6	JAVASCRIPT	29
2.6.1	JavaScript 7	31
2.6.2	Asm.js	32
2.6.3	Web Assembly	33
2.6.4	Web Animations	33

2.7	NAVEGADORES	34
2.8	ANDROID	36
2.9	DETECÇÃO DE RECURSOS	37
2.10	RENDERIZAÇÃO	38
2.10.1	SVG	38
2.10.2	Canvas	39
2.10.3	WebGL	40
2.10.4	WebVR	41
2.11	WEBCL	42
2.12	CODECS	42
2.13	ÁUDIO	44
2.13.1	Elemento Áudio	44
2.13.2	API de Áudio	45
2.14	VÍDEO	46
2.15	ARMAZENAMENTO	46
2.15.1	Web SQL	47
2.15.2	Web Storage	47
2.15.3	IndexedDB	48
2.16	WEB WORKERS	49
2.17	OFFLINE	49
2.18	ENTRADA DE COMANDOS	50
2.18.1	Gamepad	51
2.19	ORIENTAÇÃO	52
2.20	HTTP/2	52
2.21	DEPURAÇÃO	53
2.21.1	Source Maps	54
2.22	DISPONIBILIZAÇÃO DA APLICAÇÃO	55
3	PROJETO	57
3.1	MECÂNICA	57
3.2	REQUISITOS	58
3.2.1	Requisitos funcionais	58
3.2.2	Requisitos não funcionais	58
3.3	MODELAGEM	59
3.4	DESENVOLVIMENTO	59
3.4.1	Performance	66
3.5	OTIMIZAÇÕES PARA JOGOS	66
3.5.1	CSS	66
3.5.2	JavaScript	67

3.5.2.1	Modo estrito	67
3.5.2.2	Funções imediatamente invocadas	67
3.5.2.3	HTML	68
4	RESULTADOS	69
4.1	HTML	69
4.2	CSS	70
4.3	JAVASCRIPT	71
4.3.1	Sistema de tipos	72
4.3.2	Performance	72
4.3.3	Fullscreen	72
4.4	SVG	73
4.5	CANVAS	73
4.5.1	Performance	73
4.5.2	Integrações	73
4.6	WEBGL	74
4.7	ÁUDIO	75
4.8	VÍDEO	75
4.9	ARMAZENAMENTO	76
4.9.1	Web Storage	76
4.9.2	IndexedDB	77
4.10	OFFLINE	77
4.11	ORIENTAÇÃO	77
4.12	DETECÇÃO DE RECURSOS	77
4.13	DEBUG	78
4.14	ENTRADA DE COMANDOS	78
4.14.1	Gamepad	79
4.15	DISPONIBILIZAÇÃO	79
4.15.1	Monetização	79
4.16	OUTRAS LIMITAÇÕES	79
4.17	REVISÃO DAS LIMITAÇÕES	80
5	CONCLUSÕES	86
5.1	TRABALHOS FUTUROS	87
A	Diagrama de classes	92
B	Bibliotecas relevantes no desenvolvimento de jogos Web	93
B.1	–PREFIX-FREE	93
B.2	CROSSWALK	93

B.3	PHONEGAP	94
B.4	UNITY	94
B.5	TREE.JS	94
B.6	APPCELERATOR TITANIUM	95
B.7	JQUERY MOBILE	95
B.8	KENDO UI MOBILE	95
B.9	SENCHA TOUCH	95
C	Ambientes de desenvolvimento de jogos em HTML	96
C.1	PLAYCANVAS	96
C.2	INTEL HTML5 DEVELOPMENT ENVIRONMENT	96
D	Tecnologias de Compilação multiplataforma	97
D.1	GWT	97
D.2	LIBGDX	97
E	Sistemas de Building	98
E.1	GRUNT	98
E.2	GULP	98
F	Fontes do protótipo	99

1 INTRODUÇÃO

Desde a década de 90, jogos digitais se tornaram uma "forma dominante de recreação" (Davidsson; Peitz; Bjork, 2014). Devido a fatores como o barateamento da construção de hardware e sua crescente capacidade, a massificação dos dispositivos móveis inteligentes se tornou possível. Os computadores deixaram de ser apenas ferramentas científicas e de negócios, tornando-se plataformas de diversão. Atualmente jogos são utilizados em uma vasta gama de dispositivos (Pirttiäho, 2014, p. 6). Janiszewski (2014) afirma que mais de 80% do tempo total gasto utilizando dispositivos móveis é na utilização de aplicativos e 32% deste tempo é para jogar videogames.

Entretanto, devido a variedade de dispositivos, é difícil para os criadores de jogos alcançarem todos os jogadores possivelmente interessados em suas criações. Hasan et al. (2012) afirma que a maior dificuldade em capturar uma base de usuários é que o mercado de dispositivos móveis é muito fragmentado e não existe uma única plataforma popular. Esta característica força os criadores de jogos a suportarem diversas plataformas. Krill (2013) afirma que 81% dos aplicativos mobile rodam em pelo menos dois sistemas operacionais. Para atingir múltiplas plataformas os desenvolvedores utilizam de variadas estratégias, sendo uma delas o HTML5.

Desde o lançamento da versão 5, o HTML deixou de ser apenas uma plataforma para visualização de documentos na Internet e vem conquistando sua posição como uma forma de desenvolver jogos para múltiplas plataformas. Isso se dá pelas características de multimídia adicionadas nesta versão do HTML. E pelo suporte horizontal a múltiplas plataformas que o HTML provê (Hasan et al., 2012). Além destas características positivas, outros benefícios são observáveis na construção de jogos em HTML.

Muito pouco investimento é necessário para começar a desenvolver jogos utilizando as tecnologias da Web (Kuryanovich et al., 2014). Desenvolvedores de sites podem reaproveitar o conhecimento direcionando-o para o desenvolvimento de jogos. O interesse por parte dos desenvolvedores também é grande, Krill (2013) cita que cerca de 59% dos desenvolvedores estão muito interessados em desenvolver aplicativos em HTML5. As vantagens financeiras do desenvolvimento de aplicações multiplataforma em HTML também são substanciais. O tempo de desenvolvimento de uma aplicação em HTML5 é 67% menor que aplicações nativas (Hasan et al., 2012, p. 460).

Contudo, apesar da utilização da Web ser gigantesca, criar jogos dinâmicos e de tempo real não é seu primeiro objetivo (Kuryanovich et al., 2014). O processo de desenvolvimento de aplicações HTML5 está em constante fluxo de aperfeiçoamento e novos métodos, técnicas, e

ferramentas estão aparecendo todo o tempo (Barnett, 2014).

Neste contexto vê-se a necessidade de uma revisão das tecnologias do HTML para jogos, suas fraquezas e especialidades. Este trabalho propõe analisar as limitações do HTML5 relativo a construção de jogos multiplataforma através de revisão bibliográfica e da criação de um protótipo de jogo multiplataforma. O protótipo escolhido foi um jogo de matemática simples onde o usuário pode escolher se uma questão matemática, e seu resultado dado, estão corretos. Com as informações coletadas, de maneira prática e teórica, foram registradas as limitações e uma análise sobre elas foi efetuada.

1.1 PROBLEMA

A carência de definições concretas sobre a viabilidade da atual versão do HTML, quando aplicado ao desenvolvimento de jogos, e o senso comum, acostumado com soluções nativas, acabam por praticamente monopolizar a construção de jogos nativos.

Neste contexto este trabalho busca responder a seguinte pergunta: quais os reais problemas e limitações comuns no desenvolvimento de jogos multiplataforma em HTML5?

1.2 OBJETIVOS

Os objetivos gerais e específicos do trabalho serão descritos a seguir.

1.2.1 Objetivo Geral

Identificar possíveis limitações no processo de criação de jogos multiplataforma oriundas do atual estado de definição e implementação do HTML5.

Não é objetivo deste trabalho comparar o HTML com outras tecnologias de desenvolvimento de jogos, como Flash Player, Silverlight ou alternativas Desktop. Ou então demonstrar os pontos fortes do HTML5, apesar de haver revisão das tecnologias, o artefato final deste trabalho são as limitações do HTML.

1.2.2 Objetivos Específicos

O primeiro objetivo é estudar e resumir as tecnologias citadas abaixo:

- Tecnologias de renderização: Canvas, SVG, WebGL;
- Tecnologias de entrada: Eventos DOM, Gamepad;
- Armazenamento: IndexedDB, WebStorage;
- Offline: Manifesto de Cache
- Tecnologias multimídia: Áudio, Vídeo;

- Tecnologias de animações: Web Animations, CSS;
- Tecnologias base da Web: HTML, CSS, JavaScript, Web Assembly;
- Formas de depuração;
- Navegadores e plataformas de Games;
- Formas de disponibilizar jogos: PhoneGap.

Estas foram selecionadas baseando-se em recomendações da literatura e mercado. Por exemplo, Vahatupa (2014) cita que vídeo, áudio, drag-and-drop, funcionalidades de gráficos e armazenamento offline são aspectos importantes do HTML5 para o desenvolvimento de jogos. Já o site da Mozilla ¹ contém uma lista de tecnologias relevantes ao desenvolvimento de jogos na Web as quais são fortemente relacionadas aos itens acima mencionados.

Outro objetivo é a criação de um protótipo para melhor compreensão das tecnologias da Web e detecção de suas limitações. O protótipo deve ser criado sem utilização de bibliotecas ou frameworks de terceiros para eliminar a possibilidade de limitações do HTML serem tratadas ou ocultas por estas tecnologias.

Definiu-se que o protótipo deve funcionar nas plataformas Android 5 e nos navegadores Desktop Google Chrome e Firefox em suas últimas versões (47 e 43 respectivamente), para realizar a característica multiplataforma do trabalho. Optamos por Android, e não IOS, pois o primeiro contém a vasta maioria do mercado de dispositivos inteligentes, e pelo fato do autor já possuir maior experiência na já mencionada plataforma.

O objetivo final é a criação de uma lista de limitações relacionadas aos assuntos elencados que influenciem a construção de jogos.

1.3 JUSTIFICATIVA

Tendo em vista que este trabalho busca mapear possíveis problemas do desenvolvimento multiplataforma em HTML, ele serve para apoiar e justificar decisões relativas ao desenvolvimento de jogos multiplataforma.

Muitas pessoas no mercado de software são da opinião de que o desenvolvimento nativo para jogos é a melhor opção, enquanto outras formas de aplicativos são mais adequados para a Web Powell; Li (2013, p. 21). Este trabalho, por indicar problemas concretos do desenvolvimento Web, serve para verificar essa opinião.

Grande parte dos desenvolvedores estão familiarizados com as tecnologias da Web ou apontam interesse na tecnologia. A revisão tecnológica que este trabalho contém pode servir como um indicativo de sobre o que é preciso estudar para começar a desenvolver jogos na Web.

¹O site mencionado consta neste endereço <https://developer.Mozilla.org/en-US/docs/Games/Introduction>)

1.4 METODOLOGIA

O primeiro passo consiste na definição das plataformas alvo do trabalho. Estas são relevantes mercadologicamente ao desenvolvimento de jogos em HTML5.

Segue-se com a elaboração de uma lista com os recursos relevantes aos jogos que sofrem ou são comumente ligados a limitações multiplataforma. Segue-se uma pesquisa para aprofundar teoricamente cada um dos recursos, possivelmente elegendo novos.

Com um embasamento teórico substancial, o próximo passo é a criação do protótipo de um jogo multiplataforma que utilize grande parte dos recursos analisados.

Com o protótipo concebido, o passo que segue é a enumeração e descrição das limitações detectadas no processo de pesquisa e desenvolvimento. O detalhamento deve buscar responder perguntas como:

- Quais limitações são comuns no desenvolvimento de jogos em HTML5?
- Sob quais circunstâncias?
- Em quais plataformas?
- Quais limitações podem ser contornadas? Como?

1.5 TRABALHOS RELACIONADOS

Barnett (2014) elaborou uma revisão de algumas tecnologias do HTML5 através da construção de um jogo. O autor foca muito nos aspectos de criação de jogos e no feedback do desenvolvimento. Diferentemente deste trabalho, que foca nas limitações do HTML5. Em outras palavras seu escopo é mais genérico que o deste trabalho.

Powell; Li (2013) realizou uma pesquisa através de questionário e protótipo sobre a viabilidade de aplicativos em HTML5, concluindo que no geral desenvolvimento de aplicativos em HTML5 são opções viáveis e lucrativas. Seu trabalho difere a este por não focar no contexto dos jogos, não observando muitas das nuances e necessidades específicas para o desenvolvimento desta classe de aplicações. Outra diferença substancial é que o autor foca apenas na viabilidade não ressaltando as limitações da plataforma.

Pirttiahho (2014) revisa algumas tecnologias da Web e constrói um jogo protótipo para aprender um framework que possibilita a construção de jogos multiplataforma. Não obstante, o autor foca em um framework de compilação múltipla, não usando diretamente as tecnologias da Web. O autor também não foca na experiência do desenvolvimento ou em coletar limitações como este trabalho se propõe.

Morony (2013) estuda a viabilidade de aplicações comerciais multiplataforma em HTML5 através da construção de um aplicativo comercial em Senchu Touch. É construída uma lista de recursos interessantes no desenvolvimento comercial de aplicações e cada um destes recursos

é revisado depois do desenvolvimento, assemelhando-se muito a metodologia deste trabalho. Em seu estudo o autor utiliza uma ferramenta de desenvolvimento em C# que compila nativamente o que se distancia da proposta deste trabalho de avaliar as limitações com a construção de um protótipo diretamente em HTML. O autor também não se foca em tecnologias dos jogos, outrossim, em aplicações genéricas.

2 REVISÃO BIBLIOGRÁFICA

2.1 Jogos Digitais

Segundo Lemes (2009), jogo digital ou videogame constitui-se em uma atividade lúdica composta por uma série de ações e decisões, limitadas por regras e pelo universo do game, que resultam em uma condição final. Apesar da definição clara, uma taxonomia dos jogos não é tarefa trivial. Pela diversidade em itens e gêneros e a vasta quantidade de dimensões que os videogames se encontram, uma categorização dos jogos contemporâneos é extremamente difícil de desenvolver (muitos já tentaram) (Granic; Lobel; Engels, 2014, p. 60).

Para corroborar com essa complexidade, Granic; Lobel; Engels (2014) afirmam que a natureza dos jogos tem mudado drasticamente na última década, tornando-se cada vez mais complexos, diversos, realísticos e sociais em sua natureza. No contexto Web, com as inovações nas tecnologias relacionadas ao HTML, novos gêneros de jogos podem ser explorados. Cada gênero acompanha um conjunto de desafios específicos. Jogos de FPS (tiro em primeira pessoa) requerem menor latência de rede, já jogos de RPG (interpretação de personagens) podem requerer vastas quantidades de cache (Kuryanovich et al., 2014).

2.1.1 Benefícios

Em conjunto com as tecnologias, Granic; Lobel; Engels (2014) afirma que: "um grupo emergente de pesquisas sobre os benefícios dos jogos vem se desenvolvendo". Apesar de a quantidade de estudos sobre os males dos jogos ser muito maior do que os estudos sobre seus benefícios, a quantidade de benefícios já correlacionados aos jogos é substancial. Os autores demonstram que videogames melhoram as funções cognitivas, as capacidades criativas, e motivam uma visão positiva diante a falha, fator que correlaciona-se com melhor desempenho acadêmico.

Benefícios em habilidades práticas também são observados em usuários de jogos. Os autores afirmam que jogadores de jogos de tiro demonstram maior alocação de atenção, maior resolução espacial no processamento visual e melhores habilidades de rotação. Estes aspectos positivos muitas vezes não recebem a atenção devida. Habilidades espaciais derivadas de jogos de tiro, comercialmente disponíveis, são comparáveis aos efeitos de um curso universitário que busca melhorar as mesmas habilidades, segundo os autores. Estas habilidades, derivadas dos jogos, são centrais para muitas áreas de interesse humano. Os autores por fim afirmam que habilidades espaciais estão diretamente relacionadas com o sucesso em ciência, tecnologia,

engenharia e matemática.

Outro ponto importante dos jogos é seu aspecto social. Apesar de a mídia ter criado uma perspectiva negativa sobre jogos, especialmente os violentos, a realidade é mais complexa do que se pensa. Jogadores de jogos violentos, cuja jogabilidade encoraje a cooperatividade, são mais prováveis de exibir comportamento altruísta fora do contexto dos jogos, do que jogadores de jogos não violentos (Granic; Lobel; Engels, 2014).

Além dos aspectos benéficos, adiante serão abordados alguns aspectos técnicos e gerenciais dos jogos.

2.1.2 Mecânica

Kuryanovich et al. (2014) ressalta a importância do planejamento antes do desenvolvimento. Ao criar jogos deve-se planejar o que se pretende atingir e como chegar lá antes de se escrever qualquer código; definições quanto à mecânica são um passo vital neste planejamento.

A mecânica é composta pelas regras do jogo. Quais as ações disponíveis aos usuários e seu funcionamento são fortemente influenciadas pela categoria do jogo em questão. Dedicar-se na elaboração de uma mecânica é tarefa quintessencial para a construção de um jogo de sucesso. A mecânica não pode ser simplesmente boa, nos melhores jogos ela é intuitiva. O processo de o jogador entender por si a mecânica do jogo é um componente vital para a sua satisfação. Se as regras do jogo não forem assimiladas quase que instantaneamente, muitas pessoas vão perder o interesse e desistir rapidamente (Barnett, 2014).

Ainda sobre a importância da mecânica, Kuryanovich et al. (2014) afirma que se os gráficos e áudio são espetaculares, mas a jogabilidade é chata, o jogador vai parar de jogar. A substância do jogo é sua mecânica, então não invista muito em visual ao menos que isso desempenhe um papel essencial no jogo. Os desenvolvedores tem que evitar fazer o jogo para eles mesmos. Falta de crítica antes do desenvolvimento também tende a gerar jogos ruins. Bons jogos são aqueles que ao menos suprem as expectativas dos usuário. Lemes (2009) aponta alguns fatores procurados pelos usuários de jogos: desafio, socializar, experiência solitária, respeito e fantasia. Jogos que conseguem integrar o maior número destes aspectos em sua mecânica serão os mais apreciados. A elaboração da mecânica em jogos desenvolvidos profissionalmente pode ser integrada dentro de um processo de engenharia de software.

Depois dos preparativos efetuados, pode-se começar a construção do jogo.

2.1.3 Laço

Jogos digitais geralmente operam através um laço que executa uma série de tarefas a cada iteração, construindo a ilusão de um mundo animado (Prado, 2012, p. 31). Da perspectiva da programação, a parte principal de um jogo é o laço onde o jogo é executado (Pirttiaho, 2014, p. 17). A cada iteração do laço o processamento de entrada de dados do usuário e alterações da cena tem de ser computados, tornando a otimização deste processo importantíssima para que o

jogo não se torne lento¹.

Outra decisão técnica importante na hora de desenvolver jogos é a seleção das plataformas alvo mais adequadas. Neste trabalho o foco será em HTML para desktops e dispositivos móveis inteligentes. Não obstante, na seção a seguir serão descritas as plataformas que os desenvolvedores podem selecionar ao desenvolver jogos, juntamente com seus benefícios e fraquezas.

2.2 Jogos Multiplataforma

Jogos multiplataforma são jogos que rodam em mais de uma plataforma. Cada plataforma contém sua própria API (*Application Programming Interface*), sendo que se um aplicativo foi criado para uma plataforma ele não vai poder ser utilizado em outra pois as APIs são diferentes (Pirttiäho, 2014). Além da API, dispositivos de múltiplas plataformas variam em seus recursos, capacidades e qualidades. Devido a essas características, desenvolvedores de jogos que almejem múltiplas plataformas, deparam-se com uma nova gama de oportunidades e desafios.

Um destes desafios é fornecer *feedback* suficiente para o jogador, pois muitas vezes o dispositivo é limitado em proporções, som e tela. Devido a estes requerimentos, tendências como Web design responsivo (RWD) emergiram. Requerendo que os desenvolvedores busquem criar interfaces cada vez mais fluídas e intuitivas. As tecnologias da Web também tiveram que acompanhar a mudança. CSS3 media queries e tamanhos relativos, tratadas adiante, são exemplos de tecnologias desenvolvidas com o foco em multiplataforma.

Outro desafio multiplataforma é suportar os vários ecossistemas de software com tecnologias diferenciadas e versões de software divergentes. Para que um jogo multiplataforma tenha sucesso é necessário definir com cautela suas tecnologias. Kuryanovich et al. (2014) afirma que: o estágio mais complicado e crucial do desenvolvimento de jogo é a escolha das tecnologias utilizadas. Designers de jogos tem as seguintes possibilidades quando em face de desenvolver um jogo multiplataforma: criar um jogo Web, um jogo híbrido ou nativo. As opções serão descritas a seguir.

2.2.1 Jogos Web

Um jogo Web utiliza o HTML e ferramentas correlacionadas para sua construção e disponibilização. No entanto, o objetivo primário da Web nunca foi o desenvolvimento de jogos. Por muito tempo, os títulos famosos de jogos da Web residiam em jogos como *Traviam*, desprovidos de animações, compostos basicamente por formulários, imagens e textos. Durante esse período, o interesse em jogos Web residia principalmente na casualidade, flexibilidade, e no fator social.

¹Na seção 3.5 foram descritas as opções da Web na hora de construir um laço de jogo.

Mais recentemente é que a Web começou a ser vista como de fato um ambiente com interatividade para criação de jogos dos mais variados gêneros. Publicar jogos baseados em texto é uma atividade cada vez mais rara, podendo-se concluir que interface gráfica se tornou uma funcionalidade mandatória (Vahatupa, 2014). Jogos como BrowserQuest, Angry Birds, entre outros títulos expandiram os conceitos do que é possível se fazer utilizando as ferramentas da Web. Segundo Prado (2012, p. 28) um bom exemplo que tem alcançado bastante sucesso entre o público são os jogos adicionados no logotipo do Google, chamados doodles.

Quando comparados a outras abordagens de criar jogos, jogos na Web contém vários aspectos positivos. Talvez o mais reconhecível é o fato de com uma única base de código poder suprir uma gama praticamente inesgotável de dispositivos. Comparável a base de clientes está a quantidade de desenvolvedores Web, os quais podem reaproveitar grande parcela do conhecimento adquirido através do desenvolvimento de páginas Web na criação de jogos. Sua distribuição também é superior ao estilo convencional das aplicações desktop (Vahatupa, 2014). Por serem criados a partir das tecnologias da Web, jogos Web se beneficiam de uma arquitetura construída para um ambiente em rede, sendo relativamente mais fácil criar experiências sociais.

A performance é um ponto negativo da Web, é difícil chegar a performance comparável a abordagem nativa. Contudo, esse problema é cada vez menor, visto que o hardware dos dispositivos são cada vez mais potentes e as tecnologias de software também avançam substancialmente em performance. Outro problema da abordagem Web é que existem inconsistências nas implementações das especificações que formam a Web o que leva a comportamento inesperados em alguns casos, sendo necessário desenvolver regras específicas para dispositivos e versões de navegadores. Outra fraqueza notável da Web é que nem todas as funcionalidades dos dispositivos estão especificados com as tecnologias da Web e muitas vezes os recursos dos dispositivos ficam subutilizados.

Além dos jogos web, há a possibilidade de criar jogos nativos e híbridos.

2.2.2 Desenvolvimento de Jogos Nativos

Uma aplicação nativa é uma aplicação que foi desenvolvida para ser utilizada em uma plataforma ou dispositivo específico (Powell; Li, 2013, p. 7). Aplicativos nativos tendem a oferecer uma experiência mais próxima com a do resto do sistema operacional a qual estão rodando. Potencialmente, softwares nativos são mais rápidos que suas alternativas da Web, visto que interagem com o dispositivo através do sistema operacional. Diferentemente dos jogos Web, que necessitam que o navegador interaja com o sistema operacional, para por sua vez interagir com o dispositivo. Por terem acesso total ao dispositivo, aplicativos nativos podem aproveitar o hardware da melhor forma possível e oferecer ao usuário a melhor experiência possível (Powell; Li, 2013, p. 7).

Um dos pontos negativos da abordagem nativa, é que tende a ser mais caro que a alternativa da Web, visto que é necessário duplicar funcionalidades em sistemas distintos e possivelmente manter um profissional por ambiente suportado.

A alternativa híbrida fica em meio ao desenvolvimento nativo e Web e será descrita abaixo.

2.2.3 Jogos Híbridos

A alternativa híbrida é uma tentativa de beneficiar-se das melhores características da abordagem nativa e o melhor do desenvolvimento Web. Muitas vezes desenvolve-se aplicações híbridas utilizando as tecnologias da Web só que ao invés de disponibilizar a aplicação através de um navegador a aplicação Web é instalada como um aplicativo normal. A aplicação roda em uma WebView que é um componente do sistema operacional capaz de rodar as tecnologias da Web. Desta forma o aplicativo Web conversa diretamente com o sistema operacional, não necessitando da intervenção de um software terceiro para mediar a interação.

Outra possibilidade híbrida é escrever o software em uma linguagem e gerar binários para as plataformas alvo. Utilizando o Xamarin é possível desenvolver em C# e compilar para diversas plataformas nativamente. Através dessa abordagem é possível beneficiar-se de um aplicativo nativo e eliminar grande parte da duplicação geralmente imposta ².

Visto que a estratégia híbrida geralmente tem acesso ao sistema operacional, é possível criar APIs para acessar recursos não sempre disponíveis para a plataforma Web. Soluções como o PhoneGap adotam essa estratégia para habilitar controle refinado sobre os recursos dos dispositivos através de APIs JavaScript. Outro benefício da estratégia híbrida em relação à Web é que ela permite empacotar as aplicações com uma experiência exata a dos softwares nativos. Tornando imperceptível para o usuário final a diferença entre um aplicativo híbrido e um nativo.

Todavia, segundo Powell; Li (2013, p. 8), a diferença entre o que é possível com a estratégia híbrida e a Web está diminuindo devido ao grande esforço da comunidade Web para prover novas especificações.

2.3 Web

A Web (World Wide Web) é uma plataforma de compartilhamento de documentos que funciona sobre a Internet. Criada por Tim Berners-Lee em 1989, enquanto trabalhava para o CERN, sendo que hoje desempenha papel vital na proliferação de informações da civilização moderna. Em 1990 Berners-Lee havia criado todas as ferramentas necessárias para viabilizar a Web: o primeiro navegador, o primeiro servidor Web e a primeira página, que era utilizada para descrever o projeto (Wikipedia, 2015c).

Os documentos na Web são identificados por URLs (*uniform resource locator*), sendo que cada documento pode referenciar outros, criando a "teia" que o nome Web se refere. Na época de sua criação tanto Hipertexto quanto a internet já eram coisas comuns. A grande inovação da Web é que Tim ligou ambos criando uma experiência totalmente revolucionária.

²Os frameworks multiplataforma dos apêndices contém tecnologias similares ao Xamarin como o Titanium

2.3.1 Open Web

A OWP (*Open Web Platform*) é uma coleção de tecnologias livres, amplamente utilizadas e padronizadas que viabilizam a Web. Quando uma tecnologia se torna amplamente popular, através da adoção de grandes empresas e desenvolvedores, ela se torna candidata a adoção pela OWP. Os benefícios de utilizar tecnologias da OWP são vários. Neumann; Winter (2001, p. 3) cita que as tecnologias padronizadas tem um maior ciclo de vida e são mais fáceis de dar manutenção. Da mesma maneira, as tecnologias da Web são benéficas devido a sua grande adoção, permitindo que aplicações baseadas nelas tenham impacto na maior quantidade de clientes possível.

Não obstante, mais do que tecnologias a OWP é um conjunto de filosofias as quais a Web se fundamenta (Neuberg, 2008). Entre outras, a Open Web busca transparência, imparcialidade nos processos de criação e padronização de novas tecnologias, respeito ao usuário, retro compatibilidade com as especificações anteriores e consenso entre o mercado e o meio acadêmico, nunca um distanciando-se muito do outro ³.

Várias pessoas, empresas e comunidades estão interessadas neste processo, cada qual com seu próprio conjunto de ideias sobre como a Web deveria funcionar. E para que a crescente quantidade de dispositivos possa acessar a riqueza que o HTML5 permite, padrões precisam ser definidos (Powell; Li, 2013, p. 5).

A W3C é a organização responsável por boa parte das especificações da Web como: HTML (em conjunto com a WHATWG), CSS, entre outras⁴. Outros grupos detém responsabilidade por outras tecnologias da OWP, como a ECMA, responsável pelo JavaScript; ou Kronos, responsável pelo WebGL. Na W3C o processo de desenvolvimento de especificações consiste na elaboração de rascunhos (*working drafts*), criados por grupos de trabalhos (*working groups*) de especialistas no assunto, que passam por vários passos de revisão até se tornarem recomendações. As recomendações podem ser implementadas com segurança de que a especificação não mudará substancialmente.

Apesar do processo da W3C ser rigoroso, está longe de perfeito. A especificação final do HTML4 contava com quatro erros publicados via errata (Pilgrim, 2010). Mesmo assim o cenário é animador, Kuryanovich et al. (2014) cita que as tecnologias da Open Web tem evoluído desde os princípios da internet e já provaram sua robustez e estabilidade enquanto outras tecnologias crescem e morrem ao redor dela.

A tecnologia chave que inaugurou e alavancou este processo é o HTML.

³Mais informações sobre a Open Web podem ser encontradas no seguinte endereço <http://tantek.com/2010/281/b1/what-is-the-open-web>

⁴Uma lista completa das especificações mantidas pela W3C pode ser encontrada em: <http://www.w3.org/TR/>

2.4 HTML

HTML (*Hyper Text Markup Language*) é uma linguagem de marcação que define a estrutura semântica do conteúdo das páginas da Web. É a tecnologia base para a criação de páginas Web e aplicativos online. A parte denominada: "*Hyper Text*", refere-se a links que conectam páginas HTML umas as outras, fazendo a Web como conhecemos hoje (MDN, 2016).

A última versão do HTML é o HTML5, iniciado pela WHATWG e posteriormente desenvolvido em conjunto com a W3C. Seu rascunho foi proposto em 2008 e ratificado em 2014. Após 2011, a última chamada de revisão do HTML5, a WHATWG decidiu renomear o HTML5 para HTML (Hickson, 2011). Todavia, o termo HTML5 permanece em utilização pela W3C.

Além da nomenclatura, existem pequenas diferenças nas especificações da WHATWG e W3C. A W3C vê a especificação do HTML5 como algo fechado, inclusive já iniciou o desenvolvimento do HTML 5.1. Já a WHATWG vê o HTML5 como uma especificação viva. A postura da W3C tende a criar uma especificação estável. Já a postura da WHATWG reflete a realidade dos navegadores: que nunca implementam uma versão do HTML completamente. A Mozilla utiliza a especificação da WHATWG no desenvolvimento do Firefox e recomenda a da W3C para sistemas que requeiram maior estabilidade. Neste trabalho optamos pela nomenclatura da WHATWG, utilizamos o termo HTML em detrimento a HTML5, sempre que semanticamente viável.

HTML foi especificado baseando-se no padrão SGML (*Standard Generalized Markup Language*). Alguns benefícios do SGML são:

- Documentos declaram estrutura, diferentemente de aparência, possibilitando otimizações nos ambientes de uso (tamanho de tela, etc);
- São portáteis devido a definição de tipo de documento (*document type declaration*).

Apesar de o SGML especificar a não definição de aparência, os criadores de navegadores constantemente introduziam elementos de apresentação como o piscar, itálico, e negrito, que eventualmente acabavam por serem incluídos na especificação. Foi somente nas últimas versões que elementos de apresentação voltaram a ser proibidos reforçando as propostas chave do HTML como uma linguagem de conteúdo semântico, incentivando a utilização de outras tecnologias como o CSS para responder as demandas de apresentação.

Além do HTML, existe o XHTML, que é uma iniciativa de utilização de XML nas páginas da web. O XML é um padrão mais rigoroso que SGML e resulta em páginas sem problemas de sintaxe e tipografia. Alguns estimam que 99% das páginas HTML de hoje contenham ao menos um erro de estrutura (Pilgrim, 2010). Uma das maiores vantagens do XML é que sistemas sem erros de sintaxe podem ser facilmente interpretados por outras tecnologias como sistemas de indexação, buscadores, etc.

Para transformar o HTML em algo visível, os navegadores utilizam motores de renderização. O primeiro passo efetuado por esses sistemas é decodificar o documento HTML para sua representação em memória. Este processo dá-se através da análise (*parsing*) e posterior tokenização, que é a separação do HTML em palavras chave que o interpretador pode utilizar. Diferentemente do XHTML, HTML não pode ser decodificado através de tokenização tradicional. Isso deve-se ao fato do HTML ser amigável ao programador, aceitando erros de sintaxe, dependente de contexto, buscando entregar a melhor aproximação possível. Segundo Garisel; Irish (2011) essa é a maior razão do HTML ser tão popular - ele perdoa os erros e torna a vida dos autores da Web mais fácil. Esta característica deu origem a uma especificação para renderizar HTML (*HTML parser*).

Antes do HTML5 várias versões foram propostas algumas radicais em seus preceitos. O XHTML 2.0, por exemplo, quebrava com toda a compatibilidade das versões anteriores e acabou por sendo descontinuado. Outrossim, a maioria das versões HTML de grande sucesso foram versões de retrospectiva (*retro-specs*). Versões que não tentavam idealizar a linguagem, buscando alinhar-se com os requerimentos do mercado (Pilgrim, 2010). Não obstante, a ideia que a melhor forma de ajustar o HTML é substituindo ele por outra coisa ainda aparece de tempos em tempos (Pilgrim, 2010).

Uma página HTML consiste em elementos que podem ter seu comportamento alterado através de atributos. Um elemento é o abrir e fechar de uma tag e todo o conteúdo que dentro dela reside (Duckett, 2011, p. 10–11). Na sua versão inicial, o HTML contava com 18 elementos; atualmente existem aproximadamente 100 (Pilgrim, 2010). Entretanto, foi no HTML5 que a maior parte dos elementos que viabilizam a construção de jogos foram adicionados.

Uma das características do HTML que o torna tão popular é seu interesse em manter a retrocompatibilidade. Interpretadores HTML atingem isso ignorando os elementos que não conhecem, tratando seu vocabulário exclusivamente. Esse mecanismo permite que os desenvolvedores incluam marcação de reserva dentro dos elementos que podem não ser suportados.

Além da convencional linguagem de marcação, HTML é muitas vezes interpretado como um conceito guarda chuva para designar as tecnologias da Web. Segundo Pilgrim (2010) algumas dessas tecnologias (como geolocalização) estão em especificações separadas mas são tratadas como HTML5 também. Outras tecnologias foram removidas do HTML5 estritamente falando, mas são tratados como HTML5 (como a API de armazenamento de dados).



Figura 1: Suíte HTML

Fonte: <http://64vision.com/HTML5-what-is-it>

Uma tecnologia fortemente entrelaçada com o HTML é o DOM. Tendo uma relação próxima de um para um com a marcação (Garisel; Irish, 2011). DOM permite a interação entre documentos HTML e as demais tecnologias da Web de uma forma fácil e padronizada.

2.4.1 DOM

O modelo de documento de objetos (*Document Object Model*) é a representação em memória de uma árvore de elementos HTML. Esta representação é definida por um conjunto de objetos, unicamente identificados e dispostos em forma de grafo, que busca facilitar a manipulação de elementos através de JavaScript.

A primeira versão do DOM, DOM nível zero, foi especificada no HTML 4 e permitia manipulação parcial dos elementos. Foi somente com a especificação do JavaScript em 1998 que o DOM nível 1 foi criado, permitindo a manipulação de qualquer elemento. DOM nível 2 e 3 seguiram com melhorias nas consultas aos elementos e CSS.

A API de seletores (*querySelector*) do DOM permite alto nível de precisão e performance para buscar elementos. Os métodos *querySelector* e *querySelectorAll* buscam aproximar-se do funcionamento do jQuery, habilitando filtrar por classes, elementos, identificadores ou uma junção destes.

DOM também conta com uma API de eventos que possibilita que, através de JavaScript, se saiba quando algum evento interessante aconteceu. Cada evento é representado por um objeto baseado na interface *Event* e pode ter campos e funções adicionais para prover maior quantidade de informações (DevDocs, 2015). Os eventos podem ser criados pelo usuário ou serem lançados pelo navegador, possibilitando uma manipulação consistente dos mais variados aspectos de uma aplicação. Manipulação de entrada de comandos e muitas outras APIs do HTML, como o IndexedDB, se dão através de eventos DOM tornando o assunto relevante aos jogos ⁵.

⁵O site http://devdocs.io/dom_events/ contém uma lista dos eventos lançados automaticamente pelos navegadores

Fortemente entrelaçado com o HTML e o DOM está o CSS que possibilita customizar a apresentação do markup habilitando experiências muito mais ricas do que conteúdo bruto.

2.5 CSS

CSS (*Cascading Style Sheets*) é uma linguagem de folhas de estilo criada por Håkon Wium Lie em 1994 com intuito de definir a apresentação de páginas HTML. CSS, juntamente com JavaScript e HTML, compõem as tecnologias centrais no desenvolvimento Web tornando-se parte da OWP; sua especificação é atualmente mantida pela W3C.

O termo *Cascading* refere-se ao fato de regras serem herdadas pelos filhos de um elemento, eliminando grande parcela de duplicação antes necessária para estilizar uma página. Segundo Kuryanovich et al. (2014) pode-se expressar regras gerais que são "cascadeadas" para muitos elementos, e então sobrescrever os elementos específicos conforme a necessidade.

Segundo Lie (2005, p. 23–24):

CSS possibilita a ligação tardia (*late biding*) com páginas HTML. Essa característica é atrativa para os publicadores por dois motivos. Primeiramente pois permite o mesmo estilo em várias publicações, segundo pois os publicadores podem focar-se no conteúdo ao invés de se preocuparem com detalhes de apresentação.

Esta ligação tardia permitiu diferenciação entre apresentação e estrutura, sendo neste caso o CSS responsável pela apresentação. Esta característica é uma das ideias pioneiras do SGML, motivo que tornou a utilização do CSS tão conveniente para o desenvolvimento Web. Antes do CSS era impossível ter estilos diferenciados para diferentes tipos de dispositivos, limitando a aplicabilidade dos documentos. Com CSS também tornou-se possível que o usuário declare suas próprias folhas de estilo, um recurso importante para acessibilidade.

Estruturalmente falando, CSS é formado por um conjunto de regras, dentro de uma tag HTML denominada *style*, que são agrupadas por seletores em blocos de declaração. Os elementos selecionados são denominados o assunto do seletor (W3C, 2015b). Seletores tem o intuito de definir quais partes do documento HTML serão afetadas por determinado bloco de declaração.

CSS é dividido em módulos, que representam conjuntos de funcionalidades, contendo aproximadamente 50 deles. Cada módulo evolui separadamente, esta abordagem é preferível pois permite uma maior quantidade de entrega de novas funcionalidades. Visto que novos recursos não dependem da aceitação de outros para serem disponibilizados. Além do módulos, CSS também é organizado por perfis e níveis.

Os perfis do CSS organizam a especificação por dispositivo de utilização. Existem perfis para dispositivos móveis, televisores, impressoras, etc. A aplicabilidade das regras do CSS varia dependendo do perfil. O conteúdo do elemento *strong*, por exemplo, pode ser traduzido em uma entonação mais forte em um leitor de telas, já em um navegador convencional pode ser apresentado como negrito.

Já os níveis organizam o CSS por camadas de abstração. Os níveis inferiores representam as funcionalidades vitais do CSS, os níveis superiores dependem dos inferiores para construir funcionalidades mais elaboradas.

A primeira especificação do CSS, CSS1 (ou nível 1), foi lançada em 1996. Em 1997 foi lançado o CSS2 com o intuito de ampliar a completude do CSS1. Em 1998 iniciou-se o desenvolvimento do CSS3 que ainda continua em 2015. Além do nível 3 existem módulos de nível 4 no CSS, não obstante o termo CSS3 ainda é o mais utilizado.

Apesar da clara evolução das versões do CSS, esse processo nem sempre é linear. Em 2005 o grupo de trabalho do CSS decidiu aumentar a restrição de suas especificações rebaixando funcionalidades do CSS 2.1 e CSS3 de recomendações para rascunhos.



Figura 2: Os módulos do CSS

Fonte: https://commons.wikimedia.org/wiki/File:CSS3_taxonomy_and_status-v2.png

A última versão do CSS, o CSS3, introduziu várias funcionalidades relevantes para jogos, como *media-queries*, transições, transformações 3D, entre outros.

2.5.1 Media Queries

Media Queries permitem aplicar regras a dispositivos específicos, dependendo de suas capacidades, como resolução, orientação, tamanho de tela, entre outros. A especificação prevê a possibilidade de condicionalmente carregar arquivos JavaScript ou CSS, ou utilizar seletores dentro do CSS de acordo com regras de Media Queries.

Esse carregamento condicional permite implementar fluidez e adaptabilidade de layout para diferentes resoluções. Que segundo Janiszewski (2014) é o mais importante aspecto do desenvolvimento de jogos multiplataforma com as tecnologias da Web.

```
@media only screen and (min-width: 1024px) {
  background-color: green;
}
```

Figura 3: Exemplo de Media Query

A figura 3 demonstra a aplicação de uma regra via seletor Media Query, aplicando a cor de fundo para dispositivos com no mínimo 1024 pixels de resolução.

Além das propriedades convencionais, CSS nível 4 permite a utilização de media queries criadas pelo usuário (*Custom Media Queries*), com regras e definições customizadas. A figura 4 demonstra as novas possibilidades de definição de media queries tanto em CSS como em JavaScript.

```
@custom-media --narrow-window (max-width: 30em);

<script>
CSS.customMedia.set('--foo', 5);
</script>
```

Figura 4: Exemplos de media queries customizadas

Fonte: <https://developer.mozilla.org/en-US/docs/Web/CSS/MediaQueries>

2.5.2 Transições

Transições são uma forma de adicionar animações em uma página web. Estas animações são compostas por um estado inicial e um final. A especificação de transições permite grande controle sobre seus estados, habilitando o desenvolvedor a controlar o tempo de execução, os estados intermediários, e efeitos aplicados a uma transição. Para utilizar transições, assim como em uma máquina de estados, precisamos identificar estados e ações. Estados são seletores do CSS e ações são modificações realizadas entre esses dois seletores CSS (Kuryanovich et al., 2014).

Transições de CSS são interessantes em jogos, especialmente pois muitos navegadores suportam aceleração de GPU (Unidade de processamento gráfico) para estas operações. Isso garante grandes benefícios de performance sobre implementações diretamente em JavaScript.

Segundo Kuryanovich et al. (2014), transições nos permitem construir jogos degradáveis pois os interpretadores de CSS são amigáveis; se eles encontrarem propriedades desconhecidas eles simplesmente as ignoram e continuam a funcionar.

```
div {
  width: 100px;
  height: 100px;
  background: red;
  transition: width 2s;
}

div:hover {
  width: 300px;
}
```

Figura 5: Exemplo de transição

A figura 5 demonstra a utilização de uma transição de tamanho em uma *div* quando o mouse está sobre o elemento. No período de 2 segundos a largura da *div* vai de 100 pixels para 300 pixels.

Atualmente um conjunto finito de propriedades podem ser animadas com transições, e essa lista tende a mudar com o tempo, cabe ao desenvolvedor assegurar-se que determinada propriedade está disponível (MDN, 2015c).

2.5.3 Transformações 3D

Transformações é outra tecnologia do CSS3 que permite grande flexibilidade na construção de jogos. Transformações permitem que elementos sejam traduzidos, rotacionados, escalados e distorcidos em um espaço de duas dimensões (Kuryanovich et al., 2014).

A transformação demonstrada na figura 6 escala o tamanho do elemento com a classe (*test*) para vinte por cento a mais do seu tamanho original. Perceba também os comandos repetidos com o prefixo *ms* e *WebKit*. Esse tipo de abordagem é comum para tecnologias que não passam de rascunhos na especificação.

Assim como transições, as transformações são muitas vezes aceleradas via GPU incrementando a performance de animações criadas com a tecnologia.

```
<style>
.test:hover
{
  -webkit-transform: scale(1.2);
  -ms-transform: scale(1.2);
  transform: scale(1.2);
}
</style>
```

Figura 6: Exemplo de transformação

2.5.4 CSS 4

Apesar de o termo CSS 4 ser bastante utilizado, o grupo de trabalho do CSS não considera mais a existência de versões, como foi até o CSS3. Todavia, existem recursos cuja especificação está avançada e não estavam presentes no CSS 3 quando este foi lançado, dentre estas funcionalidades inclui-se:

- Suporte a variáveis no CSS;
- Media queries customizadas;
- Funções de cores como: `color()`, `hwb()` e `gray()`;
- Suporte a filtros.

Recursos recentes do CSS muitas vezes não estão presentes nos navegadores. Conquanto, muitos deles são interessantes no contexto de desenvolvimento de jogos, como o suporte a variáveis. O projeto `cssnext` <http://cssnext.io/> é uma iniciativa para permitir a utilização dos recentes recursos do CSS mesmo sem eles estarem implementados nos navegadores. O projeto funciona compilando o código não suportado em algo compatível com as versões implementadas pelos navegadores.

Além da apresentação, recurso vital para jogos, e aplicativos Web em geral, é a interatividade. Com as tecnologias da Web esta interatividade é atingida através do JavaScript.

2.6 JavaScript

EMACScript, melhor conhecida como JavaScript, criada por Brendan Eich em 1992, é a linguagem de script da Web. Devido a tremenda popularidade na comunidade de desenvolvedores, a linguagem foi abraçada pela W3C e atualmente é um dos componentes da Open Web. As definições da linguagem são descritas na especificação ECMA-262. Esta possibilitou o desenvolvimento de outras implementações além da original (SpiderMonkey) como o Rhino, V8 e TraceMonkey; bem como outras linguagens similares como JScript da Microsoft e o ActionScript da Adobe.

Segundo a ECMA (2015):

Uma linguagem de script é uma linguagem de programação que é usada para manipular e automatizar os recursos presentes em um dado sistema. Nesses sistemas funcionalidades já estão disponíveis através de uma interface de usuário, uma linguagem de script é um mecanismo para expor essas funcionalidades para um programa protocolado.

No caso de JavaScript na web, os recursos manipuláveis são o conteúdo da página, elementos HTML, elementos de apresentação, a própria janela do navegador e o crescente número de outras funcionalidades que tem seu suporte adicionado por novas especificações.

A intenção original era utilizar o JavaScript para dar suporte aos já bem estabelecidos recursos do HTML, como para validação, alteração de estado de elementos, etc. Em outras palavras, a utilização do JavaScript era opcional e as páginas da Web deveriam continuar operantes sem a presença da linguagem.

Entretanto, com a construção de projetos Web cada vez mais complexos, as responsabilidades delegadas ao JavaScript aumentaram a ponto que a grande maioria dos sistemas Web não funcionarem sem ele. JavaScript não evoluiu ao passo da demanda e muitas vezes carece de definições expressivas, completude teórica, e outras características de linguagens de programação mais bem estabelecidas, como C++ ou Java (Barnett, 2014). A última versão do JavaScript, o JavaScript 6, é um esforço nessa direção. JavaScript 6 ou ECMAScript Harmonia, contempla vários conceitos de orientação a objetos como classes, interfaces, herança, tipos, etc. Infelizmente, por hora, o suporte ao JavaScript 6 é apenas parcial em todos os navegadores. O site <http://kangax.github.io/compat-table/es6/> apresenta um comparativo de suporte das funcionalidades do JavaScript, inclusive o 6, nos navegadores.

Segundo Belchin (2015), o suporte no início de 2015 era o seguinte:

- Chrome: 30%
- Firefox: 57%
- Internet Explorer : 15%
- Opera: 30%
- Safari: 19%

Estes esforços de padronização muitas vezes não são rápidos o suficiente para produtores de software Web. Demora-se muito até obter-se um consenso sobre quais as funcionalidades desejadas em determinada versão e seus detalhes técnicos. Além da espera por especificações, uma vez definidas, é necessário que os navegadores implementem o especificado.

O projeto babel <https://github.com/babel/babel> é um compilador de JavaScript 6 para JavaScript 5. Permitindo que, mesmo sem suporte, os desenvolvedores possam usufruir dos benefícios da utilização do JavaScript 6 durante o tempo de desenvolvimento, gerando código em JavaScript 5 para rodar nos navegadores.

Alternativamente, existe uma vasta gama de conversores de código (*transpilers*) para JavaScript; possibilitando programar em outras linguagens, posteriormente gerando código JavaScript. Entretanto, essa alternativa tem seus pontos fracos, necessita-se de mais tempo de depuração, visto que o JavaScript gerado não é conhecido pelo desenvolvedor, e provavelmente o código gerado não será tão otimizado, nem utilizará os recursos mais recentes do JavaScript.

Mesmo com suas fraquezas amplamente conhecidas, JavaScript está presente em praticamente todo navegador atual. Sendo uma espécie de denominador comum entre as plataformas. Essa onipresença torna-o integrante vital no processo de desenvolvimento de jogos

multiplataforma em HTML5. Vários títulos renomeados já foram produzidos que fazem extensivo uso de JavaScript, são exemplos: Candy Crush Saga, Angry Birds, Dune II, etc.

Jogos Web são geralmente escritos na arquitetura cliente servidor, JavaScript pode rodar em ambos estes contextos, para tanto, sua especificação não define recursos de plataforma. Distribuidores do JavaScript complementam o JavaScript com recursos específicos para suas plataformas alvo. Por exemplo, para servidores, define-se objetos como: terminal, arquivos e dispositivos; no contexto de cliente, são definidos objetos como: janelas, quadros, DOM, etc.

Para o navegador o código JavaScript geralmente é disposto no elemento *script* dentro de arquivos HTML. Quando os navegadores encontram esse elemento eles fazem a requisição para o servidor e injetam o código retornado no documento e, a não ser que especificado de outra forma, iniciam sua execução.

2.6.1 JavaScript 7

Antes da finalização da especificação 6, algumas funcionalidades do JavaScript 7 já haviam sido propostas. Na página <https://github.com/tc39/ecma262> pode-se conferir os itens propostos e seu estágio de evolução. A figura 7 é a tabela de funcionalidades sugeridas e seu estágio no processo da especificação. Alguns dos recursos esperados para o JavaScript 7 são: guards, contratos e concorrência no laço de eventos (Rauschmayer, 2011).

	Proposal	Champion	Stage
	Array.prototype.includes	Domenic Denicola, Rick Waldron	4
	Exponentiation Operator	Rick Waldron	3
	SIMD.JS - SIMD APIs + polyfil	John McCutchan, Peter Jensen, Dan Gohman, Daniel Ehrenberg	3
	Async Functions	Brian Terlson	3
	Object.values/Object.entries	Jordan Harband	3
	String padding	Jordan Harband & Rick Waldron	3
	Trailing commas in function parameter lists and calls	Jeff Morrison	3
	function.sent metaproperty	Allen Wirfs-Brock	2
	Rest/Spread Properties	Sebastian Markbage	2
	ArrayBuffer.transfer	Luke Wagener & Allen Wirfs-Brock	1
	Additional export-from Statements	Lee Byron	1
	Class and Property Decorators	Yehuda Katz and Jonathan Turner	1
	Function.prototype.toString revision	Michael Ficarra	1
	Observable	Kevin Smith & Jafar Husain	1
	String.prototype.{trimLeft,trimRight}	Sebastian Markbage	1
	Class Property Declarations	Jeff Morrison	1
	String#matchAll	Jordan Harband	1
	Shared memory and atomics	Lars T Hansen	1
	Callable class constructors	Yehuda Katz and Allen Wirfs-Brock	1
	System.global	Jordan Harband	1

Figura 7: Propostas do ECMA 7

Fonte: <https://github.com/tc39/ecma262>

2.6.2 Asm.js

Asm.js é um subconjunto da sintaxe do JavaScript a qual permite grandes benefícios de performance quando em comparação com JavaScript normal. Entretanto, não é trivial escrever código em Asm.js e geralmente a criação é feita automaticamente através da conversão de outras linhagens como C. O projeto Emscripten <https://github.com/kripken/emscripten> pode ser utilizado para gerar código em Asm.js é utilizado pelo motor de jogos Unity 3D e Unreal.

No contexto dos jogos performance é um fator de extrema importância, Asm.js se destaca por utilizar recursos que permitam otimizações antes do tempo (*ahead of time optimizations*). Outra grande parcela da performance adicional, em relação ao JavaScript, é devido a consistência de tipo e a não existência de um coletor de lixo (*garbage collector*); a memória é gerenciada manualmente através de um grande vetor. Esse modelo simples, desprovido de comportamento dinâmico, sem alocação e desalocação de memória, apenas um bem definido conjunto de operações de inteiros e flutuantes, possibilita grade performance e abre espaço para otimizações.

O desenvolvimento do Asm.js iniciou-se no final de 2013; não obstante, a maioria dos navegadores não implementam ou implementam parcialmente o rascunho. O motor JavaS-

cript da Mozilla, SpiderMonkey, é a exceção, implementando a grande maioria dos recursos do Asm.js.

2.6.3 Web Assembly

Web Assembly é uma tecnologia que pretende definir um formato de máquina da Web. A tecnologia ainda está em seus estágios iniciais de desenvolvimento, nem contando com um grupo de trabalho. Mesmo assim, sabe-se que Web Assembly irá permitir que outras linguagens além do JavaScript gerem código binário que rode nos navegadores com grande ganhos de performance e flexibilidade.

Além da versão binária, otimizada para performance, uma versão em texto também está prevista, ideal para desenvolvimento e depuração. Bibliotecas e aplicações que requeiram grande performance como motores de física, simulações e jogos em geral vão se beneficiar substancialmente com o Web Assembly.

A iniciativa do Web Assembly está sendo desenvolvida pelo Google, Microsoft, Mozilla, entre outros, tornando a proposta uma possibilidade promissora. Seu objetivo não é substituir o JavaScript, outrossim habilitar que aplicações que necessitem de grande performance possam ser inclusas na Web. A ideia do Web Assembly é uma continuação do trabalho do Asm.js, uma forma de trazer performance similar a nativa eliminando grande parte das abstrações que o JavaScript traz embutidas.

Visto que os desenvolvedores de motores JavaScript terão que colocar o código do Web Assembly na mesma base que o do JavaScript, as expectativas são de que o JavaScript consiga aproveitar partes da implementação do Web Assembly incrementando a performance do JavaScript, sendo assim uma situação onde ambos ganham.

A aplicabilidade do Web Assembly em jogos em produção ainda é praticamente nula. Até então apenas um polyfill do Web Assembly está disponível e pode ser encontrado no seguinte link https://github.com/Web_Assembly/polyfill-prototype-1. Mas conforme a especificação evolui a probabilidade é que as empresas interessadas implementem a especificação em seus navegadores e os desenvolvedores de jogos comecem a integrar a tecnologia em suas aplicações.

2.6.4 Web Animations

Web Animations é uma especificação em rascunho que define uma forma imperativa de manipular animações através de JavaScript. Como demonstrado na figura 8 a tecnologia vai permitir manipular as animações de elementos do DOM, com a possibilidade de filtrar por tipo de animação, alterar a taxa de animações, o tempo de execução, entre outras propriedades de uma forma dinâmica, através de scripts. Visto que Web Animations lida diretamente com o DOM, animações podem ser aplicadas para SVG além de CSS, servindo como uma tecnologia para unificar animações.

Grande controle sobre animações é desejável para os jogos; não obstante, visto que a

especificação é muito nova, somente o Google Chrome a implementa. A biblioteca <https://github.com/web-animations/web-animations-js> serve como polyfill para os demais navegadores.

```
elem.getAnimations().filter(
  animation =>
    animation.effect instanceof
    KeyframeEffectReadOnly &&
    animation.effect.getFrames().some(
      frame => frame.hasOwnProperty('transform')
    )
).forEach(animation => {
  animation.currentTime = 0;
  animation.playbackRate = 0.5;
});
```

Figura 8: Exemplo de utilização de Web Animations

Fonte: <http://www.w3.org/TR/Web-animations/>

O site <http://web-animations.github.io/web-animations-demos/> contém uma coleção de animações utilizando a tecnologia.

2.7 Navegadores

Navegadores são aplicações, onde as tecnologias da OWP são interpretadas e geram um conteúdo útil para os usuários. São os clientes em uma arquitetura cliente servidor. O servidor desta arquitetura geralmente é um servidor Web cujo objetivo principal é fornecer páginas HTML para o navegador processar. A comunicação entre o navegador e o servidor Web se dá através da troca de mensagens no protocolo HTTP.

Nos navegadores, os usuários necessitam saber o endereço de determinado servidor, ou utilizar buscadores para auxiliá-los. Este é um processo árduo para as plataformas móveis pois necessitam maior interação dos usuários, e não são “naturais” se comparado ao modo de consumir aplicativos nestas mesmas plataformas. Simplesmente adquirindo o aplicativo na loja e abrindo-o no sistema operacional⁶.

Uma vez localizado o endereço, o navegador manda uma mensagem em HTTP requisitando o conteúdo de determinado local. O servidor responde a mensagem HTTP com um documento HTML e o cliente, ao receber, começa o processo de renderização. O processo de renderização é complexo e a grande maioria dos navegadores confia em bibliotecas especializadas para efetuar este trabalho, estas bibliotecas são denominadas motores de renderização.

Alguns motores de renderização incluem:

⁶Algumas formas de contornar este problema serão descritos nas seção 2.22.

- Blink: Utilizado no Opera, Google Chrome e projetos relacionados;
- Gecko: Utilizado nos produtos da Mozilla;
- KHTML: Utilizado no navegador Konqueror, esta serviu de base para o Blink;
- WebKit: Utilizado no Safari e versões antigas do Google Chrome.

A renderização consiste na decodificação de um documento em HTML para sua representação na memória e posterior pintura no espaço de tela do navegador. Interpretar os documentos é processo árduo e alguns motores dependem de bibliotecas externas para fazê-lo.

Para interpretar HTML o motor WebKit utiliza a biblioteca Bison, já o Gecko utiliza uma biblioteca própria (Garisel; Irish, 2011). Durante o processo de renderização o navegador pode requisitar outros arquivos do servidor a fim de completar a experiência desejada para o documento em questão. Geralmente após a renderização do documento vem a execução de scripts. As bibliotecas que executam JavaScript são chamadas de motores de JavaScript. Abaixo segue uma lista dos motores de JavaScript mais comuns.

- SpiderMonkey: Primeiro motor, desenvolvido por Brendan Eich, escrito em C++;
- Rhino: Criada pela Netscape, escrito em Java;
- Nitro: Criada pela Apple;
- V8: Criada pelo Google;
- TraceMonkey: Criada pela Mozilla.

Cada um dos motores citados acima são partes que compõem um navegador e em conjunto buscam cobrir as especificações da Web. Infelizmente a forma que as tecnologias são suportadas varia e algumas não estão presentes de qualquer forma nos navegadores. Conquanto, o suporte vem crescendo. A figura 12 apresenta o gráfico de suporte por versões de navegadores em dezembro de 2015.

TIMELINE



Figura 9: Suporte das especificações do HTML nos navegadores

Fonte: <https://html5test.com/>

2.8 Android

É um sistema operacional open-source criado em 2003 pela Android Inc e mantido pelo Google desde 2005. Android funciona em uma variedade de dispositivos de celulares, tablets, netbooks a computadores desktop, mas seu foco é dispositivos com tela sensível (JR, 2010). O sistema operacional é composto por diversos projetos open-source, sendo o mais proeminente deles o kernel Linux, utilizado como fundamento do sistema operacional. Além da versão open-source (AOSP), existe a versão do Google que utiliza ferramentas proprietárias para adicionar funcionalidades aos dispositivos.

Softwares para Android são geralmente escritos em Java e executados através da máquina virtual Dalvik. Dalvik é similar a máquina virtual Java, mas roda um formato de arquivo diferenciado (.dex), otimizados para consumir pouca memória, que são agrupados em um único pacote (.apk).

Aplicativos da Web podem ser integrados no Android através de uma arquitetura híbrida.

Os arquivos da aplicação são empacotados dentro de um apk utilizando um componente nativo do Android, a API WebView, que se responsabiliza por interpretá-los.

Além da arquitetura híbrida, é possível executar jogos Web em dispositivos Android através dos navegadores presentes nestes aparelhos. As novas versões do Android contam com o Google Chrome como navegador padrão. Já as versões antigas contêm um navegador próprio que utiliza o motor de renderização LibWebCore, baseado no WebKit (Dorokhova; Amelichev, 2010). Além do padrão, outros navegadores como o Firefox ou Opera também podem ser instalados.

Além do Android o sistema IOS é de grande relevância para o desenvolvimento de jogos. Entretanto, não será tratado neste trabalho pelos motivos supracitados. A próxima seção deste trabalho descreve como detectar recursos nas variadas plataformas que a Web se apresenta.

2.9 Detecção de Recursos

Visto que nenhum navegador implementa as especificações HTML completamente, cabe ao desenvolvedor detectar os navegadores que não comportam as necessidades tecnológicas dos aplicativos que cria. Ao deparar-se com uma funcionalidade faltante o desenvolvedor tem duas possibilidades: notificar o usuário sobre o problema ou utilizar polyfills.

Polyfills são recursos que simulam uma funcionalidade não disponível nativamente nos navegadores. A biblioteca Gears <https://developers.google.com/gears> é um exemplo. Gears serve para prover recursos de Geolocalização para navegadores que não implementam a especificação do HTML5. Essa capacidade de suportar tecnologias que não estão ainda disponíveis (ou nunca estarão no caso de dispositivos legados) através de polyfills é uma das características que faz a Web uma plataforma de tão grande abrangência. Novas tecnologias são criadas a todo o momento; entretanto, o suporte a essas funcionalidades geralmente não acompanham o passo das inovações. E ainda assim os usuários podem se beneficiar de uma taxa substancial delas através de polyfills. Algumas funcionalidades do HTML, como geolocalização e vídeo foram primeiramente disponibilizadas através de plugins. Outras funcionalidades, como o canvas, podem ser totalmente emuladas via polyfills em JavaScript (Pilgrim, 2010).

Detectar suporte aos variados recursos do HTML5 no navegador pode ser uma tarefa entediante. É possível implementar testes para cada funcionalidade utilizada abordando os detalhes de implementação de cada uma ou então fazer uso de alguma biblioteca especializada neste processo. O Modernizr é uma opção open-source deste tipo de biblioteca, este gera uma lista de booleanos sobre grande variedade dos recursos HTML5, dentre estes, geolocalização, canvas, áudio, vídeo e armazenamento local.

A quantidade de especificações que um aplicativo complexo como um jogo utiliza pode ser bem grande, e muitas vezes é difícil dizer quais navegadores implementam o quê. Uma boa referência do suporte a recursos nos navegadores é o site <http://caniuse.com/>. E uma lista com vasta quantidade de polyfills para as tecnologias da Web pode ser encontrada no site

<https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills>.

2.10 Renderização

Renderização é parte fundamental de muitos jogos. As tecnologias que permitem renderização na Web serão descritas abaixo.

2.10.1 SVG

SVG (*Gráficos de vetores escaláveis*), é uma linguagem baseada em XML especializada na criação de vetores bidimensionais (Kuryanovich et al., 2014). Segundo Neumann; Winter (2001, p. 4) SVG foi criada em conjunto por empresas como: Adobe, Apple, AutoDesk, entre outras, sendo que seus produtos contam com rápida integração a tecnologia.

Por descrever imagens utilizando vetores ao invés de mapas de bits, os tamanhos dos arquivos em SVG são geralmente pequenos e podem ser comprimidos com grande eficiência. Talvez a característica mais marcante do SVG é que não há diferença de qualidade em resoluções visto que os vetores são escaláveis. Sendo que pequenos arquivos servem igualmente bem um monitor com baixa resolução como um monitor retina.

Por ser baseado em XML, uma das tecnologias da Web, SVG permite a utilização da API do DOM para manipular seus elementos. Tornando simples a integração com outras tecnologias da Web. Pode-se utilizar arquivos CSS para customizar a apresentação, JavaScript para adicionar interatividade, etc.

Além de grande integração com as demais tecnologias, SVG conta com uma API nativa poderosa. Os elementos geométricos do SVG incluem retângulos, círculos, elipses, linhas e polígonos (Neumann; Winter, 2001, p. 5). Também existe a possibilidade de declarar caminhos customizados através do elemento *path*, algo similar com o *Path2D* do Canvas. E cada um dos elementos, ou agrupamento de elementos podem ser transformados; traduzidos, redimensionados, rotacionados e distorcidos (Neumann; Winter, 2001, p. 5). Para ilustrar a utilização, a figura 10 demonstra um círculo sendo definido em SVG.

```

<svg width="100" height="100">
  <circle
    cx="50"
    cy="50"
    r="40"
    stroke="green"
    stroke-width="4"
    fill="yellow"
  />
</svg>

```

Figura 10: Círculo em SVG.

Fonte: <http://www.w3schools.com/svg/>

Outra tecnologia popular da Web para renderização que adota uma filosofia totalmente diferente do SVG é o Canvas.

2.10.2 Canvas

O elemento *canvas* define uma camada de mapa de bits em documentos HTML que pode ser usada para criar diagramas, gráficos e animações 2D. Foi criado pela Apple em 2004 para renderizar elementos de interface no Webkit, logo foi adotado por outros navegadores e se tornou um padrão da OWP.

Em um documento HTML, canvas é um retângulo onde pode-se usar JavaScript para desenhar (Pilgrim, 2010, p. 113). Mais especificamente, o retângulo do canvas é um espaço vetorial cuja origem se dá na esquerda superior. Normalmente cada unidade do plano cartesiano corresponde a um pixel no canvas (MDN, 2015b). Manipular o retângulo é uma analogia de como desenhar manualmente, move-se o "lápis" para o local desejado e traça-se os pontos onde a linha (caminho) deve percorrer. Além da possibilidade de desenhar linhas livremente também é possível criar retângulos nativamente. Todas as demais figuras geométricas precisam ser feitas através da junção de caminhos (MDN, 2015b). Para desenhar caminhos curvos, de modo a criar círculos e elipses, existem funções especiais de arco.

Escrever no canvas envolve a manipulação de diversos caminhos e retângulos, em jogos que fazem extensivo uso do canvas a complexidade de manipulação de linhas pode crescer muito. As últimas versões do Canvas introduziram o objeto Path2D que possibilita o armazenamento e composição de instruções de caminhos a fim de possibilitar o reuso de formas. Ao invés de utilizar os métodos de caminhos diretamente no contexto do canvas utiliza-se uma instância do objeto *Path2D*. Todos os métodos relacionados aos caminhos como o *moveTo*, *arc* ou *quadraticCurveTo* estão disponíveis no objeto Path2D (MDN, 2015b). Também é possível utilizar a notação do SVG na criação de uma instancia de Path2D possibilitando a reutilização de conteúdo para ambas as tecnologias.

Além da possibilidade de desenhar programaticamente é possível carregar gráficos. Muitos dos jogos HTML5 utilizam sprites ou padrões recortáveis *tiled*, técnicas bastante similares aos títulos antigos da SNES e Game Boy (Rocheleau, 2015).

Apesar da API do canvas ser poderosa não é possível manipular diretamente as camadas já desenhadas. Alternativamente pode-se limpar o canvas inteiramente em pontos determinados ou então sobrescrever as partes que se deseja alterar.

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.fillStyle = "#FF0000";
ctx.fillRect(0,0,150,75);
```

Figura 11: Canvas

Fonte: http://www.w3schools.com/html/html5_canvas.asp

A figura 11 demonstra a utilização do canvas 2D para a criação de um retângulo. Note que o objeto canvas tem que carregar um contexto que então é utilizado como API para manipular a mapa de bits em 2D.

O canvas até aqui descrito trata-se de sua forma, ou contexto 2D. A especificação 3D do canvas é o WebGL .

2.10.3 WebGL

WebGL é uma API JavaScript otimizada para desenhar gráficos em três dimensões. Ideal para a criação de ambientes virtuais, jogos e simulações. Por ser uma tecnologia da OWP WebGL foi especificado para funcionar nativamente nos navegadores sem a ajuda de plugins ou ferramentas de terceiros.

WebGL foi desenvolvida baseando-se na especificação OpenGL a qual trata de definir como renderizar gráficos multiplataforma. Especificamente OpenGL ES, é uma versão do OpenGL otimizada para dispositivos móveis. O órgão que especifica o WebGL é o mesmo que especifica o OpenGL, o grupo sem fins lucrativos Khronos. Os primeiros rascunhos do WebGL iniciaram em 2006; não obstante, o grupo de trabalho não foi formado até 2009 e a primeira versão foi lançada em 2011.

Apesar de ter sido desenvolvida com foco em 3D, WebGL pode ser igualmente utilizada para criação de gráficos em duas dimensões(Matti; Arto, 2011, p. 6). O elemento do DOM que provê a interface do WebGL é o canvas, no contexto 3D. Essa integração com o DOM via tag canvas permite que o WebGL seja manipulado assim como os demais elementos HTML. A especificação é composta por uma API de controle em JavaScript e o processamento shaders do lado da GPU (Central de processamento gráfico).

Shaders são scripts que definem níveis de cor ou efeitos especiais sobre um modelo 2D ou 3D. Contam com grande performance, possibilitando conteúdo em tempo real como no caso de jogos. São utilizados no cinema, em imagens geradas por computadores e videogames.

Existem dois shaders principais, de vértices e fragmentos. Shaders de vértices são chamados para cada vértice sendo desenhado definindo suas posições definitivas. Já shaders de fragmentos atuam na cor de cada pixel a ser desenhado (Matti; Arto, 2011, p.15).

Kuryanovich et al. (2014) cita que conforme a habilidade do desenvolvedor aumenta, mover funções antes delegadas ao JavaScript para os shaders pode aumentar a performance e oferecer uma ampla coleção de efeitos e realismo.

Um endereço interessante para explorar exemplos WebGL do iniciante ao avançado é o blog learningwebgl.com que conta com tutoriais cobrindo áreas como diferentes tipos de iluminação, carregamento de modelos em JSON, gerenciando eventos; e como renderizar uma cena WebGL em uma textura (Matti; Arto, 2011, p.42)⁷.

Apesar da relevância, WebGL não foi utilizado no protótipo pois ainda não está completamente suportado em navegadores populares como o Firefox e a grande curva de aprendizado do WebGL puro é muito grande para se encaixar no escopo deste projeto.

Uma tecnologia que se integra profundamente como ambientes virtuais em três dimensões criados via WebGL é o WebVR.

2.10.4 WebVR

Segundo Brooks (1999) realidade virtual é uma experiência em que o usuário é efetivamente imerso em um mundo virtual responsivo. Realidade virtual é uma área nem tão nova mas que recebeu interesse renovado recentemente. Isso se dá, pelo menos em parte, pela massificação dos dispositivos móveis inteligentes. O hardware necessário para fornecer uma experiência minimamente viável como acelerômetros, câmeras e telas de alta resolução está disponível em praticamente todos os dispositivos comercializados.

Realidade virtual é uma área de grande interesse para os produtores de jogos, pois pode oferecer alto nível de imersão nos já interativos e desafiadores ambientes dos jogos. WebVR é uma especificação que pretende trazer os benefícios da realidade virtual para dentro do mundo da Web. Em termos simples, a especificação define uma forma de traduzir movimentos de acelerômetros e outros sensores de posição e movimento para dentro do contexto de uma câmera em um ambiente 3D WebGL.

Atualmente a especificação do WebVR se encontra em fase de rascunho e as últimas versões do Firefox, e versões compiladas manualmente do Google Chrome já permitem a utilização.

⁷Os apêndices contam com uma coleção de bibliotecas que facilitam a utilização de WebGL

2.11 WebCL

É uma API em JavaScript para o recursos de OpenCL que permitem computação paralela com grandes ganhos de performance. Aplicativos como motores de física e renderizadores de imagens, ambos relevantes para os jogos, podem se beneficiar grandemente de processamento feito em paralelo, possivelmente na GPU. OpenCL é um framework para escrever programas que funcionem em plataformas com diversas unidades de processamento. Assim como WebGL a WebCL é especificada e desenvolvida pelo grupo Kronos. A primeira versão da especificação foi no início de 2014 mas até então nenhum navegador implementa o definido.

Após a revisão de tecnologias de renderização serão abordadas tecnologias de multimídia: áudio e vídeo. Mas para falar de ambos, antes é necessário falar de codecs.

2.12 Codecs

Codec é o algoritmo usado para codificar e decodificar vídeo ou áudio em um conjunto de bits (Pilgrim, 2010). O termo Codec é um acrônimo, significando o processo de codificar (*coder*) um fluxo de dados para armazenamento e decodificá-lo (*decoder*) para ser consumido.

Dados multimídia são geralmente enormes, sem serem codificados, um vídeo ou áudio consistiriam em uma vasta quantidade de dados que seriam muito grandes para serem transmitidos pela Internet em um período de tempo razoável (Lubbers; Brian; Frank, 2010, p. 66). O objetivo dos codecs é diminuir o tamanho dos arquivos com a menor perda de qualidade possível. Para isso os codecs utilizam de várias estratégias de compressão ou descarte de dados; podendo rodar tanto em hardware quanto em software.

Existem codecs desenvolvidos especificamente para a Web. Buscam uma razão de tamanho e qualidade aceitável, mas prezando por tamanho. Uma das otimizações realizadas por codecs de vídeo na Web, é a não troca de todo conteúdo de um quadro para o próximo, possibilitando maiores taxas de compressão, que resulta em arquivos menores (Pilgrim, 2010).

O funcionamento de codecs pode variar muito, conforme as estratégias de compressão utilizadas, a quantidade de bits por segundo suportada, entre outros fatores. Visto que os algoritmos de compressão podem adquirir grande complexidade muitos codecs são encobertos por licenças que limitam sua utilização. Embora, também existem opções livres de patentes e licenças.

Abaixo segue uma lista de alguns codecs populares para áudio segundo Lubbers; Brian; Frank (2010, p. 67).

- ACC;
- MPEG-3;
- Vorbis.

Ainda segundo Lubbers; Brian; Frank (2010, p. 67) abaixo segue uma lista de codecs populares para vídeo.

- H.264;
- VP8;
- OggTheora.

Após um fluxo de dados multimídia ter sido codificado através do algoritmo de codec ele é armazenado em um contêiner. Contêiners são um padrão de metadados sobre as informações codificadas de modo a possibilitar que outros programas consigam interpretar estas informações de forma padronizada. Como um arquivo *zip*, contêiners podem conter qualquer coisa dentro de si (Pilgrim, 2010). Assim como codecs, existem contêiners livres e com restrições de licença. Abaixo segue uma lista de alguns contêiners de áudio.

- Audio Video Interleave (.avi);
- MPEG-2 Audio Layer III (.mp3);
- Matroska (.mkv);
- Vorbis (.ogg);
- Opus (.opus).

Abaixo segue uma lista de alguns contêiners de vídeo.

- Audio Video Interleave (.avi);
- Flash Video (.flv);
- MPEG4 (.mp4);
- Matroska (.mkv);
- Ogg (.ogv);
- WebM (.webm).

O suporte a codecs e contêiners na Web varia de navegador para navegador, de acordo com as preferências mercadológicas, técnicas ou filosofias das empresas por trás dos navegadores. Segundo Pilgrim (2010) não existe uma única combinação de contêiner e codecs que funcione em todos os navegadores ⁸.

⁸A figura 12 apresenta um comparativo interessante sobre os formatos populares de áudio considerando o fator taxa de bits (quantidade de informação armazenável por segundo) versus qualidade (perceptível por humanos).

2.13 Áudio

Áudio é um componente vital para oferecer imersão e feedback aos usuários de jogos. O componente de áudio é especialmente útil para jogos de ação (Vahatupa, 2014). Efeitos de som e música podem servir como parte da mecânica dos jogos.

Antes do HTML5 não havia como consumir áudio na Web sem a utilização de plugins de terceiros. A especificação do HTML define duas formas de utilizar áudio na Web: através do elemento HTML áudio ou através da API JavaScript de áudio.



Figura 12: Comparação de codecs de áudio

Fonte: <https://www.opus-codec.org/comparison/>

2.13.1 Elemento Áudio

O elemento *audio* foi a primeira tecnologia de áudio nativa para Web, ele define um som dentro de um documento HTML. Quando o elemento é renderizado pelos navegadores, ele carrega o conteúdo que pode ser reproduzido pelo programa dentro do navegador.

```
<audio controls>
<source src="horse.ogg" type="audio/ogg">
<source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

Figura 13: Exemplo de utilização da tag áudio

Fonte: http://www.w3schools.com/HTML/HTML5_audio.asp

A imagem 13 demonstra a utilização da tag *audio*. Os elementos *source* demonstrados na figura referenciam arquivos de áudio contendo um par de contêiner e codec. Mais de um *source* é necessário pois os criadores de navegadores não chegaram a um consenso sobre qual formato deve ser usado, sendo necessário utilizar vários para suportar todos os navegadores populares ⁹.

A especificação declara que todo o conteúdo dentro de uma tag *audio*, que não sejam elementos *source*, seja ignorado pelo navegador. O que permite que seja adicionada marcação de reserva para tratar os casos de quando não existe suporte a tag *audio* no navegador. Visto que os navegadores que não suportam áudio vão buscar renderizar o conteúdo dentro da tag. A figura 13 ilustra este comportamento através da mensagem *Your browser does not support the audio element* que só será apresentada se o navegador do usuário não tiver suporte a tag *audio*.

O objetivo inicial da tag *audio* é reproduzir um som e parar. Ideal para ouvir música ou como um som de fundo de um site. Por conseguinte, a tag *audio* não é o suficiente para comportar aplicações de áudio complexas (Rogers, 2012). A grande maioria de jogos muitas vezes precisam lançar múltiplos sons derivados de ações de usuário e outros eventos, nestes casos a API de áudio é mais adequada.

2.13.2 API de Áudio

É uma interface experimental (ainda em rascunho) em JavaScript para criar e processar áudio. O objetivo da especificação é incluir capacidades encontradas em motores de jogos modernos e também permitir o processamento, mistura e filtragem de som, funcionalidades que estão presentes nas aplicações de processamento de áudio modernas para desktop (Rogers, 2012).

A API especificada provê uma interface para manipular nodos de áudio que podem ser conectados permitindo refinado controle sobre os efeitos sonoros. O processamento se dará primeiramente em uma camada inferior (tipicamente código Assembly / C / C++), mas a síntese e processamento em JavaScript também serão suportadas (Rogers, 2012).

Essa tecnologia é muito mais nova do que o elemento *audio*. Diferentemente dos demais navegadores o Internet Explorer não dá nenhum nível de suporte a API. O polyfill *AudioContext* suporta as partes básicas da API e pode ser utilizado nos casos onde não existe suporte para a API do HTML ¹⁰.

As últimas versões da especificação da Audio API contam com a possibilidade de manipular áudio através de Web Workers, o que traz oportunidades interessantes para aplicações que dependam de muito processamento de áudio, visto que o processamento pode ser feito em uma *thread* separada.

⁹Com o site <http://hpr.dogphilosophy.net/test/> é possível detectar os formatos de codecs de áudio suportados pelo navegador sendo utilizado

¹⁰O polyfill *AudioContext* pode ser encontrado no seguinte endereço <https://github.com/shinnn/AudioContext-Polyfill>

Além de grande flexibilidade com áudio alguns jogos requerem a disponibilidade de vídeo para utilizar como introdução, cinemáticas, entre outros recursos que habilitam uma experiência mais rica ao usuário. Abaixo será feita uma revisão sobre a tecnologia de vídeo em HTML.

2.14 Vídeo

O elemento *video* define uma forma de adicionar vídeos na Web nativamente, sem a necessidade de utilizar plugins de terceiros como o Flash Player. Assim como com o elemento *audio* pode-se adicionar um arquivo através do atributo *src* do elemento ou adicionar vários formatos de contêiner e codec dentro da tag através de elementos *source*. O navegador decidirá em tempo de execução qual formato executar dependendo de suas capacidades.

```
<video controls style="width:640px;height:360px;" poster="poster.png">
  <source src="devstories.webm"
    type='video/webm;codecs="vp8, vorbis"' />
  <source src="devstories.mp4"
    type='video/mp4;codecs="avc1.42E01E, mp4a.40.2"' />
  <track src="devstories-en.vtt" label="English subtitles"
    kind="subtitles" srclang="en" default></track>
</video>
```

Figura 14: Exemplo de utilização de vídeo

Fonte: <http://www.html5rocks.com/en/tutorials/video/basics/>

A figura 14 demonstra a utilização de algumas funcionalidades do elemento *video*. Como demonstrado na figura, além do elemento *source*, a tag *video* suporta o elemento *track*. O qual permite informar subtítulos para os vídeos sendo apresentados. Também é possível habilitar controles de vídeo nativos dos navegadores, como demonstrado através do atributo *controls*.

Como o elemento vídeo se encontra no HTML é possível manipulá-lo com uma gama de tecnologias. Com CSS é possível aplicar escala de cinza do sobre o elemento vídeo gerando um efeito preto e branco interessante. A especificação do elemento *video* também permite controlar quais partes do vídeo serão mostradas através de parâmetros de tempo passados como argumentos junto ao nome do arquivo. Ou capturar frames de vídeo dentro do elemento canvas.

Além de ser flexível nas tecnologias de multimídia, um requerimento comum em jogos é haver uma forma de armazenar dados eficientemente e buscá-los com agilidade. Abaixo serão discutidas as tecnologias de armazenamento disponíveis para a Web.

2.15 Armazenamento

Uma das grandes limitações do HTML era a ausência de capacidade de armazenamento de dados no lado do cliente. Antes do HTML5 a única alternativa era usar cookies, os quais

tem um armazenamento de no máximo 4k e trafegam em toda a requisição, tornando o processo lento. Essa área era onde as aplicações nativas detinham grande vantagem sobre as aplicações Web. O HTML5 solucionou este problema introduzindo várias formas de armazenamento de dados (Hasan et al., 2012). Armazenamento local é um recurso importante para jogos, tanto por diminuir a latência da persistência na rede, quanto para possibilitar uma experiência offline.

Existem algumas especificações sobre armazenamento, mas a grande parte delas não conta com suporte completo em todos os navegadores comuns, um polyfill interessante para Web Storage e IndexedDB é o projeto localForge <https://github.com/mozilla/localForage> da Mozilla.

2.15.1 Web SQL

A especificação Web SQL introduz uma API para manipular banco de dados relacionais em SQL. A especificação suporta transações, operações assíncronas e um tamanho de armazenamento substancial: 5 MB, o qual pode ser estendido pelo usuário. O grupo de trabalho do Web SQL iniciou-se em 2010 e foi suspenso ainda como rascunho. Apesar de ser um recurso desejável para muitos desenvolvedores, foi descontinuada pelos motivos descritos abaixo.

Segundo Pilgrim (2010)

Todos os implementadores interessados em Web SQL utilizaram a mesma tecnologia (Sqlite), mas para a padronização ficar completa é necessário múltiplas implementações. Até outro implementador se interessar em desenvolver a especificação a descrição do dialeto SQL apenas referencia o SQLITE, o que não é aceitável para um padrão.

Entretanto, a especificação ainda é suportada pelo Google Chrome, Safari, Opera e Android, entre outros. Mas até que outros implementadores se prontifiquem a especificação continuará suspensa. No lugar do Web SQL a W3C recomenda a utilização do Web Storage e do IndexedDB.

2.15.2 Web Storage

Web Storage, também conhecido como Local Storage, provê uma forma de armazenar dados no formato chave-valor dentro do navegador. Os dados são persistidos mesmo que o usuário feche a página ou o navegador. Web Storage é um recurso similar a cookies, contudo algumas diferenças substanciais são perceptíveis. Web Storage não requer que os dados sejam trafegados como cabeçalhos nas requisições. Também provê maiores espaços de armazenamento quando comparado a cookies. A tecnologia começou como parte da especificação do HTML5 mas agora conta com um documento próprio mantido pela W3C. A especificação é suportada pela grande maioria dos navegadores populares.

A especificação oferece duas áreas de armazenamento, o armazenamento local e de sessão. O armazenamento local é persistido por domínio e outros scripts provindos deste mesmo domínio poderão fazer uso da informação. O armazenamento de sessão é para informações que

podem variar de aba para aba e que não é interessante que sejam persistidos para demais acessos além do atual.

```
// Store value on browser for duration of the session
sessionStorage.setItem('key', 'value');

// Retrieve value (gets deleted when browser is closed and re-opened)
alert(sessionStorage.getItem('key'));

// Store value on the browser beyond the duration of the session
localStorage.setItem('key', 'value');

// Retrieve value (persists even after closing and re-opening the browser)
alert(localStorage.getItem('key'));
```

Figura 15: Web Storage na prática

Fonte: https://en.wikipedia.org/wiki/Web_storage#usage

A API do Web Storage é simples, consistindo em uma interface para buscar dados e outra para armazenar, no formato chave/valor. A figura 15 exemplifica a utilização do Web Storage, para realizar o armazenamento em sessão utiliza-se o objeto *sessionStorage*. Já para utilizar o armazenamento local utiliza-se o objeto *localStorage*.

Web Storage é uma solução simples que comporta muitos casos de uso. Contudo, muitas vezes é necessário um controle mais refinado sobre os dados, ou mais performance em uma base de dados massiva. Para responder a estes desafios existe a especificação do IndexedDB.

2.15.3 IndexedDB

IndexedDB é um banco de dados que suporta o armazenamento de grandes quantidades de dados no formato de chave/valor o qual permite alta performance em buscas baseadas em índices. A tecnologia é uma recomendação da W3C desde janeiro de 2015 e suportada, pelo menos parcialmente, por praticamente todos os navegadores populares.

Inicialmente IndexedDB permitia operações síncronas e assíncronas. Todavia, a versão síncrona foi removida devido a falta de interesse da comunidade. Operações assíncronas permitem que aplicativos JavaScript não esperam pelo resultado para continuar a execução. Outrossim, cada interação com o banco de dados é uma transação que pode retornar um resultado ou um erro. Os eventos da transação são internamente eventos DOM cuja propriedade *type* do elemento foi definida como *success* ou *error*.

Ao invés de tabelas, IndexedDB trabalha com repositórios de objetos. Cada entrada, tupla em SQL, de um determinado repositório pode ser de um formato diferenciado, com exceção da chave única que deve estar presente em cada uma das entradas.


```

var db;
var request = window.indexedDB.open("Mydb", 9);
request.onsuccess = function(event) {
  db = event.target.result;
  var transaction = db.transaction(["customers"], "readwrite");
  var objectStore = transaction.objectStore("customers");
  var request = objectStore.add({email: "mymail@domain.com", name: "foo"});
  request.onsuccess = function(event) {
    console.log('customer added')
  };
}

```

Figura 16: Adicionando um cliente em IndexedDB.

A figura 16 demonstra um exemplo simplificado da utilização do IndexedDB. Na figura fica visível como cada interação com o banco de dados é construído através de uma nova requisição e como o tratamento do resultado é dado dentro de eventos.

A característica assíncrona do IndexedDB, é fundamentada na premissa de não perturbar o fluxo principal da aplicação enquanto processamento não vital, e possivelmente demorado, ocorre. Outra tecnologia da Web que utiliza os mesmos princípios é Web Workers.

2.16 Web Workers

É uma API que possibilita executar vários scripts (*threads*) JavaScript ao mesmo tempo. O script que cria uma thread é chamado de pai da thread, e a comunicação entre pai e filhos pode acontecer de ambos os lados através de mensagem encapsuladas em eventos. Um script que não seja pai de uma thread não pode se comunicar com ela, a não ser que a thread esteja em modo compartilhado.

O contexto global (objeto *window*) não existe em uma thread, no seu lugar o objeto *DedicatedWorkerGlobalScope* pode ser utilizado. Workers compartilhados podem utilizar o *SharedWorkerGlobalScope*. Estes objetos contém grande parte das funcionalidades proporcionadas pelo *window* com algumas exceções, por exemplo, threads não podem fazer alterações no DOM.

2.17 Offline

Disponibilizar aplicações Web offline é uma característica introduzida no HTML5. Uma nova gama de aplicativos Web foram possibilitados devido as tecnologias offline. A importância de gestão offline é tanta em alguns nichos que os avaliadores do mercado de software do IOs consideram quase uma obrigação realizar gestão offline nas aplicações (Monony, 2015).

Jogos de usuário único podem se beneficiar enormemente de aplicativos offline, tornando possível utilizar a aplicação com ou sem a presença de rede. Para tanto é necessário

poder armazenar dados locais, tecnologias como IndexedDB e Web Storage permitem isso. O outro requerimento para estar offline é uma forma de armazenar os arquivos da Web localmente, de forma que sejam utilizados quando não houver uma conexão a rede.

HTML5 conta com uma especificação estável de APIs de cache offline mantida pela W3C. Esta especificação determina que uma arquivo de manifesto contenha quais arquivos serão guardados para utilização offline e possivelmente quais serão usados pela rede. A figura 17 exemplifica um arquivo de manifesto. Os arquivos abaixo da palavra *CACHE MANIFEST* serão armazenados em cache e não serão buscados na rede a não ser que o arquivo de manifesto seja modificado. Já os arquivos abaixo da palavra *NETWORK*: serão utilizados exclusivamente com rede e serão buscados todas as vezes. Ainda existe a palavra chave *FALLBACK* onde todos os itens abaixo dela serão utilizados para substituir arquivos de rede.

É uma boa prática colocar um comentário com a versão do arquivo de manifestos. Desse modo quando um arquivo for modificado incrementa-se a versão do arquivo de manifestos e as modificações serão baixadas nos navegadores clientes.

```
CACHE MANIFEST
index.html
help.html
style/default.css
images/logo.png
images/background.png

NETWORK:
server.cgi
```

Figura 17: Exemplo de arquivo de manifesto offline

Fonte: <http://www.w3.org/TR/offline-webapps/>

2.18 Entrada de Comandos

Na construção da grande maioria dos jogos é muitas vezes imprescindível grande flexibilidade na gestão de entrada comandos. Esta necessidade se amplia na criação de jogos multiplataforma, em determinadas plataformas a entrada de comandos pode-se dar através de teclado, em dispositivos móveis através tela sensível ou sensor de movimentos. O HTML5 trata todos estes casos abstratamente na forma de eventos, os quais podem ser escutados através de *listeners*. JavaScript pode ser configurado para escutar cada vez que um evento ocorre seja um clique do mouse ou pressionar de uma tecla ou o mover de um eixo em um joystick.

Quando um evento de interação é disparado, um *listener* que esteja ouvindo a este evento pode invocar uma função e realizar qualquer controle desejado (Rocheleau, 2015). O teclado é um periférico comum no caso de jogos da Web, para existem os eventos *keyup* e *keydown* que representam uma tecla sendo solta e pressionada respectivamente. A 18 demonstra a cap-

tura do pressionar das setas em JavaScript. Cada número corresponde a um botão do teclado especificado através da tabela ASCII.

```

window.addEventListener('keydown', function(event) {
  switch (event.keyCode) {
    case 37: // Left
      Game.player.moveLeft();
      break;

    case 38: // Up
      Game.player.moveUp();
      break;

    case 39: // Right
      Game.player.moveRight();
      break;

    case 40: // Down
      Game.player.moveDown();
      break;
  }
}, false);

```

Figura 18: Utilização dos eventos do teclado

Fonte: <http://nokarma.org/2011/02/27/javascript-game-development-keyboard-input/>

2.18.1 Gamepad

Atualmente a única forma de utilizar gamepads na Web é através software de terceiros. E sua utilização é limitada restringida a emulação de mouse e teclado subutilizando seus recursos (W3C, 2015a).

Em 2015 a W3C introduziu uma API de Gamepads, atualmente em rascunho, que pretende solucionar estes problemas. A especificação define um conjunto de eventos e um objeto *Gamepad* que, em conjunto, permitem manipular os estados de um Gamepad eficientemente.

Existem eventos para atividades esporádicas como a conexão de desconexão de dispositivos. Já acontecimentos mais frequentes, como o pressionar de botões, é detectado através da inspeção dos objetos aninhados ao objeto principal (*Gamepad*). Cada objeto botão contém um atributo *pressed* que pode ser utilizado para saber se foi pressionado.



Figura 19: Objetos de um Gamepad

Fonte: <https://w3c.github.io/gamepad>

A figura 19 contém os objetos tradicionais de um Gamepad.

2.19 Orientação

Muitos dispositivos móveis contam com tecnologias que permitem detectar a orientação física e movimento como acelerômetros e giroscópios. Visto que são comuns em dispositivos móveis, jogos podem se beneficiar deste tipo de ferramenta para criar experiências peculiares para seus usuários. A W3C tem uma especificação em rascunho que abstrai as diferenças dos dispositivos e prove uma API padronizada para consumir informações de orientação. A especificação define dois eventos de DOM principais: *deviceorientation* e *devicemotion*.

O evento *deviceorientation* provê a orientação do dispositivo expressa como uma série de rotações a partir de um ponto de coordenadas locais (W3C, 2011). Para colocar claramente, o evento lançado em uma mudança de orientação provê variáveis correspondentes a eixos (alfa, beta, gama) que podem ser consultadas para determinar a orientação do dispositivo.

Já o evento *devicemotion* dispõe de informações de orientação, como o evento *deviceorientation*, com o adicional de informar a aceleração do dispositivo. A aceleração também é descrita em eixos e sua unidade de medida é metros por segundo.

2.20 HTTP/2

HTTP/2 é a última versão do protocolo de trocas de documentos entre cliente e servidor na Web. Quando um navegador requisita algum documento de esta requisição é geralmente feita

através do protocolo HTTP. O foco da nova versão do HTTP é performance; especialmente a latência percebida pelos usuários e o uso de rede e servidores (HTTP/2, 2015). A forma que os documentos trafegam do servidor para o cliente afeta diretamente a performance de uma aplicação, nos jogos esse fator se amplia devido a grande quantidade de arquivos geralmente necessários para montar uma cena de jogo.

Diferentemente do HTTP/1, HTTP/2 abre apenas uma conexão por servidor. Usando a mesma para trafegar todos os dados necessários para montar a página HTML. Dessa forma o HTTP/2 não necessita repetir as negociações de protocolo nem aguardar parado quando o limite de requisições que os navegadores suportam concorrentemente é atingido. Segundo Prall (2012) o limite concorrente de conexões por servidor é geralmente 5. Jogos, que muitas vezes trafegam muitos arquivos via rede, podem se beneficiar drasticamente desta características.

Outro benefício do HTTP/2 em relação a seu predecessor é que os cabeçalhos das requisições são comprimidos, diminuindo substancialmente o tamanho das requisições. HTTP/2 também permite mensagens do servidor para o cliente (*full-duplex*), o que abre um leque de novas oportunidades que antes só podiam ser obtidas através de requisições de tempos em tempos ao servidor (*pooling*) ou através de WebSockets.

HTTP/2 não recomenda a utilização de minificação nos arquivos, visto que não existe abertura de novas conexões, trafegar múltiplos arquivos se tornou barato. Dessa forma, algumas práticas, antes recomendadas no desenvolvimento Web, tem de ser revistas depois do HTTP/2.

Dentro do navegador as requisições HTTP/2 não convertidas em equivalentes do HTTP/1, mantendo a retrocompatibilidade em aplicações legadas. Sendo assim, os fatores que justifiquem a utilização do HTTP/1 são escassos e a tendência é observarmos cada vez mais aplicações utilizando com o HTTP/2.

2.21 Depuração

Depuração (*debug*) é o processo de encontrar e reduzir defeitos em um aplicativo de software ou mesmo hardware (Wikipedia, 2015b). As ferramentas de desenvolvimento do Google Chrome (*DevTools*) são uma boa opção para depurar aplicações feitas utilizando as tecnologias da Web. Dos depuradores para navegadores o do Google Chrome é o mais fácil de utilizar e já vem integrado nativamente junto com o software (Alexander, 2013). Chrome (2013) cita algumas funcionalidades do DevTools:

Provê aos desenvolvedores profundo acesso as camadas internas do navegador e aplicações Web. É possível utilizar o DevTools para eficientemente detectar problemas de layout, adicionar *breakpoints* em JavaScript, e pegar dicas de otimização de código.

Estas características são comuns na maioria dos depuradores dos navegadores como o do Internet Explorer (Visual Studio For Web) e do Firefox (FireBug) (Alexander, 2013).

Todavia, com o DevTools também é possível emular dispositivos alvo da aplicação ou conectar-se a um dispositivo Android real, ideal para testar ambientes multiplataforma. Para

conectar-se a um dispositivo real é necessário habilitar a depuração via USB no dispositivo, conectar o dispositivo via cabo, e rodar o aplicativo no dispositivo através do Google Chrome ou habilitando o modo depuração nos aplicativos híbridos. Dessa forma é possível visualizar a aplicação dentro Google Chrome do computador e utilizar as já mencionadas tecnologias do DevTools.

O DevTools costumava permitir depuração do elemento canvas. Sobre ele Kuryanovich et al. (2014) mencionou: com o inspetor WebGL é possível conferir os estados dos buffers, informações de texturas, frames individuais e outras informações úteis. Infelizmente este recurso foi removido e atualmente inspeção de canvas é um terreno escasso no ambiente Web, mais detalhes sobre o problema dos depuradores canvas serão tratados nos resultados.

Especificamente para WebGL existe o plugin independente WebGL Inspector. A ferramenta permite fazer inspeção da execução de métodos WebGL, observar o estado de texturas, buffers, controlar o tempo de execução entre outras funcionalidades ¹¹.

2.21.1 Source Maps

Source Maps é uma tecnologia que permite mapear códigos fontes minificados para seus respectivos originais. Este recurso é interessante pois permite que os desenvolvedores visualizem o código fonte em sua versão original, legível e fácil de depurar, enquanto entregam ao usuário final a versão minificada, otimizada para performance. Para o usuário final não há diferença pois Source Maps são carregados apenas se as ferramentas de desenvolvimento estão abertas e com a funcionalidade de Source Maps habilitada.

Source Maps foi desenvolvido como um trabalho em conjunto entre a Mozilla e Google em 2010, atualmente na terceira revisão o projeto é considerado estável e não recebe modificações na especificação desde 2013. Sendo suportado por diversas ferramentas de desenvolvimento como Google Chrome (DevTools) e Firefox.

A especificação prevê a existência de um arquivo *.map* o qual contém o mapeamento dos arquivos fonte e outros metadados. Este arquivo é referenciado pelos arquivos minificados de modo a permitirem o navegador a realizar o mapeamento. É possível informar o navegador a localização do arquivo de metadados seguindo a seguinte sintaxe:

```
//# sourceMappingURL=/path/to/script.JavaScript.map
```

Ou através de cabeçalhos HTTP como demonstrado abaixo.

```
X-SourceMap: /path/to/script.JavaScript.map
```

Para criar arquivos de Source Maps é possível utilizar ferramentas especializadas ou integrá-los ao processo de *build* como Grunt ou Gulp. A biblioteca <https://github.com/benvanik.github.io/WebGL-Inspector/>

¹¹Mais informações sobre o WebGL Inspector podem ser encontradas no seguinte endereço <http://benvanik.github.io/WebGL-Inspector/>

UglifyJS2 é uma ferramenta de minificação capaz de gerar Source Maps. Após ter o jogo ter sido depurado e seus erros minimizados, pode-se disponibilizar para ser consumido por seus usuários finais.

Abaixo serão abordadas algumas formas de disponibilizar jogos Web em ambientes multiplataforma.

2.22 Disponibilização da Aplicação

Os aplicativos puramente em HTML não requerem instalação e funcionam apenas acessando o endereço através de um navegador. Vahatupa (2014) cita à respeito de aplicações Web "por não requererem instalação, sua distribuição é superior ao estilo convencional de aplicações desktop".

Conquanto, se o objetivo é fornecer uma experiência similar aos demais aplicativos mobile ou integrar um sistema de compras, geralmente feitos através de um mercado como o GooglePlay, pode-se adotar a alternativa híbrida criando um pacote para o software.

O PhoneGap é uma tecnologia que permite encapsular um código em HTML e disponibilizá-lo nativamente. Não obstante, para empacotar os aplicativos localmente é necessária configuração substancial e só é possível empacotar para IOS em um computador da Apple. Alternativamente, o PhoneGap disponibiliza um serviço de empacotamento na nuvem que soluciona estes problemas o PhoneGap Build.

Através do PhoneGap Build pode-se carregar um arquivo zip, seguindo determinado formato, o qual contenha os arquivos escritos com as ferramentas da web, que o PhoneGap Build se responsabiliza por empacotá-los nos formatos requeridos para serem instalados em Android, IOS e Windows Phone.

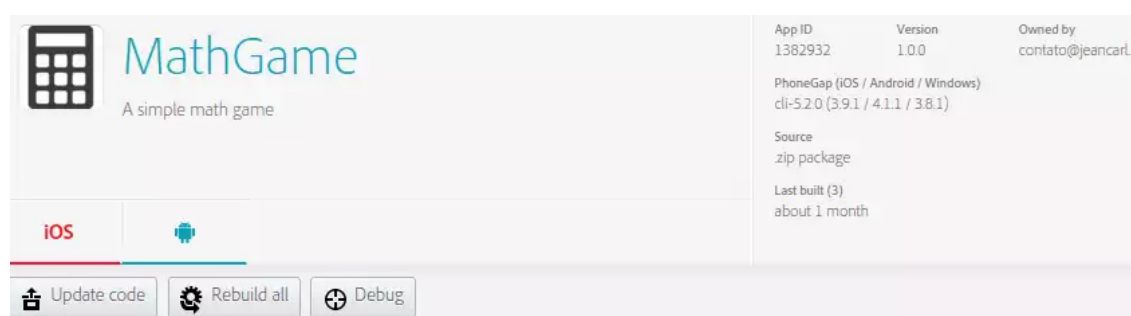


Figura 20: Protótipo no PhoneGap Build

Fonte: <https://build.phonegap.com/apps>

Monony (2015) descreve os passos que são feitos pelo PhoneGap Build para disponibilizar a aplicação nativamente:

- É criada uma aplicação nativa utilizando a WebView da plataforma;
- Todos os recursos da aplicação são armazenados dentro da aplicação nativa;

- O PhoneGap carrega o HTML dentro da WebView;
- A WebView mostra a aplicação para o usuário;

No endereço <https://github.com/phonegap/phonegap-start> encontra-se um *template* no formato requerido pelo PhoneGap Build que pode ser utilizado para começar aplicações que serão servidas através da solução.

Além do PhoneGap existem outras formas de empacotar aplicações para múltiplas plataformas. Por exemplo, através de plataformas de desenvolvimento como o Sencha Touch e Ionic, AppGyver ou Xamarin. Entretanto, estas plataformas muitas vezes utilizam o PhoneGap internamente ou, apesar de criar aplicações para Web, não comportam o desenvolvimento com tecnologias da Web.

3 PROJETO

Este capítulo tem por objetivo detalhar os aspectos do desenvolvimento do protótipo. Sua mecânica, funcionamento, decisões tecnológicas, diagramas, entre outras características serão tratadas aqui.

3.1 Mecânica

Para a análise prática das limitações foi escolhido um jogo de matemática simples. Consistindo na geração de equações cada qual com uma resposta candidata. Cabe ao usuário informar se o resultado apontado pelo jogo está correto ou não. A cada resposta dada, o nível de complexidade da equação cresce. O tempo é um fator determinante no resultado do jogo pois quanto mais rápido o jogador acertar se a afirmação está correta ou não mais pontos ele receberá. Esta categoria de jogo foi selecionada por ter profundidade, oferecendo a possibilidade de explorar diversos recursos do HTML, e criar melhorias incrementais. E também por oferecer uma dificuldade técnica não tão desafiadora, visto que não disponho de experiência profunda no desenvolvimento de jogos em geral.

Jogos como o Math Workout e o Countdown para Android tem uma temática similar. Conquanto, o Math Workout não apresenta a resposta, sendo o papel do usuário computar a equação e digitar o resultado. Já o jogo Countdown apresenta um número final e requer que o usuário determine a equação que resultou no valor a partir de um dado conjunto de números e operadores. O jogo desenvolvido para o protótipo parece ter uma melhor jogabilidade em dispositivos móveis que ambos os jogos acima citados pois não requer a presença de um teclado numérico. Os botões de verdadeiro ou falso contém todas as possibilidades de interação com a aplicação. A figura 21 demonstra a interface contendo os botões descritos.



Figura 21: Interface requerida

3.2 Requisitos

Abaixo estão detalhados os requisitos que uma mecânica como a descrita acima deve prover.

3.2.1 Requisitos funcionais

As funcionalidades requeridas para o jogo são as seguintes:

- O sistema deve prover equações matemáticas de dificuldade crescente para o usuário informar se estão corretas ou não;
- O sistema deve pontuar as respostas dadas com maior agilidade com uma pontuação maior que as respondidas com menor agilidade;
- O sistema deve apresentar a colocação da partida do usuário em comparativo com seu histórico.

3.2.2 Requisitos não funcionais

Outros aspectos requisitados mas que não fazem parte da regra de negócio:

- O sistema deve ser desenvolvido utilizando as ferramentas da web.

- O sistema deve funcionar para a plataforma desktop e Android.
- O sistema deve ser desenvolvido sem a utilização de nenhuma biblioteca ou framework.

3.3 Modelagem

A figura 22 apresenta o diagrama de classes simplificado ¹.



Figura 22: Diagrama de classes simplificado

3.4 Desenvolvimento

O desenvolvimento se deu com uma postura de melhoria progressiva criando incrementos funcionais periodicamente. O processo assimilou-se bastante ao método ágil SCRUM, com algumas diferenças notáveis, os tamanhos dos sprints foram flexíveis e não houve reunião de retrospectiva, devido a não haver uma equipe de desenvolvimento. Criou-se primeiramente um produto minimamente viável (MVP) e, a partir dele, novos recursos foram sendo adicionados a fim de melhorar a experiência do usuário. Os passos que compuseram os *sprints* do processo serão descritos abaixo.

Antes focar no desenvolvimento do software, foi configurado um ambiente automatizado com o Grunt, com plugins de minificação e empacotamento de JavaScript e CSS ². O Grunt foi configurado para disponibilizar a aplicação pronta para distribuição dentro do diretório *dist/web*. Uma vez funcionando a distribuição Web, foi feita a integração para o PhoneGap Build. Para tanto foi utilizado um *makefile* que simplesmente copia os arquivos de distribuição

¹Nos anexos pode-se encontrar a versão completa do diagrama de classes

²Para mais informações sobre o Grunt veja os apêndices

da Web, gerados pelo Grunt, e o arquivo de metadados do PhoneGap Build, e cria a árvore de arquivos padronizada que o PhoneGap Build requer. Além do ambiente automatizado, foi utilizado um servidor Web Apache através do docker para testar a aplicação.

Iniciei o desenvolvimento na plataforma Desktop pois esta contém o maior número de ferramentas de desenvolvimento nativas que não requerem configuração especial. O primeiro passo, na concepção do software propriamente dito, foi a criação do documento HTML. Para a representação das telas do jogo foram utilizados elementos *div* dentro do arquivo *index.html*. Conter todas as telas em único HTML caracteriza o jogo como uma aplicação de página única (*Single Page Application*). Alternativamente, poderia se depender de um servidor para mandar as páginas prontas, mas isso distribui a complexidade do sistema para tecnologias do lado do servidor, distanciando-se da proposta de um jogo construído exclusivamente com as tecnologias da Web.

Aplicativos SPA introduzem novos desafios. Por exemplo, o botão de voltar perde sua utilidade, visto que não se está trafegando de uma página a outra, somente em partes de uma mesma página. HTML5 provê uma API em JavaScript para manipular o histórico que pode resolver este problema. Entretanto, no protótipo não adicionamos esta funcionalidade pois a quantidade de telas é realmente baixa e os benefícios introduzidos seriam pequenos.

Seguindo a construção do documento HTML veio o desenvolvimento de um CSS simples que comporta a visualização em múltiplos dispositivos. Para tal, foram utilizadas posições e tamanhos relativos. Por exemplo, a largura de cada tela é 98% do tamanho total disponível no dispositivo. Já o tamanho da fonte do quadro principal, é duas vezes o tamanho da fonte normal (2em).

Sem a ajuda de bibliotecas especializadas, o processo de criação da interface não é trivial. Apesar da interface ser simples, fazer os elementos se alinharem em múltiplos tamanhos de telas não foi fácil, e pode se tornar um problema substancial para interfaces realmente complexas. As figuras 23 e 24 apresentam a tela principal do jogo em um dispositivo móvel e em um desktop, respectivamente.



Figura 23: Interface do jogo com equação sendo apresentada



Figura 24: Interface do jogo com equação sendo apresentada no desktop (1440x900)

Após a concepção do CSS deu-se início ao desenvolvimento da lógica das páginas em JavaScript. O primeiro passo consistiu em habilitar o funcionamento de múltiplas telas em um único arquivo HTML através de JavaScript. Os botões que levam a outras telas, quando clicados, simplesmente escondem todas as seções e por fim carregam a que estão configurados

para mostrar.

Ao iniciar a execução do JavaScript todas as *divs* que representam telas são escondidas e a *div* de carregamento de recursos é mostrada em seguida. O objetivo desta *div* é não deixar o usuário sem feedback enquanto os recursos necessários para a utilização do jogo não estão disponíveis. Todavia, o carregamento é realmente rápido e o usuário, a não ser que dependa de uma rede muito lenta, geralmente não vê a tela. No final do carregamento de todos os recursos é disparado o evento *window.onload*, neste momento carrega-se a *div* da partida e dá-se início a mesma.

Durante esta fase do desenvolvimento também foram adicionados *listeners* aos demais elementos interativos do jogo, como no clicar nas configurações e nos botões de certo e errado da página principal. Deixando as telas prontas para receber as funcionalidades propriamente ditas.

Com esqueleto da aplicação definido foi introduzida a lógica de negócio. A figura 22 apresenta a relação entre as classes do jogo. De toda a regra do jogo, a classe *Match* é a mais importante. Ela simboliza uma partida dentro do jogo, é na classe *Match* que as informações de quantas equações existem, quantas foram acertadas, o tempo total da partida, bem como a pontuação atual do usuário são armazenadas. A cada iteração do laço do jogo uma nova pergunta é gerada pela classe *Match* e espera-se por uma resposta. Quando a resposta é dada, a classe *Match* computa se ela está correta ou não e o tempo que levou para chegar ao resultado. Quando não existem mais perguntas para serem processadas é lançado um evento de final de partida onde o resultado pode ser processado pelas demais classes do jogo.

O cálculo da pontuação se dá por uma operação matemática simples. Ao final de uma partida é computado o total de questões acertadas, multiplicado por dez, dividido pelo total de respostas, menos o total de respostas certas adicionando-se 20% do tempo da duração em segundos. A figura 25 apresenta o código utilizado para o cálculo acima descrito.

```
parseInt(
  (this.rightAnswered * 10 ) / (
    (this.answersTotal - this.rightAnswered)
    + (this.duration*0.2))
)
```

Figura 25: Cálculo da pontuação do jogador

A construção da classe *Match* disponibilizou o comportamento principal do jogo; entretanto, nesta etapa não havia um gerador de equações. O jogo contava apenas com uma coleção de equações preestabelecidas que eram selecionadas aleatoriamente a cada turno. Adiar a construção do gerador de equações se provou uma boa escolha pois possibilitou que o desenvolvimento se focasse em outros aspectos importantes, como a elaboração do laço do jogo, ranking, configurações, entre outros.

O ranking serve para armazenar o resultado de cada partida do jogador possibilitando

uma percepção de histórico de performance. Os dados são armazenados em Local Storage, escolhido por ter uma API simples. Arquiteturalmente falando, IndexedDb se encaixaria bem na aplicação por ter uma interface chave valor, ideal para um ranking, onde as chaves poderiam ser as posições do usuário. Entretanto, a interface totalmente orientada a eventos do IndexedDb introduz uma camada de complexidade desnecessária para os casos simples. Por conseguinte, visto que os requerimentos do protótipo não demandam grande performance ou armazenamento massivo de dados, a opção modesta, Local Storage, foi preferida. Para colocar claramente, a classe ranking é simplesmente uma interface para converter partidas, armazenar e recuperar estas informações em Local Storage. A figura 26 demonstra as informações do resultado de uma partida, na qual as informações de ranking se encontram.

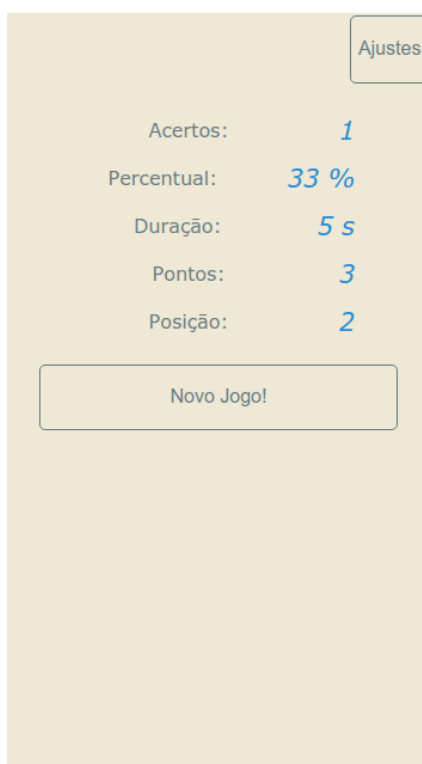
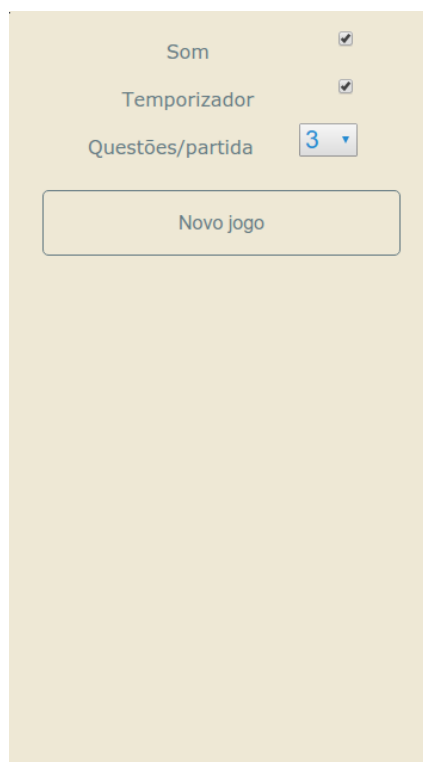


Figura 26: Resultado de uma partida

O objeto *Settings*, assim como o Ranking, utiliza Local Storage e provê uma interface para armazenar e recuperar preferências sobre o jogo. Cada campo editável na tela de configurações contém *listeners* prontos para registrar no objeto *Settings* cada mudança que ocorrer em seus estados. Os objetos que utilizam estas configurações também o fazem através do objeto Settings, nunca acessando configurações diretamente em Web Storage. Dessa forma a validade das configurações é garantida e a migração para uma forma de armazenamento diferente é possível com relativa facilidade. A figura 27 demonstra a tela de configurações do jogo.



Som ☒

Temporizador ☒

Questões/partida 3

Novo jogo

Figura 27: Configurações do jogo

Implementados estes mecanismos essenciais para o jogo, pude me focar no ponto central do negócio, e possivelmente o mais complexo: a geração de equações. A geração das equações é feita randomicamente e envolve duas classes: *Statement* e *StatementGenerator*. O objeto *Statement*, é responsável por armazenar as informações de uma equação, à dizer: a afirmação sendo feita e se seu resultado está correto ou não. O objeto *StatementGenerator* é responsável por gerar instâncias da classe *Statement*.

A classe *StatementGenerator* conta com um método *getStatement* que realiza o processamento para gerar um novo *Statement*. Esta função recebe como argumento um inteiro que simboliza a dificuldade da equação. A cada resposta dada pelo usuário, armazenada no objeto *Match*, a dificuldade é incrementada e repassada para o gerador. O valor da dificuldade é utilizado internamente no *StatementGenerator* para selecionar quais operadores serão utilizados e o tamanho do multiplicador dos números que compõem as equações. Para colocar claramente: as equações são geradas através da randomização de valores e operadores (com suas respectivas dificuldades processadas). Segue-se com a execução da equação para determinar seu resultado e a geração, em 50% dos casos, de um valor errado de modo que a resposta não seja sempre correta.

O objeto *StatementGenerator* reside como membro de classe de *Match* e a cada interação com o usuário uma nova equação é gerada por ele e armazenada no atributo *currentStatement* do objeto *Match*. Ao final das iterações com o usuário o evento *endOfMatch* é lançado onde os pontos são computados, armazenados e mostrados para o usuário. Neste ponto é possível começar outra partida, reiniciando o processo.

Estas classes comportam os requisitos funcionais do jogo. Entretanto, após um período de uso, foi identificado que muitas vezes o usuário começa uma partida e não está prestando atenção na tela, o que acarreta na perda de pontos no período inicial da partida. Para reduzir este problema foi adicionada a classe *Countdown*, que é basicamente um temporizador regressivo que demarca o início de cada partida, notificando o usuário quanto falta para a partida iniciar. O temporizador foi desenvolvido em canvas e apresenta 4 demarcações desenhadas a cada 90 graus, formando um círculo com os números decrescentes no centro. A figura 28 demonstra o contador prestes a iniciar uma nova partida.



Figura 28: Contador em Canvas

Para melhorar a usabilidade em desktops foram adicionados controles de teclado além dos já presentes botões na interface, possibilitando que o usuário utilize o teclado além do mouse. Quando o usuário utilizar a seta para esquerda ou direita os botões Sim ou Não, respectivamente, são clicados através de JavaScript. Simular o clique ao invés de acionar duas vezes o tratamento das escolhas de resposta foi uma boa estratégia pois centralizou o controle, evitando duplicação de código.

Nos dispositivos móveis, com intuito de melhorar a experiência, foram adicionadas vibrações para as respostas erradas. A API de vibração é trivial e sua utilização possibilita uma experiência mais profunda com a aplicação, sendo uma adição de bom custo/benefício. Se o usuário desejar utilizar o mouse no desktop, foram adicionadas transformações do CSS quando o ponteiro se encontra acima dos botões. Outrossim, com o intuito de incrementar o feedback visual, os botões de certo e errado tornam-se verdes ou vermelhos dependendo se o usuário acertou ou não a questão dada. No quesito feedback auditivo foram adicionados sons através da

API de áudio do HTML para as respostas corretas e incorretas. Com estes detalhes pormenorizados, fica visível que grande parte do tempo de desenvolvimento de jogos multiplataforma é utilizado em fornecer uma experiência rica em interatividade, que faça uso das peculiaridades de cada plataforma.

Durante todo o processo de desenvolvimento e validação, a aplicação foi testada em dispositivos móveis através do DevTools do Google Chrome. Utilizando as opções de inspeção de dispositivos remotos, o DevTools permitiu conferir se o jogo funcionava corretamente utilizando um dispositivo real, na versão alvo do Android, enquanto no conforto de um ambiente de desenvolvimento desktop.

3.4.1 Performance

A performance, mesmo sem otimizações ficou razoável. O tempo de carregamento no Google Chrome desktop em uma rede 4G, comum no Brasil, ficou em 1.4 segundos em média. Esta métrica foi extraída através da ferramenta de depuração do Google Chrome que fornece a possibilidade de simular velocidades de redes.

Utilizando os arquivos minificados, gerados pelo Grunt, o tempo médio de carregamento ficou em 1.1 segundos. Uma diferença substancial (22%), dada a quantidade pequena de arquivos minificados (8 arquivos JavaScript e um arquivo CSS).

A seguir serão tratadas algumas otimizações para jogos descobertas através do desenvolvimento e pesquisa bibliográfica deste trabalho.

3.5 Otimizações para jogos

Navegadores tentam otimizar a experiência de navegação definindo um conjunto de regras e configurações razoáveis para a maioria dos casos. Não obstante, nem sempre estes valores padrões são as melhores opções no contexto de jogos. Abaixo seguem algumas otimizações nas tecnologias da Web, direcionadas aos jogos, que foram identificadas através da revisão e desenvolvimento do protótipo.

3.5.1 CSS

Scroll é um recurso interessante para longas páginas de texto, o mesmo não se pode dizer à respeito de jogos. Principalmente aqueles dependentes de contato com a tela, pois no contato a tela pode mover-se e desconcentrar o usuário. Para remover este comportamento deve-se utilizar o *overflow: hidden;* no seletor do corpo do documento (*body*).

A barra de endereço é outro recurso de pouca utilidade no contexto de jogos, e algumas vezes um empecilho para jogos em dispositivos móveis, devido ao limitado tamanho da tela. Para desabilitar a barra em dispositivos da Apple pode-se utilizar a seguinte configuração:

```
<meta name="apple-mobile-web-app-capable" content="yes" />
```

Segundo Mobilexweb (2013b) o Google Chrome, a partir da versão 31, adicionou suporte a esta notação, inclusive o prefixo Apple, mas as intenções são que o prefixo seja removido nas próximas versões. Para os demais dispositivos não existe meio oficial de esconder a barra de endereço. No entanto, alguns sites recomendam a solução descrita abaixo:

```
<body onload="setTimeout(function() {window.scrollTo(0, 1)}, 100)">
</body>
```

Apesar de não fazer parte de nenhuma especificação, a maioria dos navegadores implementa a possibilidade de desativar a seleção de texto na tela. Em jogos essa possibilidade é útil, pois não é natural, e possivelmente frustrante, selecionar texto ao tentar interagir com a interface. Kuryanovich et al. (2014) cita que desabilitar a seleção de texto em jogos é uma otimização importante para a experiência do usuário. Para fazê-lo, pode-se utilizar as regras CSS demonstradas abaixo.

```
-moz-user-select: none;
-webkit-user-select: none;
-ms-user-select: none;
```

3.5.2 JavaScript

A seguir serão descritas algumas otimizações de JavaScript no contexto de jogos.

3.5.2.1 Modo estrito

Um recurso interessante do JavaScript é seu modo estrito, este faz um conjunto de modificação na semântica do interpretador de modo que alguns recursos suportados, mas propensos a problemas, sejam desabilitados. Um exemplo de sintaxe válida propensa a erros é a utilização variáveis não prefixadas pela palavra chave *var*, que torna as variáveis em variáveis globais.

O modo estrito pode ser entendido como uma variante mais rígida do JavaScript. Este mecanismo pode ser habilitado utilizando o termo *"use strict;"* nos cabeçalhos de arquivos ou funções. Este controle localizado permite que código não estrito trabalhe em conjunto com código estrito, característica conveniente para a utilização em sistemas legados.

3.5.2.2 Funções imediatamente invocadas

Um problema comum de sistemas complexos em JavaScript é que muitos objetos vivem em ambiente global. Isso pode causar uma coleção de problemas, desde conflitos de nomes à sobrecarga de variáveis. Para contornar esse problema pode-se utilizar as funções imediatamente invocadas IFE (*Immediately invoked function expression*).

```

(function() {
    'use strict';

    function bar() {
        return 'foo';
    }

    window.bar = bar;
})();
window.bar();

```

Figura 29: Exemplo de função imediatamente invocada.

A figura 29 demonstra a utilização deste padrão. As funções definidas no mesmo nível que `bar` não estarão no contexto global, a não ser que seja especificado diretamente, e não sofrerão conflitos de nomes e outros problemas relativos ao contexto global.

Outro fator importante na construção de jogos em JavaScript é a otimização do laço do jogo. Para escrever o laço é possível utilizar as funções do JavaScript *setTimeout* ou *setInterval*. Não obstante, a forma mais recomendada é utilizar a função *requestAnimationFrame*. Esta função reduz ou completamente desabilita a execução do laço enquanto o usuário está em outra aba. Por conseguinte, o consumo de bateria é reduzido, uma característica importante para dispositivos móveis.

3.5.2.3 HTML

Um problema que jogos em HTML sofrem é a demora no carregamento da grande quantidade de recursos que geralmente precisam estar em memória para o jogo funcionar. Para minimizar este problema, muitos jogos utilizam uma tela de carregamento enquanto os recursos são adquiridos. Existem, todavia, algumas formas alternativas de reduzir este problema, como utilização de cache, CDN's, minificação, entre outros.

Uma funcionalidade do HTML interessante para otimizar o tempo de rede é o pré carregamento de recursos (*Link Prefetching*). Esta tecnologia possibilita que o navegador, em seu tempo livre, adquira recursos que provavelmente serão necessários em um futuro próximo. Nem todos os recursos necessitam ser pré carregados, mas uma impressão muito superiora é criada se os recursos estão imediatamente disponíveis quando uma nova fase é carregada (Seidelin, 2010, p. 39).

A seguir serão apresentadas as limitações do HTML encontradas durante o desenvolvimento do protótipo e pesquisa relacionada.

4 RESULTADOS

Muitas das limitações dos jogos multiplataforma não são problemas específicos dos jogos, mas aplicam-se a todos tipos de software (Bruins, 2014, p. 3). Alguns problemas são inerentes da categoria multiplataforma Morony (2013, p. 7) afirma que é geralmente muito mais complexo obter aparência nativa, funcionalidade e performance em aplicações multiplataforma. Outros problemas derivam-se dos dispositivos ou da tecnologia atual.

A abaixo constam os problemas e limitações do HTML5, aplicáveis, mesmo que não exclusivamente, aos jogos encontrados durante a pesquisa e concepção do protótipo. Quando possível, buscou-se apresentar as limitações na mesma ordem das tecnologias estudadas na revisão bibliográfica. Não obstante, algumas limitações foram tratadas em partes separadas visto que se aplicam a várias das tecnologias da Web. Para melhor organizá-las junto ao texto, foram adicionados códigos as limitações que seguem o seguinte padrão: LMT + número da limitação.

4.1 HTML

Barnett (2014) afirma que (LMT 1):

Enquanto o HTML é desenvolvido muitas das funcionalidades disponibilizadas são testadas em um pequeno conjunto de navegadores para um pequeno conjunto de versões. Isso acarreta em suporte inconsistente. A forma mais segura de garantir suporte é testando em todas as versões alvo, entretanto essa solução não é prática.

Este não é um problema exclusivo dos navegadores. Segundo Mobilexweb (2013a), da versão 4.4 do Android em diante a WebView mudou de um projeto local para utilizar o Chromium. Entretanto, grande parcela dos usuários Android ainda utilizam o sistema antigo o que força os desenvolvedores suportarem ambas as versões. É razoável afirmar que para o início de 2016 um terço dos usuários de Android ainda utilizem a versão antiga da WebView (Mobilexweb, 2013a).

O Crosswalk é uma tecnologia que pode resolver parcialmente o problema de suporte de funcionalidades em diversas versões de dispositivos. O Crosswalk funciona disponibilizando, juntamente com a aplicação, uma versão recente do Chromium que será responsável por rodar a aplicação. Desse modo todos os dispositivos que utilizam o pacote gerado pelo Crosswalk vão rodar na mesma versão de navegador. Não obstante, o Crosswalk só é suportado para plataforma

Android ¹.

4.2 CSS

É muito custoso desenvolver interfaces que pareçam nativas para cada dispositivo sem a utilização de plugins e outras ferramentas auxiliares. No protótipo foi utilizada uma estilização simples a qual pode ser interessante na Web. Conquanto, nos dispositivos móveis, o layout criado para o jogo é muito diferente da experiência normal destes aparelhos. Sendo que cada sistema operacional conta com regras próprias para definir a experiência do usuário ² (LMT 2).

Uma solução para este problema é utilizar frameworks como o jQuery Mobile e Kendo UI Mobile ³. Estes frameworks permitem criar elementos típicos de interfaces mobile como listas com scroll, botões e transições, com aparência nativa, de forma relativamente fácil (Monony, 2015).

Em alguns casos o tamanho das telas pode ser um fator limitante, como por exemplo em jogos de estratégia. Estes jogos geralmente necessitam mostrar uma vasta quantidade de informações, neste contexto, jogadores com telas menores podem sair em desvantagem (LMT 3). Em termos gerais, pode-se mitigar este problema utilizando estratégias de design focadas nas telas menores (*mobile first*).

Na perspectiva do desenvolvedor, pode-se lidar com múltiplos tamanhos de tela trabalhando com tamanhos relativos via CSS. Todavia há casos, como o dos botões de certo e errado do protótipo, em que as proporções ficam exageradas. Nestas ocasiões, utilizar controles como o *max-width* e *min-width* é uma solução conveniente. No protótipo os botões de certo e errado foram inicialmente configurados para preencherem 40% da tela cada. Em dispositivos móveis este valor é sensato, mas em desktops com grandes resoluções a largura fica exagerada. Para resolver este problema foi configurado via CSS que a largura dos botões tenham 40% do tamanho enquanto esse valor for menor de 300 pixels, via regra *max-width*.

Outro fator problemático do CSS é a presença de prefixos (LMT 4). Tecnologias experimentais do CSS geralmente levam o nome do distribuidor como prefixo da propriedade. Utilizar as mesmas regras em CSS com prefixos diferentes para suportar diversos motores de renderização é um processo entediante e propenso à duplicação e erros. A biblioteca *-prefix-free* é uma possível solução para este problema, detectando automaticamente quando prefixos são necessários e adicionando-os em tempo de execução ⁴.

¹Na seção B.2 dos Apêndices o Crosswalk é tratado com mais detalhes

²O Android utiliza o projeto Material design <https://www.google.com/design/spec/material-design> como base para suas regras de como construir interfaces. Já o IOS utiliza um conjunto de regras próprias que podem ser encontradas neste endereço <https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/>

³Mais informações sobre o Kendo UI e jQuery Mobile podem ser encontradas nos apêndices

⁴Mais informações sobre a biblioteca *-prefix-free* podem ser encontradas nos apêndices

4.3 JavaScript

O JavaScript, por ser uma linguagem desenvolvida por consenso, tem um ciclo de vida de atualizações demorado; pois necessita que todos os envolvidos na especificação entrem em acordo (LMT 5). Com a especificação pronta, outra fase demorada é a adoção das tecnologias nos navegadores (LMT 6). O site <https://kangax.github.io/compat-table/es6/> contém uma lista do suporte aos recursos do ECMAScript nos motores JavaScript. Através desta lista pode-se observar que as novas funcionalidades do ECMAScript 6 contém pobre suporte na grande maioria dos motores, especialmente quando referente a funcionalidade de subclasses.

Por ser uma linguagem interpretada, erros tem de descobertos rodando a aplicação (LMT 7). Alternativamente, se JavaScript fosse compilado, vários problemas poderiam ser capturados e informações úteis reveladas antes de se testar (Morony, 2013, p. 12). Uma forma de solucionar este problema é utilizando alguma linguagem compilada através de Web Assembly.

Arquivos JavaScript geralmente necessitam ser minificados, reunidos e ofuscados. Muitas vezes esses processos precisam ser executadas durante o desenvolvimento, e sem a utilização de automatizadores como o Grunt Watch, o processo torna-se entediante bem como propenso a erros.

Segundo McCann (2012), um problema do JavaScript é que não é possível transferir métodos de objetos através de sistemas via WebSockets, somente dados (LMT 8). Uma forma de contornar este problema é utilizar funções para converter os dados em objetos em cada ponta do processamento, mas isto adiciona complexidade ao software.

Garisel; Irish (2011) cita à respeito de scripts requerendo informações de estilo durante o processo de parse: "se o estilo ainda não foi carregado o script vai utilizar informações erradas, causando uma série de problemas" (LMT 9). Pode-se contornar este problema executando scripts exclusivamente ao final da renderização do HTML.

Por ser uma linguagem orientada a protótipos, JavaScript dá grande poder e flexibilidade para os desenvolvedores. Não obstante, um possível problema desta característica é quanto a manutenibilidade de código (LMT 10). Hasan et al. (2012) afirma que:

A flexibilidade de manutenção de código em JavaScript é muito dependente do expertise da pessoa que está escrevendo o código. Escrever código "sustentável" em JavaScript é mais difícil se comparado com Java ou C# mas, aplicando bons padrões de design, é bem possível escrever bom JavaScript.

Além de bons padrões de design, outra forma de minimizar os problemas de manutenibilidade de JavaScript é utilizando guias de estilo de escrita. Arora (2015) afirma que quando não existem padrões de escrita todos acabam escrevendo código que apenas os autores conseguem entender. O endereço <http://noeticforce.com/best-JavaScript-style-guide-for-maintainable-code> contém um comparativo entre os guias de estilo mais comuns para JavaScript.

4.3.1 Sistema de tipos

O sistema de tipos do JavaScript também é problemático. Erros numéricos resultam no valor NaN (*not a number*). Sendo que todas as operações com NaN como operadores irão retornar outro NaN. Isso torna a depuração de erros desnecessariamente complexa (Kuryanovich et al., 2014) (LMT 11). Uma saída para este problema é checar constantemente os tipos de forma a assegurar-se que eles continuam válidos. Entretanto, este processo pode se tornar burocrático e uma forma adicional de criar erros.

Outra limitação do JavaScript é quanto a checagem de tipos. Ao testar o tipo de uma variável vazia com a função *typeof* o JavaScript retorna como se a variável fosse um objeto (LMT 12).

```
function is(type, object) {
  type = type[0].toUpperCase + type.slice(1);
  return Object.prototype.toString.call(object)
    === '[object ' + type + ']';
}
```

Figura 30: Função para testar tipos que funciona como o esperado.

Fonte: <http://www.slideshare.net/fdaciuk/javascript-secrets-front-in-floripa-2015>

A figura 31 demonstra uma solução possível para remediar o problema dos testes de tipos em JavaScript. A diferença desta função é que ela converte a variável que se está testando para sua representação em texto, a qual contém o nome do tipo escrito por extenso. Dessa forma, utilizando-se deste valor pode-se deduzir o tipo da variável corretamente.

4.3.2 Performance

Apesar da performance ter notavelmente melhorado, ainda é geralmente menos eficiente produzir animações em JavaScript do que utilizando transições e animações do CSS, que por sua vez são mais otimizadas e aceleradas via hardware (Kuryanovich et al., 2014) (LMT 13). O Web Assembly poderia resolver este problema, substituindo o JavaScript para os casos onde grande performance é necessária. Mas para isso o Web Assembly precisa evoluir em sua especificação e implementações.

4.3.3 Fullscreen

Não existe forma padronizada de detectar se uma aplicação está em tela cheia ou não através de JavaScript (LMT 14). No final de 2015 um rascunho de especificação (*Fullscreen API*) foi iniciado, que irá permitir requisitar a utilização de Fullscreen e detectar elementos neste modo. Não obstante, é uma especificação muito nova e nenhum navegador a suporta completamente.

No IOS existe a possibilidade de consultar variável *navigator.standalone* para identificar se a aplicação está em tela cheia. Mobilexweb (2013b) recomenda a utilização do seguinte trecho de código para determinar se a aplicação está em modo tela cheia para os demais navegadores:

```
navigator.standalone = navigator.standalone
|| (screen.height-document.documentElement.clientHeight<40)
```

Figura 31: Teste de tela cheia

Fonte: <http://www.mobilexweb.com/blog/home-screen-web-apps-android-chrome-31>

4.4 SVG

Uma das grandes vantagens do SVG é que ele é uma linguagem de marcação; por conseguinte, se integrando bem com as demais tecnologias da Web. Contudo, isso também implica em problemas de performance para arquivos muito grandes pois os elementos são manipulados via DOM (LMT 15). Segundo Kuryanovich et al. (2014) a grande desvantagem do SVG é que quanto maior o documento mais lenta a renderização.

Controle refinado sobre posicionamento também é um problema com SVG. Kuryanovich et al. (2014) afirma que é muito difícil atingir a perfeição na posição dos pixels, por ser uma linguagem vetorizada (LMT 16). Essa característica acaba limitando a aplicabilidade do SVG como renderizador de jogos para os casos não muito complexos, onde posicionamento não seja uma fator crucial.

4.5 Canvas

Segundo (Kuryanovich et al., 2014), os aspectos negativos do canvas são que a performance varia de plataforma para plataforma e não existe implementação nativa para animações.

4.5.1 Performance

O problema de performance (LMT 17) pode ser parcialmente remediado com o FastCanvas. FastCanvas é uma implementação nativa em C++ do canvas para Android que roda separadamente do JavaScript. Devidas as características acima citadas FastCanvas é substancialmente mais rápido do que a tag canvas para navegadores Android. Todavia, o FastCanvas não suporta a especificação do canvas completamente, e existem algumas diferenças no comportamento entre o canvas original e o FastCanvas.

4.5.2 Integrações

A implementação de animações de fato não existe no canvas (LMT 18). Similarmente carece-se de integração com as demais tecnologias da Web (LMT 19). Os desenhos do canvas

não podem ser acessados via DOM nem serem manipulados via CSS. Todas as modificações necessárias devem ser feitas através de JavaScript. No protótipo, especificamente na classe *Countdown*, foi necessário adicionar regras de estilo via JavaScript para a interação com o canvas ficar completa.

CSS já conta com definições quanto a animações e a manipulação de elementos do canvas através do DOM facilitaria uma gama de situações. Por exemplo, seria possível utilizar os eventos do DOM para capturar interações do usuário através de objetos utilizados no canvas. Se um retângulo fosse clicado o evento seria lançado a partir dele. Controle similar é implementado atualmente acessando as coordenadas do canvas onde uma interação aconteceu e fazendo o processamento com aquela determinada área.

Felizmente a integração entre o canvas e as demais tecnologias da Web está começando a acontecer. A interoperabilidade entre o Path2D do canvas e a notação SVG é uma iniciativa na direção correta, tornando ambas tecnologias cada vez mais relevantes e dinâmicas.

Outra característica peculiar, descoberta durante o desenvolvimento do protótipo, é que o Canvas pode gerar resultados inesperados se seus tamanhos de elemento e tela diferirem de formas específicas. Em suma, existem dois tamanhos, o tamanho do elemento e da superfície de desenho. Quando o tamanho do elemento é maior do que o da superfície de desenho, o documento escala a superfície de desenho para preencher o elemento, o que pode gerar resultados inesperados.

4.6 WebGL

Como a especificação do WebGL é baseada na versão otimizada para dispositivos móveis do OpenGL não é possível utilizar muitos recursos especiais disponíveis para os ambientes desktop (LMT 20). Como a definição de caminhos com a função *glBegin* e a utilização de pontos flutuantes para calcular coordenadas de vértices (KAMACI, 2010). Em outras palavras, a especificação do WebGL é cortada por baixo. Visto que alguns dispositivos não tem performance utilizando estes recursos, nenhum dispositivo que use WebGL pode os desfrutar.

Segundo Kuryanovich et al. (2014) um dos problemas do WebGL é sua alta curva de aprendizagem e o fato de não ter suporte para o Internet Explorer. Entretanto, o suporte foi adicionado na última versão do Internet Explorer (11). Infelizmente a dificuldade de utilização ainda persiste (LMT 21), forçando a maioria dos desenvolvedores a utilizarem abstrações criadas por bibliotecas de terceiros ⁵.

É importante também ressaltar que o WebGL não se comporta de maneira simétrica nos navegadores que implementam a especificação. Pode haver diferenças substanciais de performance em plataformas diferentes e em navegadores diferentes. Para ver um programa em WebGL é necessário um navegador recente, uma placa gráfica recente e um sistema operacional que suporte a tecnologia (Kuryanovich et al., 2014) (LMT 22). Existe também uma lista de

⁵Os apêndices contém algumas destas bibliotecas como o *tree.js*

placas gráficas com *drivers* bloqueados por não funcionarem corretamente no Firefox (Matti; Arto, 2011, p.42).

Além dos problemas citados, o suporte a especificação ainda é incompleto em vários navegadores (LMT 23). CocoonJS é uma aplicativo híbrido que preenche a fraca implementação de WebGL nos dispositivos móveis. Habilitando assim, se desenvolver em WebGL em dispositivos legados a partir do Android 2.3 e iPhone 5.

4.7 Áudio

Segundo Kuryanovich et al. (2014) a limitação do elemento de áudio no HTML5 é que seu propósito é para executar apenas um som, como o som de fundo dentro de um jogo. Não sendo adequada para efeitos sonoros ou necessidades flexíveis de áudio.

Tendo em mente esta característica limitante do elemento áudio, durante a concepção do protótipo foi utilizada a API de áudio. Infelizmente algumas vezes o som não é executado ou demora até executar de modo que é executado ao mesmo tempo que o próximo som (LMT 24). A experimentação do protótipo, relativo a API de áudio, confirmou as indicações de Kuryanovich et al. (2014) que afirma que a API de som é boa se você deseja apenas tocar alguma música, mas se você está lançando eventos em um jogo ela ainda é problemática.

Outro problema de áudio em HTML5 são os codecs. Alguns navegadores favorecem formatos OGG (vorbis) e outros favorecem o MP3 (LMT 25). No protótipo foi utilizado o MP3, que apesar de não bem visto por muitos navegadores, é suportado como uma forma de mínimo múltiplo comum. Todavia, segundo CoolUtils (2015), MP3 é um codec antigo e existem alternativas mais adequadas para a Web como o codec Opus, que oferece uma melhor relação entre taxa de compressão e qualidade.

Além das restrições de codecs de áudio nos produtos da Apple, áudio, especificamente no Safari do IOS, contém alguns problemas específicos. Por exemplo, não é possível trocar o volume através de JavaScript, também não é possível tocar mais de um som ou vídeo simultaneamente (Miller, 2011) (LMT 26).

4.8 Vídeo

O suporte a vídeo, apesar de estar melhorando, ainda é rudimentar. No Silverlight existe uma coleção de possibilidades como aplicar shaders diretamente no vídeo (Huang, 2011, p. 8) (LMT 27). Não obstante, efeito similar a shaders em vídeo é atingido através da importação de vídeo dentro de um contexto WebGL ⁶ ou canvas 2D.

Assim como áudio, o elemento *video* sofre com problemas de codecs (LMT 28), segundo Hasan et al. (2012):

⁶O endereço https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Animating_textures_in_WebGL contém exemplos de como utilizar vídeos em um contexto WebGL

O maior problema com as API's de áudio e de vídeo do HTML5 é a disputa entre os codecs dos navegadores. Por exemplo, Mozilla e Opera suportam Theora, já o Safari suporta H.264 que também é suportado pelo IE9. Ambos, Iphone e Android suportam H.264 em seus navegadores. A W3C recomenda OggVorbis e OggTheora para áudio e vídeo respectivamente.

O Safari do IOS também contém problemas exclusivos de vídeo. Segundo Miller (2011) não é possível capturar frames de vídeo usando o método *drawImage* do canvas. Também não é possível pré carregar arquivos de vídeo sem interação do usuário (LMT 29).

Para os raros casos onde o suporte a vídeo não existe, como no Opera Mini, o projeto *Vídeo for Everybody* http://camendesign.com/code/video_for_everybody é um polyfill que recorre à flash para apresentar o conteúdo.

4.9 Armazenamento

Não existe opção oficial que permita a utilização de tecnologias SQL para o lado do cliente (LMT 30). O WebSQL foi uma tentativa nesta direção mas, visto que não continha mais de uma implementação, foi descontinuado. Apesar de depreciado, o Web SQL ainda é usado por muitos desenvolvedores e mantido por alguns navegadores. Todavia, os desenvolvedores que procedem desta maneira ficam à mercê dos caprichos dos navegadores, não contando com a força de um órgão regulador para suportar a tecnologia. SQL seria interessante pois muitos desenvolvedores tem experiência com este tipo de tecnologia e por SQL permitir filtragem e correlacionamento de dados de uma forma flexível e poderosa.

Uma outra característica da Web que pode afetar negativamente aplicações que utilizam persistência local de dados é que o usuário pode configurar o navegador para não aceitar armazenamento para determinado domínio. Possivelmente comprometendo a experiência de determinada aplicação. Esta característica é muito diferente do que geralmente é oferecido em aplicações desktop, onde a aplicação determina o que precisa usar sem necessitar de consenso do usuário, que por sua vez pode escolher usar ou não a aplicação.

4.9.1 Web Storage

Muitos navegadores não permitem armazenar mais de 5MB por domínio em Web Storage, apesar da especificação permitir fazê-lo (Prall, 2012) (LMT 31). Também não é possível saber quanto espaço já foi consumido pelo Web Storage (LMT 32).

Outra limitação do Web Storage é que todas as informações são guardadas no formato texto (LMT 33). Isso força os desenvolvedores a converter os valores toda a vez que algo for armazenado ou recuperado (Prall, 2012). No protótipo isso não foi problemático pois não existe grande manipulação sobre os dados. Mas no caso de jogos mais complexos esta característica do Web Storage pode ser uma limitação substancial.

4.9.2 IndexedDB

Apesar de ser desenvolvido com objetivo de ser uma solução para todas as necessidades de armazenamento no lado do cliente, IndexedDB ainda sofre algumas limitações. O comportamento em abas anônimas não está especificado e os resultados também variam de navegador para navegador (LMT 34). Também não é possível realizar buscas em textos, algo similar ao *LIKE* do SQL (LMT 35).

Outro problema encontrado durante a pesquisa é que no Firefox existe uma pequena probabilidade de os dados se perderem. Isso se dá pois a API não espera confirmação do sistema operacional para considerar um dado válido, essa foi uma escolha dos desenvolvedores do navegador para obter um ganho em performance. Este comportamento pode ser modificado, mas é a forma padrão de funcionamento, e os desenvolvedores podem não estar considerando esta peculiaridade.

4.10 Offline

Nos arquivos de cache via manifestos, quando o download falha, o navegador emite um evento mas não há indicação de qual problema aconteceu (LMT 36). Isso pode tornar a depuração ainda mais complicada que o usual (Pilgrim, 2010). Este problema aconteceu durante o desenvolvimento do protótipo e foi trabalhoso de depurar até descobrir que um arquivo estava faltando, e qual especificamente. Uma coleção de códigos de erro e suas respectivas mensagens na especificação poderia solucionar este problema.

Os itens da palavra chave *NETWORK*: apresentam um comportamento não intuitivo. Arquivos de rede que não são declarados ali não poderão ser consumidos uma vez que exista cache na aplicação. O exemplo neste endereço <http://appcache-demo.s3-website-us-east-1.amazonaws.com/without-network/> demonstra este comportamento. Segundo Prall (2012) a especificação define que se um arquivo não for listado em alguma das seções do cache então o arquivo não estará disponível de qualquer forma para a aplicação.

4.11 Orientação

Visto que a especificação de orientação não está pronta, o suporte está longe de completo. Da mesma forma, existem diferenças substanciais nas implementações entre os navegadores (LMT 37). Por exemplo, o Firefox e Google Chrome não manipulam ângulos da mesma forma. Outrossim, alguns eixos se comportam de maneiras opostas (MDN, 2015a). O polyfill *gyronorm.js* é uma tentativa de normalizar o uso de dados de orientação nos navegadores que pode ser utilizada até a especificação e os navegadores evoluírem.

4.12 Detecção de recursos

Pilgrim (2010) cita que (LMT 38):

Grande parte da detecção de funcionalidades é feita através de JavaScript, isso força os desenvolvedores a criarem pelo menos parte da marcação em JavaScript, o que pode ser um fator limitante para o uso generalizado de HTML5.

Para resolver este problema, o ideal seria que cada tecnologia contasse com formas de detectar as funcionalidades que comporta. Não obstante, este tipo de mecanismo conflita com outros princípios da Web. Pelo fato de o HTML ser uma linguagem fundamentada exclusivamente em marcação e estrutura, adicionar condicionais não é uma alternativa viável.

4.13 Debug

Com o depurador remoto do Google Chrome foi observado uma falta de sincronia nas taxas de atualização da imagem no computador. Os efeitos do CSS não são bem apresentados sendo provavelmente difícil depurar animações. Durante o desenvolvimento do protótipo, em alguns casos quando a tela era modificada substancialmente (com nas trocas de tela), a mudança de estado no computador levou vários segundos para ser atualizada.

Outro problema do depurar do Chrome é que não existe inspetor nativo de canvas. O inspetor embutido do Google Chrome foi removido pois continha comportamentos indesejáveis para os desenvolvedores do navegador (Sjogren, 2015). Os desenvolvedores mencionaram criar uma extensão para a funcionalidade mas até o tempo de desenvolvimento deste trabalho não havia nenhuma disponível. A única alternativa viável até o momento é utilizar uma versão antiga do Chromium. O Firefox introduziu uma ferramenta de inspeção para o canvas 2D em uma Conferência de desenvolvimento de jogos em São Francisco em 2014, mas até então a tecnologia não apareceu no navegador (Porof, 2014). Atualmente a área de ferramentas para depuração do canvas 2D está parcialmente comprometida por não existir um plugin de terceiro onipresente nos navegadores como o WebGL Inspector (LMT 39).

O projeto Canvas Interceptor é uma iniciativa para preencher esta demanda permitindo capturar alguns eventos do contexto 2D e tirar *snapshots* dos passos da renderização do canvas. Entretanto, o projeto requer que o desenvolvedor inclua um arquivo JavaScript em seu projeto e inicialize o depurador dentro do código, se tornando uma saída mais custosa do que um inspetor integrado no navegador ou via extensão ⁷.

4.14 Entrada de comandos

Powell; Li (2013, p. 9) cita que: fazer scroll juntamente com gestos de ações similares é uma área onde o HTML ainda é fraco (LMT 40). O autor diz isso no sentido que os movimentos entram em conflito e os navegadores não se comportam de forma similar. Para contornar este problema pode-se utilizar bibliotecas como o iScroll, TouchScroll, GloveBox, Sencha, jQuery Mobile, entre outros.

⁷Mais informações sobre o projeto Canvas Interceptor podem ser encontradas no seguinte endereço <https://github.com/Rob--W/canvas-interceptor>

4.14.1 Gamepad

A forma de gerenciar a conexão de Gamepads é diferente entre o Firefox e Google Chrome. No Firefox é lançado um evento toda a vez que o Gamepad é desconectado, já no Google Chrome é necessário verificar um vetor de gamepads pela existência do objeto de tempos em tempos (W3C, 2015a) (LMT 41). Este problema é reflexo da especificação ainda incompleta. Forçando os desenvolvedores a duplicarem a mesma atividade de formas diferentes.

4.15 Disponibilização

Durante o desenvolvimento notou-se que o serviço PhoneGap Build não contém a última versão do PhoneGap e não é possível utilizar plugins através dele. Embora no protótipo isso não tenha sido um problema, em projetos maiores estes tipos de requerimentos podem ser essenciais.

Outra limitação de disponibilização, neste caso para aplicativos Web puros, é que a possibilidade de adicionar aplicativos a área de trabalho está disponível apenas para o Safari e Google Chrome (LMT 42). Este tipo de possibilidade é importante por gerar uma experiência similar a nativa em dispositivos móveis.

A disponibilização via mercado do IOS também é problemática. Toda a aplicação precisa ter seu conteúdo revisado a cada nova versão sendo que a revisão pode ser demorada e geralmente existe uma fila de espera substancial. Se for comparado a muitos processos de disponibilização automatizados na Web, disponibilizar aplicações para o IOS é rudimentar.

4.15.1 Monetização

Outro aspecto que pode ser limitante para aplicações Web é sua forma de monetização. É mais complexo monetizar jogos para navegadores, visto que não se pode vender pacotes para o usuário baixar (Vanhatupa, 2010, p. 44).

Dado este problema, desenvolvedores são muitas vezes forçados a criar formas alternativas de lucro. A monetização de aplicativos Web é geralmente feita através de propagandas (Vanhatupa, 2010, p. 44). Outra saída é a venda de recursos adicionais, Vanhatupa (2010, p. 44) afirma que a venda de moeda virtual é uma forma comum de monetização em jogos. Jogos Web também podem ser integrados a sistemas de pagamento, mas possivelmente aumentado a complexidade da aplicação. Ou, se optarem pela arquitetura híbrida, podem ser integrados a algum mercado como o do Android ou da Apple. Alternativamente, se estiverem dispostos a relegar parcela do lucro ao Facebook, podem ser integrados a plataforma e usufruir de seus usuários.

4.16 Outras Limitações

Hasan et al. (2012) menciona algumas limitações em uma âmbito HTML em geral:

Não podemos mudar a imagem de fundo do dispositivo, ou adicionar toques etc. Similarmente, existem muitas API's de nuvem como os serviços de impressão do iCloud ou Google Cloud que estão disponíveis para aplicações nativas mas não para HTML5. Outros serviços utilitários como o C2DM do Google que está disponível para desenvolvedores Android para utilizar serviços de *push* também não estão disponíveis para o HTML5.

4.17 Revisão das Limitações

Nesta seção será feita uma análise sobre as limitações encontradas. Nem todas as limitações registradas previamente aparecem nos quadros adiante pois algumas são específicas demais para se encaixarem como limitações da Web. Por exemplo, os problemas no depurador do Chrome. Apesar de serem limitações concretas para usuários do Chrome, e terem afetado o desenvolvimento do protótipo, não se tratam de problemas relativos a Web, outrossim ao navegador em específico.

O quadro 4.1 é um comparativo das tecnologias pesquisadas e seu suporte. As informações de suporte foram extraídas, em grande parte, do site *Can I Use* e os navegadores populares em questão são: Internet Explorer (11), Edge (13), Firefox (43), Google Chrome (47), Safari (9) e Opera (34).

Tecnologia	Suporte nas últimas versões estáveis dos navegadores populares	Polyfills disponíveis para versões antigas
Canvas	Sim	Sim
SVG	Sim	Sim
WebGL	Parcial	Sim
Gamepad	Não	Sim
WebSockets	Sim	Sim
IndexedDB	Parcial	Sim
Web Storage	Sim	Sim
Offline	Sim	Não
WebCL	Não	Não
WebVR	Não	Não
WebAssembly	Não	Sim
Tag áudio	Sim	Sim
Áudio API	Não	Sim
Vídeo	Não	Sim
Prefetch	Não	Sim
Web Animations	Não	Sim
Web Workers	Sim	Sim

Quadro 4.1: Tecnologias dos jogos e seu suporte

Das tecnologias citadas acima 82% tem suporte nos navegadores populares, sendo através de implementação nativa ou através de polyfills. Com este comparativo fica visível que, mesmo em alguns casos não existindo suporte nativo ou ele sendo incompleto e com limitações, na vasta maioria dos casos, é possível utilizar tecnologias da Web direcionadas para o desenvolvimento de jogos.

O quadro 4.2 é um resumo das limitações mais relevantes encontradas e se as mesmas podem ser contornadas ou não. Cada limitação apresentada a seguir é acompanhada de seu código que serve de ligação ao trecho do texto onde a limitação foi detalhada.

Código	Descrição	Existe solução ou contorno?
LMT 1	Para garantir que algo de fato funcione é necessário testar em múltiplos dispositivos	Parcial
LMT 2	Dificuldade em construir interfaces nativas para os diversos dispositivos	Sim
LMT 3	Desvantagem para alguns jogadores devido ao tamanho da tela	Parcial
LMT 4	Tecnologias equivalentes em fase experimental levam o prefixo do motor de renderização	Sim
LMT 5	Toma-se um tempo substancial para obter um consenso na criação de especificações	Não
LMT 6	O suporte as novas especificações muitas vezes demora para ser implementado	Sim
LMT 7	Por ser interpretado, erros no código JavaScript só podem ser descobertos em tempo de execução	Parcial
LMT 8	Impossibilidade de passar objetos completos em WebSockets	Não
LMT 9	Scripts só podem ser rodados com segurança no final da renderização	Parcial
LMT 10	A manutenibilidade de JavaScript tende a ser pior do que a de outras linguagens comuns como Java ou C#	Sim
LMT 11	Propagação de valores inválidos NAN	Parcial
LMT 12	Checagem de alguns tipos retorna resultados inesperados	Parcial

LMT 13	É pouco eficiente produzir animações em JavaScript	Parcial
LMT 14	Impossibilidade de detectar e manipular <i>fullscreen</i> de forma padronizada	Parcial
LMT 15	Problemas de performance em arquivos SVG muito grandes	Não
LMT 16	É praticamente impossível obter posicionamento preciso com SVG	Não
LMT 17	A performance do canvas varia de plataforma para plataforma	Parcialmente
LMT 18	Não existe suporte nativo a animações em Canvas	Não
LMT 19	A integração do Canvas com as demais tecnologias da Web é pobre	Não
LMT 20	Falta de alguns recursos disponíveis para OpenGL em ambientes que os permitiriam	Não
LMT 21	Para construir jogos é muito difícil utilizar WebGL diretamente	Sim
LMT 22	WebGl não funciona em placas gráficas antigas	Não
LMT 23	Em muitas plataformas o suporte a WebGL é incompleto	Parcial
LMT 24	A API de Audio é problemática com sons frequentes	Não
LMT 25	Não existe um único codec de áudio moderno suportado em todos os navegadores	Parcial
LMT 26	O Safari conta com problemas de controle de execução de multimídia	Não
LMT 27	Não é possível aplicar shaders diretamente em vídeo	Sim

LMT 28	Falta de consenso sobre um codec de vídeo padrão	Não
LMT 29	A integração de vídeo e canvas do Safari é problemática e vídeo só pode ser carregado pelo usuário	Não
LMT 30	Não existência de tecnologia oficial de SQL para Web	Parcial
LMT 31	Em muitos navegadores não é possível armazenar mais de 5MB em Web Storage	Não
LMT 32	Não existe forma padronizada de descobrir quanto espaço ainda resta em Web Storage	Parcial
LMT 33	Web Storage só permite armazenamento de dados como texto	Não
LMT 34	O comportamento do IndexedDB em abas anônimas varia	Não
LMT 35	Não é possível realizar buscas em texto com o IndexedDB	Não
LMT 36	Erros nas configurações do manifesto offline não contém mensagens descritivas	Não
LMT 37	A implementação da API de orientação é divergente entre os navegadores	Sim
LMT 38	Checagem de recursos exclusivamente por JavaScript força a implementação de markup em scripts	Não
LMT 39	Não existe inspetor Canvas 2D multiplataforma para navegadores	Parcial
LMT 40	Scroll juntamente com outros gestos não funciona corretamente na maioria dos navegadores	Sim
LMT 41	Não existe forma padronizada de detectar a entrada e saída de <i>gamepads</i> nos navegadores	Não

LMT 42	Não existe forma padronizada de adicionar ícones de aplicativos Web a área de trabalho	Não
--------	--	-----

Quadro 4.2: Lista de problemas e limitações

Das limitações dispostas no quadro 4.2, aquelas resolvidas parcialmente identificam que o problema pode ser contornado em partes, ou então que o problema é contornado de forma inesperada e não intuitiva. Como a utilização de um depurador internamente no código, no caso da limitação LMT 39.

A distribuição do quadro é a seguinte: 47% atualmente não são resolvidas de forma alguma, 33% são resolvidas parcialmente e 19% são totalmente contornáveis.

Apesar da parcela de limitações não contornáveis ser substancial, muitas delas são problemas relativos a fase prematura de suas especificações e, ao que tudo indica, serão resolvidos com o passar do tempo. Como no caso da limitação LMT 41, que as diferenças se dão em grande parte por a especificação não estar pronta.

Talvez as limitações mais difíceis de contornar sejam aquelas que entram em conflito com as filosofias da Web. Algumas possibilidades que são triviais em ambientes desktop podem tornar-se fatores limitantes devido a estas filosofias. Possivelmente a forma mais expressiva deste problema se dá com tecnologias que de alguma forma podem ser consideradas inseguras. Como, até recentemente, era o caso do controle de tela cheia através de JavaScript (LMT 14). Em plataformas nativas esse controle é geralmente garantido. Já na Web, um dos argumentos contra a sua especificação era que poderia ser um problema de segurança permitir o uso de Fullscreen via JavaScript. Felizmente com o avanço de especificações como a *Permissions API*, as tecnologias, ao invés de serem privadas de existir, tendem a cada vez mais requisitar o consenso do usuário para sua utilização. Isso pode ser abstraído como uma tendência geral, de os problemas serem convertidos em novas especificações e, aos poucos, sendo eliminados da Web.

5 CONCLUSÕES

Neste trabalho revisamos tecnologias relevantes no desenvolvimento de jogos multiplataforma em HTML5 e algumas das limitações a elas relacionadas. Para tanto, elaborou-se uma pesquisa bibliográfica dos assuntos relevantes. Em seguida, criou-se um protótipo de jogo para avaliar experimentalmente algumas das tecnologias e detectar limitações. Apesar de simples, o protótipo desenvolvido ajudou substancialmente na validação de algumas limitações previamente encontradas e na detecção de alguns problemas adicionais.

As limitações encontradas através do processo mencionado foram registradas e avaliadas. Os resultados obtidos nos levam a crer que, apesar de o número de limitações ser substancial, grande parte destas limitações podem ser contornadas pelo programador ou já estão no processo de serem solucionadas pelas especificações e navegadores. As tecnologias da Web vem evoluindo a grande passo e o número de pontos positivos da abordagem HTML para desenvolvimento de jogos é muito grande em comparação às limitações.

É seguro dizer que o HTML ainda não está pronto para substituir todas as estratégias de desenvolvimento de jogos, nem pode atender todas as categorias de jogos. Não obstante, as tecnologias da Web podem ser aplicadas ao desenvolvimento de jogos em um leque crescente de casos.

Existem várias funcionalidades interessantes para jogos que ainda não estão prontas ou, por hora, não funcionam em todos os navegadores como: WebVR, ES6, HTTP/2, Gamepad, entre outras. Entretanto a maioria destas tecnologias já podem ser utilizadas pelo programador, requerendo apenas um pouco de diligência.

No contexto multiplataforma, provavelmente o maior desafio é prover interatividade em meio a uma coleção de dispositivos diferentes. Todavia, através da criação de aplicações incrementais, adicionando recursos conforme a capacidade dos dispositivos, e utilizando os recursos de retrocompatibilidade que fazem parte dos alicerces do HTML. É possível fornecer a melhor experiência que cada plataforma comporta.

Das limitações existentes, provavelmente as mais difíceis de resolver são aquelas relacionadas as próprias filosofias da Web, como a impossibilidade de controlar recursos via script por levantar problemas em potencial sobre a segurança do usuário. Contudo, mesmo estes problemas podem ser solucionados dado o interesse da comunidade.

De uma visão macroscópica, se considerarmos o histórico do HTML, principalmente após o lançamento do HTML5, as limitações do HTML são passageiras. A figura 9 corrobora com esta hipótese, o crescimento do suporte das tecnologias nesta imagem apresenta uma

curva, em traços gerais, exponencial. Ao que tudo indica o futuro dos jogos em HTML5 parece brilhante.

5.1 Trabalhos Futuros

Visto que a especificação do HTML é viva, checar suas limitações é uma tarefa que poderia ser feita de tempos em tempos. EMAScript 6 e Web Assembly podem mudar completamente o cenário de desenvolvimento de jogos Web assim que se tornarem opções viáveis. Uma nova revisão das limitações dos jogos seria bem-vinda quando isso acontecer.

Seria interessante que assuntos como o WebGL e WebVR, levemente abordados neste trabalho, fossem estudados com profundidade, visto que são deveras importantes para a criação de jogos cada vez mais interativos. O suporte a mais versões de navegadores e plataformas, bem como outras categorias de jogos, também poderiam ser estudadas.

A utilização de plugins foi intencionalmente ignorada neste trabalho. Conquanto, em jogos comerciais, estas ferramentas são imprescindíveis. Trabalhos que analisem a construção de um jogo sob a perspectiva comercial, utilizando plugins, da forma mais aproximada o possível da realidade do mercado, seriam igualmente interessantes.

REFERÊNCIAS BIBLIOGRÁFICAS

- ALEXANDER, M. *Debugging HTML5 Games In The Browser*. 2013. Disponível em: <<http://help.yoyogames.com/entries/25057993-Debugging-HTML5-Games-In-The-Browser>>. Acesso em: 25 dez. 2015.
- ARORA, S. *10 Best JavaScript Style Guides Including Airbnb and Idiomatic*. 2015. Disponível em: <<http://noeticforce.com/best-javascript-style-guide-for-maintainable-code>>. Acesso em: 19 jan. 2016.
- BARNETT, J. Building a Cross-Platform Mobile Game with HTML5. *University of East Anglia*, 2014.
- BELCHIN, M. *ECMAScript 6 Today*. 2015. Disponível em: <<https://moisesbm.wordpress.com/2015/02/22/ecmascript-6-today/>>. Acesso em: 7 dez. 2015.
- BRISTOWE, J. *jQuery UI vs Kendo UI*. 2014. Disponível em: <<http://jqueryuivskendoui.com/>>. Acesso em: 1 jan. 2016.
- BROOKS, F. P. J. What's real about virtual reality. *University of North Carolina*, 1999.
- BRUINS, S. The current state of cross-platform game development for different device types, 2014.
- CEVELLEJA, C. *A Simple Guide to Getting Started With Grunt*. 2014. Disponível em: <<https://scotch.io/tutorials/a-simple-guide-to-getting-started-with-grunt>>. Acesso em: 24 dez. 2015.
- CHROME. *Chrome DevTools Overview*. 2013. Disponível em: <<https://developer.chrome.com/devtools>>. Acesso em: 19 dez. 2015.
- CIMAN, M.; GAGGI, O.; GONZO, N. Cross-Platform Mobile Development: A Study on Apps with Animations. *University of Padua*, 2012.
- COOLUTILS. *What is Opus?* 2015. Disponível em: <<https://www.coolutils.com/Formats/Opus>>. Acesso em: 20 jan. 2016.
- DAVIDSSON, O.; PEITZ, J.; BJORK, S. Game Design Patterns for Mobile Games. *Saimaa University of Applied Sciences*, 2014.
- DEVDOCS. *DOM Events*. 2015. Disponível em: <http://devdocs.io/dom_events/>. Acesso em: 24 dez. 2015.

- DOROKHOVA, R.; AMELICHEV, N. Comparison of Modern Mobile Platforms from the Developer Standpoint. *8TH CONFERENCE OF FINNISH-RUSSIAN UNIVERSITY COOPERATION IN TELECOMMUNICATIONS*, 2010.
- DUCKETT, J. *HTML and CSS - Design and Build Websites*. 2011.
- ECMA. *ECMAScript 2015 Language Specification*. 2015. Disponível em: <<http://www.ecma-international.org/ecma-262/6.0/>>. Acesso em: 1 dez. 2015.
- GARISEL, T.; IRISH, P. How Browsers Work: Behind the scenes of modern web browsers, 2011.
- GRANIC, I.; LOBEL, A.; ENGELS, R. C. M. E. The Benefits of Playing Video Games. *Radboud University Nijmegen*, 2014.
- HASAN, Y. et al. Smart Phones Application development using HTML5 and related technologies: A tradeoff between cost and quality. *International Journal of Computer Science Issues*, 2012.
- HICKSON, I. *HTML is the new HTML5*. 2011. Disponível em: <<https://blog.whatwg.org/html-is-the-new-HTML5>>. Acesso em: 11. nov. 2015.
- HTTP/2. *HTTP/2*. 2015. Disponível em: <<https://http2.github.io/>>. Acesso em: 25 dez. 2015.
- HUANG, J. S. Research on Html5 Development. *Jiangsu University*, 2011.
- JANISZEWSKI, M. Tese de doutoramento. Vrije Universiteit Amsterdam, 2014.
- JR, R. *Chrome OS vs. Android: What's the difference?* 2010. Disponível em: <<http://www.computerworld.com/article/2469738/mobile-apps/chrome-os-vs--android--what-s-the-difference-.html>>. Acesso em: 26 dez. 2015.
- KAMACI. *OpenGL ES versus OpenGL*. 2010. Disponível em: <<http://stackoverflow.com/questions/4519264/opengl-es-versus-opengl>>. Acesso em: 2 jan. 2015.
- KRILL, P. *Mobile app developer's interest in HTML5 is slipping*. 2013. Disponível em: <<http://www.infoworld.com/article/2609608/javascript/mobile-app-developers--interest-in-html5-is-slipping.html>>. Acesso em: 2 jan. 2015.
- KURYANOVICH, E. et al. *HTML5 Games Most Wanted: Build the Best HTML5 Games*. 2014.
- LEHR, M. *Creating a WebGL Game with Unity 5 and JavaScript*. 2016. Disponível em: <<http://www.sitepoint.com/creating-webgl-game-unity-5-javascript/>>. Acesso em: 21 jan. 2015.
- LEMES, D. d. O. Tese de doutoramento. Pontifícia Universidade Católica de São Paulo, 2009.
- LIE, H. W. Cascading Style Sheets. Tese de doutoramento. 2005.
- LUBBERS, P.; BRIAN, A.; FRANK, S. *Pro HTML5 Programming - Powerful APIs for Rich Internet Application Development*. 2010.

- MATTI, A.; ARTO, S. Building 3D WebGL Applications. *Tampereen University of Technology*, 2011.
- MCCANN, C. HTML5 Research Journal. *IT Carlow Games*, 2012.
- MDN. *Detecting device orientation*. 2015. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/Detecting_device_orientation>. Acesso em: 25 dez. 2015.
- MDN. *Drawing shapes with canvas*. 2015. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Drawing_shapes>. Acesso em: 23 dez. 2015.
- MDN. *Using CSS transitions*. 2015. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions>. Acesso em: 1 nov. 2015.
- MDN. *HTML HyperText Markup Language*. 2016. Disponível em: <<https://developer.mozilla.org/en-US/docs/Web/HTML>>. Acesso em: 7 jan. 2016.
- MILLER, H. B. M. *Unsolved HTML5 video issues on iOS*. 2011. Disponível em: <<http://blog.millermedeiros.com/unsolved-html5-video-issues-on-ios/>>. Acesso em: 2 jan. 2015.
- MOBILEXWEB. *Android 4.4 KitKat, the browser and the Chrome WebView*. 2013. Disponível em: <<http://www.mobilexweb.com/blog/android-4-4-kitkat-browser-chrome-webview>>. Acesso em: 26 dez. 2015.
- MOBILEXWEB. *Home screen web apps for Android thanks to Chrome 31+*. 2013. Disponível em: <<http://www.mobilexweb.com/blog/home-screen-web-apps-android-chrome-31>>. Acesso em: 1 jan. 2015.
- MONONY, J. *The Step-by-Step Guide to Publishing a HTML5 Mobile Application on App Stores*. 2015. Disponível em: <<http://www.joshmorony.com/the-step-by-step-guide-to-publishing-a-html5-mobile-application-on-app-stores/>>. Acesso em: 24 dez. 2015.
- MORONY, J. Viability of developing cross-platform mobile business applications using HTML5 Mobile Framework. 2013.
- NEUBERG, B. *What Is the Open Web and Why Is It Important?* 2008. Disponível em: <<http://codinginparadise.org/weblog/2008/04/whats-open-web-and-why-is-it-important.html>>. Acesso em: 25 nov. 2015.
- NEUMANN, A.; WINTER, M. A. Time for SVG - Towards High Quality Interactive Web - Maps. *Institute of Cartography, ETH Zurich*, 2001.
- PILGRIM, M. *Dive into HTML5*. 2010.
- PIRTTIAHO, J. Cross-platform mobile game development. *Saimaa University of Applied Sciences*, 2014.

- POROF, V. *Introducing the Canvas Debugger in Firefox Developer Tools*. 2014. Disponível em: <<https://hacks.mozilla.org/2014/03/introducing-the-canvas-debugger-in-firefox-developer-tools/>>. Acesso em: 25 dez. 2015.
- POWELL, M.; LI, Y. HTML5 - A Serious Contender to Native App Development or Not?, 2013.
- PRADO, E. F. Introdução ao desenvolvimento de Games com GWT e HTML5. *Departamento de Computação, Univerisidade de São Carlos*, 2012.
- PRALL, C. *Asset Management for Web Games*. 2012. Disponível em: <<http://buildnewgames.com/game-asset-management/>>. Acesso em: 2 jan. 2015.
- RAUSCHMAYER, A. *A first look at what might be in ECMAScript 7 and 8*. 2011. Disponível em: <<http://www.2ality.com/2011/09/es6-8.html>>. Acesso em: 7 dez. 2015.
- ROCHELEAU, J. *Introductory Guide to Building Your First HTML5 Game*. 2015. Disponível em: <<http://marketblog.envato.com/tips/building-your-first-html5-game/>>. Acesso em: 19 dez. 2015.
- ROGERS, C. *Web Audio API*. 2012. Disponível em: <<https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html>>. Acesso em: 18 dez. 2015.
- SEIDELIN, J. *HTML5 Games - Creating fun with HTML5, CSS3, and WebGL*. 2010.
- SJOGREN, K.-J. *HTML5 canvas inspector?* 2015. Disponível em: <<http://stackoverflow.com/questions/31439201/chrome-canvas-inspector-2015>>. Acesso em: 25 dez. 2015.
- VAHATUPA, J.-M. On the development of Browser Games - Current Technologies and the Future, 2014.
- VANHATUPA, J. M. Browser Games for Online Communities. *International Journal of Wireless and Mobile Networks*, 2010.
- W3C. *Device Orientation Event Specification*. 2011. Disponível em: <<http://www.w3.org/TR/orientation-event/>>. Acesso em: 25 dez. 2015.
- W3C. *Gamepad*. 2015. Disponível em: <<https://w3c.github.io/gamepad>>. Acesso em: 25 dez. 2015.
- W3C. *Selectors*. 2015. Disponível em: <<http://www.w3.org/TR/CSS21/selector.html>>. Acesso em: 17 nov. 2015.
- WIKIPEDIA. *Crosswalk Project*. 2015. Disponível em: <<https://en.wikipedia.org/wiki/Crosswalk\Project>>. Acesso em: 24 jan. 2016.
- WIKIPEDIA. *Depuração*. 2015. Disponível em: <<https://pt.wikipedia.org/wiki/Depuracao>>. Acesso em: 25 dez. 2015.
- WIKIPEDIA. *World Wide Web*. 2015. Disponível em: <https://en.wikipedia.org/wiki/World_Wide_Web>. Acesso em: 24 jan. 2016.

A Diagrama de classes

A figura 32 é o diagrama de classes detalhado do protótipo.



Figura 32: Diagrama de classes completo

B Bibliotecas relevantes no desenvolvimento de jogos Web

A quantidade de ferramentas Web a disposição dos usuários é enorme. Segundo Kuryanovich et al. (2014) desenvolvedores da Web são geralmente de mente aberta e criaram uma quase infinita variedade de bibliotecas e frameworks pela internet. Sabendo disso, é uma tarefa praticamente impossível ter habilidade em todas as ferramentas existentes. Entretanto, é importante que os desenvolvedores ao menos conheçam as ferramentas mais importantes disponíveis para que, no momento necessário, saibam qual aprender. Todavia, Seidelin (2010) ressalta a importância de sermos moderados quanto a escolha de bibliotecas no contexto Web multiplataforma.

Muitos desenvolvedores da Web utilizam bibliotecas como jQuery e o Prototype de modo a se verem livres de ter que lidar com partes triviais do desenvolvimento Web, como selecionar e manipular elementos do DOM. Muitas vezes essas bibliotecas incluem várias funcionalidades que não são utilizadas. É recomendável cautela para verificar se realmente é necessário adicionar 50-100k de bibliotecas, ou se alguma coisa mais simples e menor não trará os mesmo benefícios, especialmente quando desenvolvendo multiplataforma onde uma rápida conexão a internet nem sempre é garantida.

Esta preocupação aumenta no contexto de desenvolvimento de jogos. Ainda segundo Seidelin (2010) o site MicroJS <https://microjs.com> oferece uma coleção de micro bibliotecas focadas em áreas particulares em detrimento de grandes bibliotecas cheias de funcionalidades.

B.1 -prefix-free

A biblioteca *-prefix-free* <http://leaverou.github.io/prefixfree/> é um polyfill que possibilita os desenvolvedores utilizarem CSS sem a adição de prefixos, o que torna o trabalho de desenvolvedores bem menos redundante. A biblioteca é razoavelmente leve (2KB), e quando o sistema desenvolvido com ela estiver pronto, e o CSS evoluir removendo os prefixos utilizados, ela pode ser removida sem ter que mudar o resto do código.

B.2 Crosswalk

Crosswalk é uma solução para eliminar a diferenças de versões nos aplicativos Android híbridos e prover sempre as últimas atualização do HTML5. Para tanto, o projeto empacota os

fontes de uma aplicação Web juntamente com uma versão do Chromium e do motor de renderização Blink. Isso faz com que o software se comporte da mesma forma para todos os dispositivos Android a partir da versão 4. Crosswalk foi criado pela Intel em 2013 desde então novas versões são disponibilizadas de seis em seis semanas incorporando as últimas modificações do Chromium (Wikipedia, 2015a).

B.3 PhoneGap

PhoneGap é um framework para desenvolvimento de sistemas híbridos que empacota uma aplicação Web dentro de um contêiner nativo e provê um conjunto de funcionalidades nativas via JavaScript. PhoneGap permite acesso as APIs de câmera, geolocalização, contatos, calendário, etc (Ciman; Gaggi; Gonzo, 2012, p. 3). Todos os sistemas populares são suportados pelo PhoneGap: Android, IOS, Windows Phone, etc. Diferentemente de outros frameworks, PhoneGap não provê uma forma para criar funcionalidades de interface como animações e listas (Morony, 2013, p. 15). O framework geralmente é combinado com outros para obter este tipo de funcionalidade.

PhoneGap também conta com um serviço para empacotamento online o PhoneGap Build. Este serviço possibilita que se carregue um arquivo compactado com os fontes que o PhoneGap Build se encarrega de gerar os binários para as plataformas requeridas. Este recurso é valioso pois a alternativa é configurar o computador de desenvolvimento para compilar para as plataformas alvo, um processo entediante e complexo.

B.4 Unity

Unity é um motor de jogos multiplataforma utilizado para criar jogos para PC's consoles, dispositivos móveis e Websites (Lehr, 2016). Mais do que um conjunto de bibliotecas, o Unity disponibiliza também um ambiente integrado de desenvolvimento onde é possível alterar aspectos de jogos através do uso de interfaces gráficas. Com Unity é possível desenvolver em um dialeto JavaScript, entre outras linguagens, e exportar para diversas plataformas. Para a Web, o Unity possibilita exportar em WebGL e Web Assembly, sendo uma combinação poderosa e que pode habilitar grande performance.

B.5 Tree.js

Treejs é um framework popular para o desenvolvimento em WebGL. Consistem em uma abstração sobre WebGL que permite os autores se focarem na criação de conteúdo para Web, ao invés de dispenderem tempo manipulando os detalhes da WebGL. Possibilita trabalhar com efeitos, luzes, cenas e outras abstrações em detrimento de shaders, vértices, e outros conceitos primitivos.

B.6 Appcelerator Titanium

Appcelerator Titanium é um framework JavaScript que possibilita a construção de aplicativos mobile nativos para Android, IOS e outras plataformas. Titanium oferece uma vasta quantidade de APIs; sendo bem documentadas, contendo descrições de métodos, parâmetros de entrada e saída e algumas vezes exemplos de utilização (Ciman; Gaggi; Gonzo, 2012, p. 2). A tecnologia também conta com uma IDE especializada para o desenvolvimento em Titanium, enquadrando-se também em um ambiente desenvolvimento de jogos.

B.7 jQuery Mobile

jQuery Mobile é um dos mais famosos frameworks mobile da Web, isso se dá, parcialmente, pela popularidade do jQuery em si (Morony, 2013, p. 14). Ciman; Gaggi; Gonzo (2012, p. 2) cita que:

jQuery provê uma grande gama de APIs para muitos propósitos, por exemplo adicionar ou remover elementos, gestão de eventos de clique, manipulação de estilo, etc. Também provê APIs para animações, por exemplo aparecer/desaparecer, etc.

jQuery Mobile não é um framework para todas as necessidades mobile. Focando-se principalmente na interface; acesso ao hardware, instalação nativa e outros aspectos do desenvolvimento multiplataforma são responsabilidades do programador. jQuery mobile sofre com problemas de performance em ambientes móveis, em partes por não utilizar aceleração de hardware para criar suas interfaces, como fazem alguns concorrentes como o Sencha Touch (Morony, 2013, p. 14).

B.8 Kendo UI Mobile

É um framework baseado em jQuery que provê componentes para a construção de interfaces semelhantes as nativas através da utilização de ferramentas da Web. Existem interfaces especializadas para IOS, Android, Windows Phone e Blackberry (Bristowe, 2014).

B.9 Sencha Touch

Sencha Touch é um framework para desenvolvimento multiplataforma que fornece um conjunto de componentes para criação de interfaces gráficas, estruturas MVC e empacotamento. Morony (2013, p. 14) cita que o Sencha Touch é um dos mais rápidos frameworks disponíveis. Com o Sencha Touch desenvolve-se em JavaScript e nas demais tecnologias da Web e cria-se binários para Android, IOS, Windows Phone, Tizen, etc.

C Ambientes de desenvolvimento de jogos em HTML

C.1 PlayCanvas

PlayCanvas é uma plataforma para a construção de jogos 3D na nuvem desenvolvida com foco em performance. Permite a hospedagem, controle de versão e publicação dos aplicativos nela criados, possibilita também a importação de modelos 3D de softwares populares como: Maya, 3ds Max e Blender;

C.2 Intel HTML5 Development Environment

Fornece uma solução na nuvem, completa para o desenvolvimento em múltiplas plataformas. Com serviços de empacotamento, serviços para a criação e testes de aplicativos, com montagem de interfaces *drag-and-drop* e bibliotecas para a construção de jogos utilizando aceleração de hardware, o que garante até duas vezes mais performance que aplicativos mobile baseados em Web tradicionais. Esta solução é gratuita, open-source e funciona através de um plugin para o Google Chrome.

D Tecnologias de Compilação multiplataforma

D.1 GWT

Segundo Prado (2012, p. 29)

O GWT é um framework essencialmente para o lado do cliente que dá suporte à comunicação com o servidor através de RPCs (*Remote Procedure Calls*). Ele não é um framework para aplicações clássicas da Web, pois deixa a implementação da aplicação Web parecida com implementações em desktop.

Com o GWT se programa em Java e exporta-se com código otimizado para as plataformas alvo. Sua licença é Apache 2 e, como o framework é em Java, pode ser rodado em todos os sistemas operacionais.

D.2 Libgdx

Libgdx <https://libgdx.badlogicgames.com/> é um framework focado no desenvolvimento de jogos nativos multiplataforma. O código é escrito uma vez e a aplicação pode ser portada para todas as plataformas sem nenhuma modificação (Pirttiaho, 2014, p. 8). A linguagem do Libgdx é Java, e é possível compilar para Android, IOS, HTML5, entre outros. Também é possível desenvolver em todos os desktops comuns.

E Sistemas de Building

Segundo Cevelleja (2014)

Durante o desenvolvimento de aplicações Web existem muitas tarefas que tem que ser feitas repetidamente. Estas tarefas incluem minificar o JavaScript e CSS, rodar testes unitários, aplicar *linters* nos arquivos para checar erros, compilar os pré processadores de CSS (LESS, SASS), e muito mais.

E.1 Grunt

O Grunt é um automatizador destas tarefas recorrentes. Para tanto, o Grunt conta com um grande ecossistema de plugins que podem ser compostos para realizar as mais diversas tarefas desejadas para determinada aplicação. No protótipo o Grunt foi utilizado com o intuito primário de realizar a minificação e disponibilização da aplicação como um conjunto de arquivos prontos para a produção.

Sendo assim, os seguintes plugins foram utilizados:

- grunt-contrib-uglify: responsável por minificar os arquivos JavaScript;
- grunt-contrib-cssmin: responsável por minificar arquivos CSS;
- grunt-contrib-copy: responsável por copiar os arquivos de desenvolvimento para a pasta de distribuição;
- grunt-contrib-watch: observar modificações nos arquivos de desenvolvimento e iniciar o processamento dos plugins acima;

E.2 Gulp

O Gulp utiliza o conceito de *streams* para aplicar todas as modificações sobre um arquivo de uma vez só.

Barnett (2014) afirma que:

Essencialmente, Gulp trabalha com streams de arquivos para dentro e fora dos módulos do Node.js possibilitando que a aplicação se transforme em uma variedade de combinações. Uma vez que a transformação é completa, a saída pode ser escrita em um arquivo no sistema. Essa forma de trabalhar com streams é possibilitada pelo Node.js, sendo altamente eficiente comparado com a maioria dos sistemas de build, visto que não existem arquivos temporários para serem escritos em disco (tudo é feito em tempo de execução utilizando objetos de virtuais).

F Fontes do protótipo