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

LIMITAÇÕES DO HTML5 NO DESENVOLVIMENTO DE JOGOS MULTIPLATAFORMA

JEAN CARLO MACHADO

Bento Gonçalves, dezembro de 2015.

JEAN CARLO MACHADO

LIMITAÇÕES DO HTML5 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: Mr. Rafael Jaques

Bento Gonçalves, dezembro de 2015.

RESUMO

LLorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

orem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Palavras-chave: HTML5, Limitações, Jogos, Multiplataforma

ABSTRACT

LLorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

orem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Palavras-chave: HTML5, Limitações, Jogos, Multiplataforma

LISTA DE FIGURAS

SUMÁRIO

1 CONTEXTUALIZAÇÃO

1.1 JOGOS

Desenvolvedores de jogos web podem rapidamente satisfazer as necessidades de seus jogadores, mantendo-os leais a tecnologia HTML5 (Zhang 2012).

Mais de 80% do tempo total gasto utilizando dispositivos móveis é na utilização de aplicativos, e 32% deste tempo é para jogar vídeo games (Janiszewswi 2014).

Mais de 70% das pessoas que jogam vídeo games são adultos (Granic, Lobel e Engels 2014).

A maioria dos desenvolvedores demonstra interesse para o HTML5. (referenciar) E muito pouco investimento é necessário para começar a desenvolver jogos utilizando as tecnologias da WEB (Kuryanovich et al. 2014).

O tempo de desenvolvimento de uma aplicação em HTML5 é 67% menor (referenciar) que aplicações nativas. Isso mostra o custo efetivo de aplicações baseadas em HTML5.

A real vantagem de aplicações em HTML5 é o suporte horizontal entre as plataformas - que é a maior razão por trás do custo efetivo («Smart Phones Application development using HTML5 and related technologies: A tradeoff between cost and quality»).

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.(«Smart Phones Application development using HTML5 and related technologies: A tradeoff between cost and quality»).

Alguns jogadores de Hattrick tem participado do jogo por mais de 10 anos (Vanhatupa 2010)

Funcionalidades foram disponibilizadas de diversas fontes e não foram construídas de forma consistente com as demais. Além disso, devida a única característica da Web, erros de implementação se tornam frequentes, e muitas vezes se tornam o padrão, pois outras funcionalidades dependem destas primeiras antes que elas estejam estáveis. (W3C manual)

Os desenvolvedores de navegadores podem interpretar/implementar as especificações erroneamente aumentando os problemas de compatibilidade.

1.2 Limitações

A utilização da WEB é gigantesca, mas criar jogos dinâmicos e de tempo real não é o primeiro objetivo da WEB (Kuryanovich et al. 2014).

1.3 ESTE TRABALHO

Este projeto propõe analisar as limitações do HTML5 quanto relativo a construção de jogos multiplataforma. Através de revisão bibliográfica e da criação de um protótipo de jogo multiplataforma.

2 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 à construção de jogos nativos.

2.1 OBJETIVOS

2.1.1 OBJETIVO GERAL

Identificar possíveis limitações no processo de desenvolvimento 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.

2.1.2 OBJETIVOS ESPECÍFICOS

O primeiro objetivo é estudar e resumir as seguintes tecnologias:

- Tecnologias de renderização
- Formas de disponibilizar jogos
- Eventos de entrada
- Armazenamento
- Offline

Estas foram selecionadas por serem recomendadas pela 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 *Introduction to game development for the Web* contém uma lista de tecnologias que julga 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 serem tratadas pelos mesmos. Definiu-se que o protótipo deve funcionar nas plataformas Android e navegadores Desktop Google Chrome e Firefox em suas últimas versões, 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 por termos 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.

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 que o desenvolvimento nativo para jogos é a melhor opção, quando outras formas de aplicativos são mais adequados para a WEB Powell e Li 2013, pp. 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.

4 REVISÃO BIBLIOGRÁFICA

4.1 JOGOS

Segundo (Lemes 2009) jogo digital constitui-se em uma atividade lúdica composta por uma série de ações e decisões, limitada 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 vídeo games se encontram, uma categorização dos jogos contemporâneos é extremamente difícil de desenvolver (muitos já tentaram) (Granic, Lobel e Engels 2014, pp. 60).

Para corroborar com essa complexidade Granic, Lobel e Engels 2014 afirmam que a natureza dos jogos tem mudado drasticamente na última década, se tornando cada vez mais complexos, diversos, realísticos e sociais em sua natureza.

Com as inovações nas tecnologias relacionadas ao HTML novas mecânicas e gêneros de jogos podem ser explorados na WEB. Sendo que cada gênero acompanha um conjunto de desafios específicos. Jogos de FPS (tiro em primeira pessoa) requerem menor latência de rede, já jogos de RPG podem requerer vastas quantidades de cache (Kuryanovich et al. 2014).

Assim como as tecnologias, um grupo emergente de pesquisas sobre os benefícios do jogos vem se desenvolvendo Granic, Lobel e Engels 2014. 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.

(Granic, Lobel e Engels 2014) demostra que vídeo games melhoram as funções cognitivas, as capacidades criativas, e motivam uma visão positiva diante a falha. Também segundo (Granic, Lobel e Engels 2014) postura positiva em relação a falha correlaciona-se com melhor performance acadêmica.

Benefícios em habilidades práticas também são observados em usuários de jogos. Jogadores de jogos de tiro demonstram maior alocação de atenção, maior resolução espacial no processamento visual e melhor habilidades de rotação (Granic, Lobel e Engels 2014).

Estes aspectos positivos muitas vezes não recebem a atenção devida. Habilidades espaciais derivadas de jogar jogos de tiro comercialmente disponíveis são comparáveis aos efeitos de um curso universitário que busca melhorar as mesmas habilidades (Granic, Lobel e Engels 2014). Estas habilidades derivadas dos jogos são centrais para muitas áreas de interesse humano, Granic, Lobel e Engels 2014 afirmam que habilidades espaciais estão diretamente relacionadas com

o sucesso em ciência, tecnologia, engenharia e matemática.

Outra 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 e Engels 2014).

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

4.1.1 MECÂNICA

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

A mecânica é composta pelas regras do jogo. Quais as ações disponíveis aos usuários, e seu funcionamento, é fortemente influenciada pela categoria do jogo em questão. Dedicar-se na elaboração de uma mecânica é tarefa quintessencial para a construção de um jogo de sucesso.

(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.

A elaboração da mecânica em jogos desenvolvidos profissionalmente pode ser integrada dentro de um processo de engenharia de software [1]. Os desenvolvedores tem que evitar fazer o jogo para eles mesmos. Falta de crítica antes do desenvolvimento também tende a gerar criações 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.

Depois dos preparativos efetuados, pode-se começar a construção do jogo. Os jogos são dirigidos por um laço que executa uma série de tarefas a cada frame, construindo a ilusão de um mundo animado (Prado 2012, pp. 31). Da perspectiva da programação a parte principal de um jogo é o laço onde o jogo é executado (Pirttiaho 2014, pp. 17). A cada iteração do jogo o laço é executado e o processamento de entrada de usuário e computações correlacionadas são executadas.

Na seção Otimizações para jogos em JavaScript será abordada as possibilidades na hora de construir o laço.

[1] O sumário também conta com um resumo sobre metodologias de desenvolvimento de software contextualizada para a criação jogos

4.2 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 (Pirttiaho 2014). Além da API, dispositivos de múltiplas plataformas variam em seus recursos, capacidades e qualidade.

Desenvolvedores de jogos que almejem múltiplas plataformas, devido aos fatores acima citados, 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, tela. Por estas peculiaridades tendências como WEB design responsivo (RWD) emergiram, requerendo interfaces mais fluídas e intuitivas o possível. As tecnologias também tiveram que evoluir, como o CSS3 media queries, tamanhos relativos, entre outros.

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 abaixo.

4.2.1 JOGOS WEB

Um jogo web é um jogo que utiliza o HTML e ferramentas correlacionadas para sua construção e disponibilização.

O objetivo primário da WEB nunca foi o desenvolvimento de jogos, mas as novas versões do HTML estão provendo funcionalidades que permitem a criação de conteúdo interativo, incluindo jogos. Sendo assim, nem sempre as ferramentas desejadas estão disponíveis mas a maturidade atual do HTML5 aponta para a possibilidade de criação de jogos cada vez mais complexos e interativos.

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. Não obstante, 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).

Durante esse período o interesse em jogos WEB residia principalmente nas de casualidade, flexibilidade, e o fator social dos jogos. 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. 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, pp. 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 da 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 de aplicações desktop (Vahatupa 2014). Por serem criados à partir das tecnologias da WEB, jogos na WEB, se beneficiam de uma arquitetura construída para um ambiente social, sendo relativamente mais fácil criar experiências sociais.

A performance é um ponto negativo da abordagem WEB, é difícil chegar a performance comparável a abordagem nativa. Não obstante esse problema é cada vez menor o hardware dos dispositivos são cada vez melhores e as tecnologias tecnologias de software também avançam substancialmente.

Existem inconsistências nas implementações das especificações WEB o que leva a comportamento inesperados em alguns caos, sendo necessário desenvolver regras específicas para dispositivos e versões de navegadores.

Outro aspecto negativo da WEB é que nem todas as funcionalidades dos dispositivos estão especificados com as tecnologias da WEB e muitas vezes os recursos do dispositivo ficam sub utilizados.

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

4.2.2 DESENVOLVIMENTO DE JOGOS NATIVOS

Uma aplicação nativa é uma aplicação que foi desenvolvida para ser utilizada em uma plataforma ou dispositivo específico (Powell e Li 2013, pp. 7).

Aplicativos nativos tendem a oferecer uma experiência mais próxima com a do resto do sistema operacional o qual está 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 ver 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 e Li 2013, pp. 7).

Desenvolver jogos nativos tende a ser mais caro que a alternativa da WEB visto que é necessário duplicar funcionalidades em sistemas distintos e manter um profissional por ambiente suportado. As versões nativas são totalmente incompatíveis impossibilitando reúso.

A alternativa hibrida fica em meios ao desenvolvimento nativo e WEB e será explicada abaixo.

4.2.3 JOGOS HÍBRIDOS

A alternativa híbrida busca beneficiar-se das melhores características do nativo 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 WebWiew 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 de um software de 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 possibilitar granular controle sobre os dispositivos através de APIs JavaScript implementadas nativamente.

Outro benefício da estratégia híbrida em relação a WEB é que ela permite empacotar as aplicações com um experiência exata a dos softwares nativos. Não sendo perceptível para o usuário final a diferença ente um aplicativo híbrido e um nativo.

Não obstante, segundo Powell e Li 2013, pp. 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 especificações.

4.3 WEB

4.3.1 OPEN WEB

A OWP (*Open Web Platform*) é uma coleção de tecnologias livres, amplamente utilizadas e padronizadas. 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. «Time for SVG - Towards High Quality Interactive Web - Maps», pp. 3 cita que o tecnologias padronizadas tem um maior ciclo de vida e são mais fáceis de mudar. Da mesma maneira, as tecnologias da WEB são benéficas devido sua

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

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 (*What Is the Open Web and Why Is It Important?*). Entre outras, a Open Web busca transparência, imparcialidade nos processos de criação e padronização de novas tecnologias. Retro compatibilidade com as especificações anteriores. 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 seus próprios conjunto de ideias sobre como a WEB deveria funcionar. Mas para que a crescente quantidade de dispositivos possa acessar a riqueza que o HTML5 permite, padrões precisam ser definidos (Powell e Li 2013, pp. 5).

A W3C é uma comunidade responsável por boa parte das especificações da web como: HTML (em conjunto com a WHATWG), CSS, entre outras[4]. 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 (*workin 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 especiação 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 (**HTML5**). Não obstante, 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 robusteza e estabilidade enquanto outras tecnologias crescem e morrem ao redor dela.

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

4.4 HTML

HTML (*Hyper Text Markup Language*) é uma linguagem de marcação que define a estrutura semântica do conteúdo das páginas da web. Criada por Tim Berners Lee em 1989 no CERN. HTML 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 (*HTML HyperText Markup Language*).

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

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

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

HTML (*HTML is the new HTML5*). Não obstante, o termo HTML5 permanece em utilização pela W3C.

Além da nomenclatura, exitem pequenas diferenças nas especificações da W3C e WHATWG. 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 da W3C reflete mais a realidade dos navegadores, que nunca implementam uma versão 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 portáveis 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 inclusos 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 paginas HTML de hoje contenham ao menos um erro de estrutura (Pilgrim 2010). Uma das maiores vantagem do XML é que sistemas sem erros de sintaxe que podem ser facilmente interpretadas 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. Deve-se ao HTML ser amigável ao programador, aceitando erros de sintaxe, dependente de contexto, buscando entregar a melhor aproximação possível. Segundo Garisel e 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 fechar de uma tag e todo o conteúdo que dentro dele reside (*HTML and CSS - Design and Build Websites*, pp. 10–11). Por exemplo, na figura 4.1 o elemento meta (<meta>) tem um atributo *charset*, que especifica o formato de codificação do documento.

```
<}!DOCTYPE HTML>
<html lang="en-US">
<head>
<meta charset="UTF-8">
<title></title>
</head>
<body>
    <video>
        <span>Seu navegador não suporta vídeo</span>
    </video>
</body>
</html>
```

Fig. 4.1: Exemplo de documento HTML

Na sua versão inicial, o HTML contava com 18 elementos; atualmente existem aproximadamente cem (Pilgrim 2010). Não obstante, 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 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. O elemento *span* na figura 4.1 só aparecerá para o usuário caso seu navegador não suporta a tag vídeo.

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).

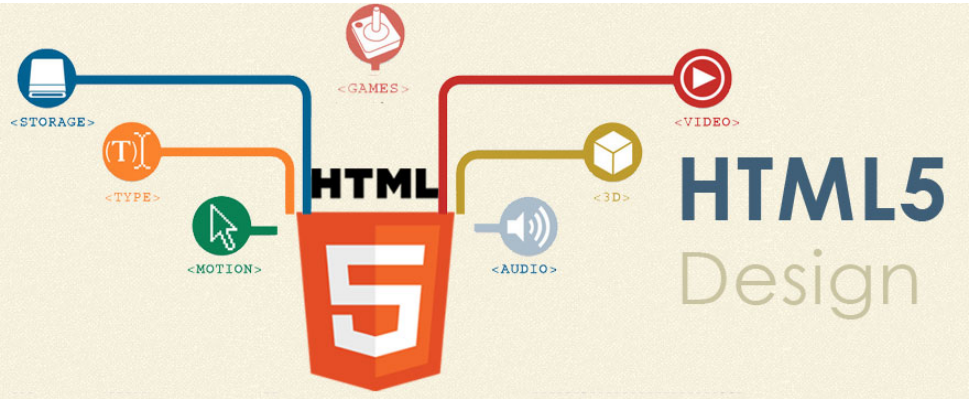

Fig. 4.2: Suíte HTML

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

4.4.1 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 especificado, permitindo a manipulação de qualquer elemento. DOM nível 2 e 3 seguiram com melhorias nas consultas aos elementos e CSS.

```
var elementos = document.querySelector( ".main, #sceen"  );
var elementosB = document.querySelectorAll( "a.minhaClasse, p"  );
```

Fig. 4.3: Exemplo de utilização de seletores do DOM em JavaScript

A API de seletores (*querySelector*) do DOM permite alto nível de precisão e performance para buscar elementos. A figura 4.3 exemplifica a utilização dos seletores do DOM em um documento JavaScript. O método *querySelector* seleciona o primeiro elemento em conformidade com o padrão especificado. Já o método *querySelectorAll* seleciona todos os elementos que estão em acordo com o padrão especificado.

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 informação do evento ocorrido (*DOM Events*).

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 varias outras APIs do HTML, como o IndexedDB, se dá através de eventos, tornando o assunto relevante aos jogos [5].

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

4.5 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 "cascateadas"para muitos elementos, e então sobrescrever os elementos específicos conforme a necessidade.

Segundo «Cascading Style Sheets», pp. 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-se 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 (*Selectors*). Seletores tem o intuito de definir quais partes do documento HTML serão afetadas por determinado bloco de declaração.

A figura 4.4 exemplifica este processo. O seletor em questão é o elemento <*p*>, simbolizando todos os parágrafos. O bloco de declaração é o que está dentro das chaves, aplicando alinhamento e cores aos parágrafos.

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

```
<style>
p {
    text-align: center;
    color: red;
}
</style>
```

Fig. 4.4: Exemplo de Folha de Estilo

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 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 as funcionalidades 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 o CSS 2.1, Seletores do CSS3 e Texto do CSS3 de recomendações para rascunhos.

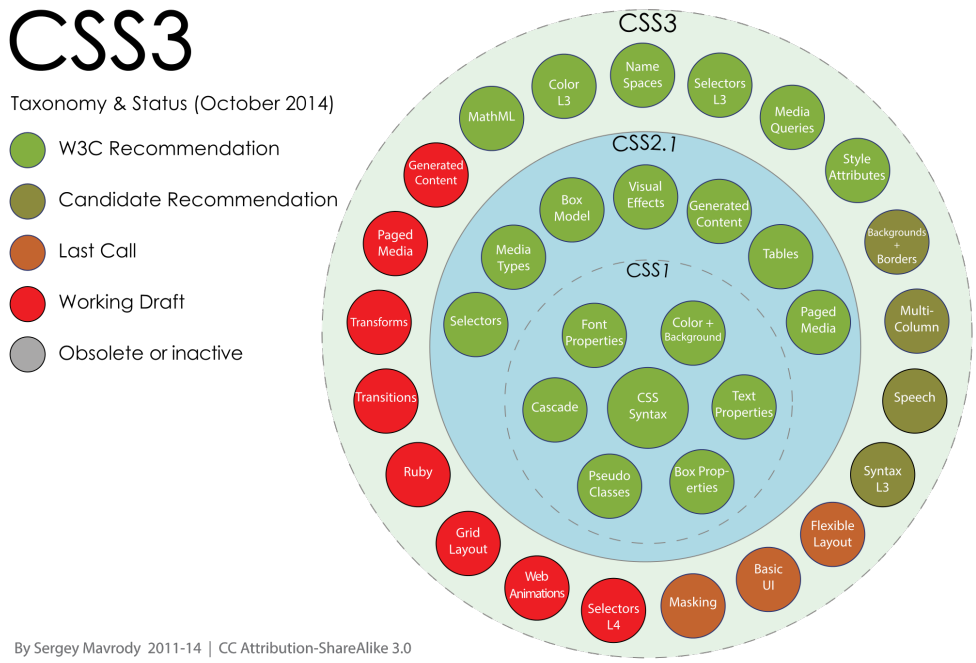

Fig. 4.5: Os módulos do CSS

Fig. 4.6: *
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.

4.5.1 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 Janiszewswi 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;
}
```

Fig. 4.7: Exemplo de Media Query

A figura 4.7 demostra a aplicação de uma regra via seletor Media Query, aplicando o a cor de fundo para dispositivos com no minimo 1024 pixels de resolução.

CSS nível 4 permite a utilização de media queries (*Custom Media Queries*) criados pelo usuário, com regras e definições customizadas. A figura 4.8 demostra 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>
```

Fig. 4.8: Exemplo de media queries customizados

Fig. 4.9: *
Fonte: https://developer.mozilla.org/en-US/docs/Web/CSS/MediaQueries

4.5.2 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 uma transição.

Para utilizar transições, assim omo 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 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;
}
```

Fig. 4.10: Exemplo de transição

A figura 4.10 demostra 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 ara 300 pixels.

Atualmente um conjunto finito de propriedades podem ser animadas com transições, e essas lista tende a mudar com o tempo, cabe ao desenvolvedor assegurar-se que determinada propriedade está disponível (*Using CSS transitions*).

4.5.3 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 4.11 escala o tamanho do elemento com a classe (*test*) para vinte porcento 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>
```

Fig. 4.11: Exemplo de transformação

4.5.4 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. Não obstante 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, não obstante 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 os mesmos estarem implementados nos navegadores. O projeto funciona compilando o código não suportado em algo compatível com versões para as versões implementadas pelos navegadores.

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

4.6 JAVASCRIPT

EMACScript, melhor conhecida como JavaScript, criada por Brendan Eich em 1992, é a linguagem de script da Web. Devido a tremenda popularidade entre 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 *ECMAScript 2015 Language Specification*:

> 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 variados outros recursos que tem 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.

Não obstante, 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 o C++ ou Java (Barnett 2014).

A última do JavaScript, o JavaScript 6, é um esforço nessa direção. JavaScript 6 ou EMACScript Harmonia, contempla vários conceitos de orientação a objetos como classes, interfaces, herança, tipos, etc. Não obstante 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.

Segundo *ECMAScript 6 Today* 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 de implementação. Além da espera por especificações, uma vez definidas, é necessário que os navegadores 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 [6]. Entretanto, essa alternativa tem seus pontos fracos, necessita-se de mais tempo de depuração,

[6] Uma lista das tecnologias para converter código HTML pode ser encontrada nos apêndices

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 a o JavaScript com recursos específicos para suas plataformas alvo. Por exemplo, para servidores, define-se objetos como: console, 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.

4.6.1 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 4.13 é a tabela de funcionalidades sugeridas e seu estágio no caminho da especificação.

Alguns dos recursos esperados para o JavaScript 7 são: guards, contratos e concorrência no laço de eventos (*A first look at what might be in ECMAScript 7 and 8*).

	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 Wagneer & 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

Fig. 4.12: Propostas do ECMA 7

Fig. 4.13: *
Fonte: https://github.com/tc39/ecma262

4.6.2 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 de código asm.js é feita 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*). Grade 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 JavaScript da

Mozilla, SpiderMonkey, é a exceção, implementando a grande maioria dos recursos do asm.js.

4.6.3 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 conta com um grupo de trabalho. Não obstante, 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 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 traz JavaScript 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.

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.

4.6.4 Web Animations

Web Animations é uma especificação em rascunho que define uma forma imperativa de manipular animações através de JavaScript. Como demostrado na figura 4.15 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;
});
```

Fig. 4.14: Exemplo de utilização de WEB Animatios

Fig. 4.15: *
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.

4.7 NAVEGADORES

Navegadores são aplicações, onde as tecnologias da OWP são interpretadas e geram um conteúdo útil para os usuários. Navegadores geralmente 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 a plataformas móveis pois necessitam maior interação do 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 operacional. Algumas formas de contornar este problema serão descritos nas seção de Disponibilização da Aplicação.

Uma vez localizado o endereço o navegador manda uma mensagem em HTTP requisitando o conteúdo de determinado endereço. 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 os motores de renderização.

Alguns motores de renderização incluem:

- Blink: Utilizado no Opera, Google Chrome e projetos relacionados;
- Gecko: Utilizado nos produtos da Mozilla;
- KHTML: Utilizado no navegador Konkeror, 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 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 e 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 que compõem um navegador implementam partes da especificação do HTML. E, operando em conjunto, tentam comportar todas as tecnologias da WEB. Infelizmente a forma que as tecnologias são suportadas varia e algumas não estão presentes de qualquer forma nos navegadores. Não obstante o suporte vem crescendo. A figura 4.21 apresenta o gráfico de suporte por versões de navegadores em dezembro de 2015.

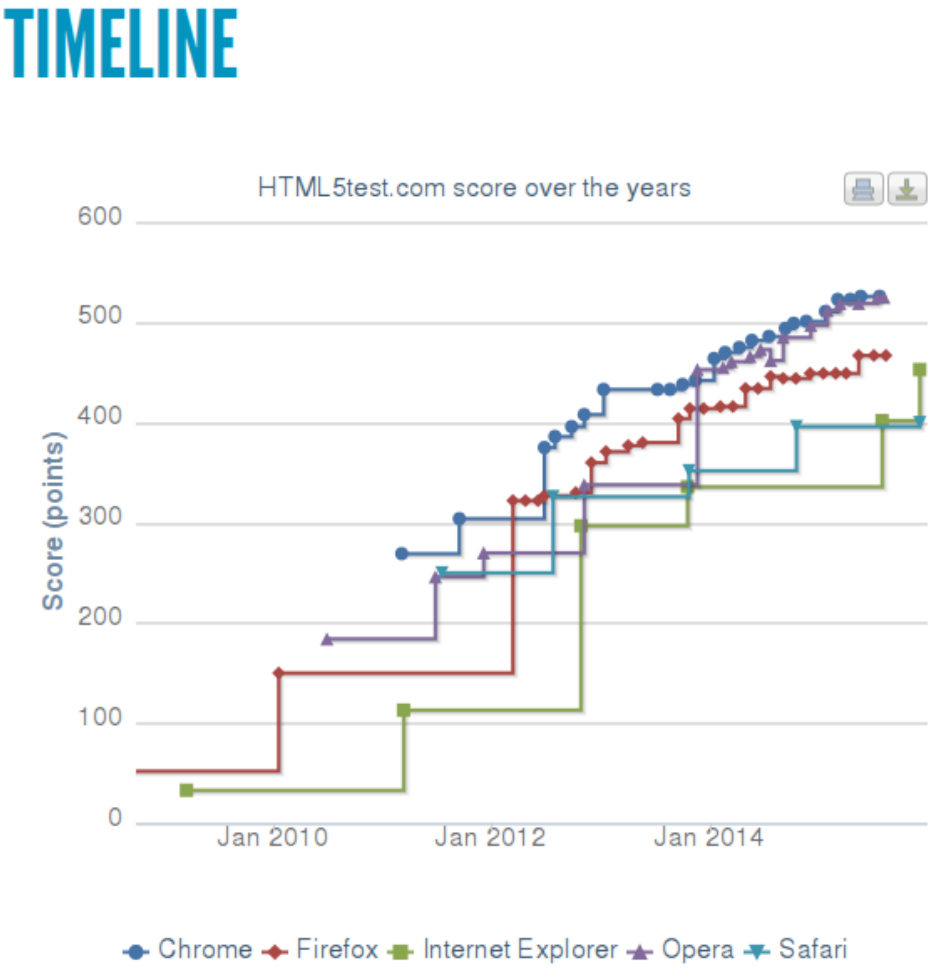

Fig. 4.16: Suporte das especificações do HTML nos navegadores

4.8 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 desde celulares a tablets, netbooks a computadores desktop, mas seu foco é dispositivos com tela sensível (*Chrome OS vs. Android: What's the difference?*). 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, para executar as tecnologias da OWP.

Além da arquitetura híbrida, é possível executar jogos WEB em dispositivos Android através dos navegadores presentes nestes aparelhos. A versão do Google conta com o Google Chrome como navegador padrão. Não obstante, a versão AOSP contém um navegador próprio que utiliza o motor de renderização LibWebCore, baseado no WebKit («Comparison of Modern Mobile Platforms from the Developer Standpoint»). Além do padrão outros navegadores como o Firefox ou Opera também podem ser instalados.

Além do Android o sistema IOS é de grande relevância mercadológica para o desenvolvimento de jogos. Não obstante, 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.

4.9 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 qual quais navegadores implementam o quê. Uma

boa referência do suporte a recursos nos navegadores é o site `http://caniuse.com/`.

4.10 RENDERIZAÇÃO

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

4.10.1 SVG

SVG (*Gráficos de vetores escaláveis*), é uma linguagem baseada em XML especializada na criação de vetores bidimensionais (Kuryanovich et al. 2014). Segundo «Time for SVG - Towards High Quality Interactive Web - Maps», pp. 4 svg foi criada em conjunto por empresas como: Adobe, Apple, AutoDesk, entre outas, 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 («Time for SVG - Towards High Quality Interactive Web - Maps», pp. 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 («Time for SVG - Towards High Quality Interactive Web - Maps», pp. 5).

Para ilustrar a utilização, a figura **??** 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>
```

Fig. 4.17: Círculo em SVG.

Fig. 4.18: *
Fonte: http://www.w3schools.com/svg/

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

4.10.2 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, pp. 113). Mais especificamente, o retângulo do canvas é um espaço vetorial cuja origem se da na esquerda superior. Normalmente cada unidade do plano cartesiano corresponde a um pixel no canvas (*Drawing shapes with canvas*).

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 (*Drawing shapes with canvas*). 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 instancia do objeto *Path2D*. Todos os métodos relacionados aos caminhos como o *moveTo*, *arc* ou *quadratic-*

CurveTo estão disponíveis no objeto Path2D (*Drawing shapes with canvas*). 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*, bastante similar aos títulos antigos da SNES e Game Boy (*Introductory Guide to Building Your First HTML5 Game*).

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);
```

Fig. 4.19: Canvas

Fig. 4.20: *
Fonte: http://www.w3schools.com/html/html5_canvas.asp

A figura 4.20 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.

4.10.3 WEBGL

WebGL é uma API JavaScript otimizada 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 desenvolvido baseando-se na especificação OpenGL a qual trata de definir como renderizar gráficos multiplataforma. Especificamente OpenGL ES, 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 Kronos. Os primeiros rascunhos do WebGL iniciaram em 2006, não obstante o grupo de trabalho não foi formado até 2009 e a primeira versão do foi lançada em 2011.

Apesar de ter sido desenvolvido com foco em 3D, WebGL pode ser igualmente utilizado para criação de gráficos em duas dimensões(Matti e Arto 2011, pp. 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.

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 vídeo games.

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 e Arto 2011, pp.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 site interessante para explorar exemplos WebGL avançados é o blog `http://learningwebgl.com` que conta com tutoriais cobrindo áreas como diferentes tipos de iluminação, carregamento de modelos em JSON, gerenciando eventos do mouse e teclado; e como renderizar uma cena WebGL em uma textura (Matti e Arto 2011, pp.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 OpenGL é o WebVR.

4.10.4 WEBVR

Realidade virtual é uma área nem tão nova mas que recebeu interesse renovado recentemente. Isso se dá, pelo menos me 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 interessa para os produtores de jogos, pois pode oferecer alto nível de imersão nos já interativos e desafiadores ambientes dos jogos.

A 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 movimentos para dentro do contexto de uma um contexto 3D através de JavaScript.

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.

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

4.11 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 e 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.

4.12 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 enorme quantidade de dados que seriam muito grandes para serem transmitidos pela Internet em um período de tempo razoável (Lubbers, Brian e Frank 2010, pp. 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. A maioria dos codecs de vídeo não mudam todo o 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. Não obstante, também existem opções livres de patentes e licenças.

Abaixo segue uma lista de alguns codecs populares para áudio segundo Lubbers, Brian e Frank 2010, pp. 67.

- ACC
- MPEG-3
- Vorbis

Ainda segundo Lubbers, Brian e Frank 2010, pp. 67 abaixo segue uma lista de codecs populares para vídeo.

- H.264
- VP8
- OggTheora

Após ter sido codificado um fluxo de dados 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 funcionem em todos os navegadores [8].

[8] A figura 4.21 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).

4.13 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 do jogo. 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 utilizar áudio na WEB: através do elemento HTML áudio ou através da API JavaScript de áudio.

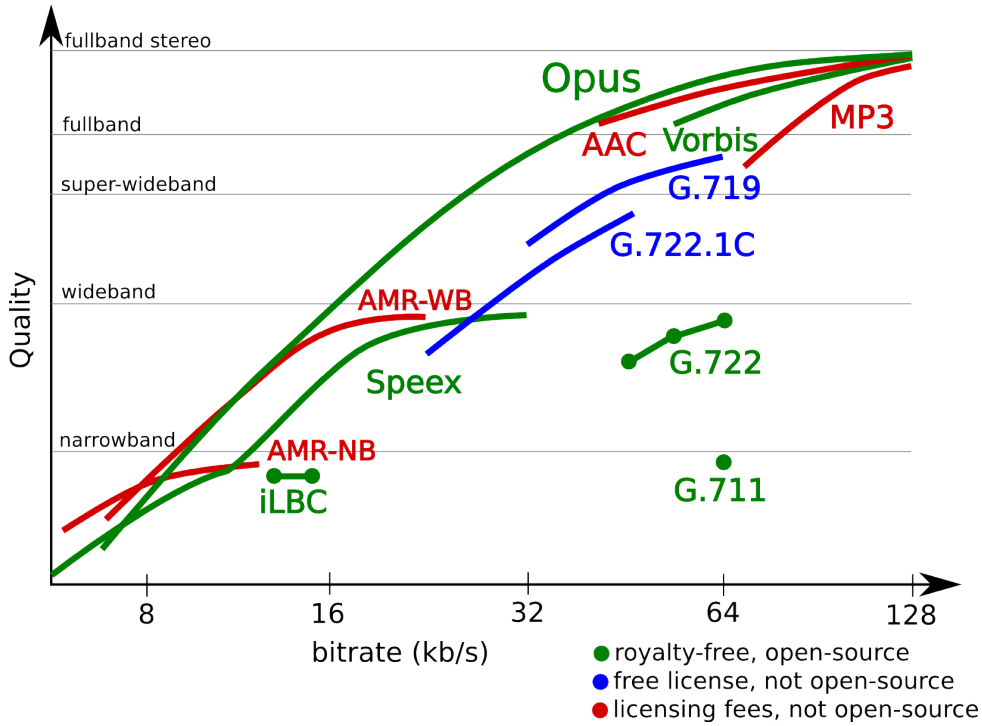

Fig. 4.21: Comparação de codecs de áudio

Fig. 4.22: *
Fonte: https://www.opus-codec.org/comparison/

4.13.1 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>
```

Fig. 4.23: Exemplo de utilização da tag áudio

Fig. 4.24: *
Fonte: http://www.w3schools.com/HTML/HTML5_audio.asp

A imagem 4.23 demonstra a utilização da tag *audio*. Os elementos *source* demonstrados na figura referenciam a arquivos de áudio contendo um contêiner e codec específicos. 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.

A especificação declara que todo o conteúdo dentro de uma tag audio (e video também), que não sejam tags *source*, sejam ignoradas 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. A figura 4.23 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, como um som de fundo. Por conseguinte, a tag *audio* não é o suficiente para comportar aplicações de áudio complexas (*Web Audio API*). 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.

4.13.2 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, funcionalidades que estão presentes nas aplicações de processamento de áudio modernas para desktop (*Web Audio API*).

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 cada inferior (tipicamente código Assembly / C / C++), mas síntese e processamento em JavaScript também será suportado (*Web Audio API*).

Essa tecnologia é muito mais nova do que a tag áudio. 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 utilizada nos casos onde não existe suporte para a

API do HTML [9].

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

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.

4.14 VÍDEO

O elemento *video* especifica 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>
```

Fig. 4.25: Exemplo de utilização de vídeo

Fig. 4.26: *
Fonte: http://www.html5rocks.com/en/tutorials/video/basics/

A figura 4.26 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 na figura 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.

[9] O polyfill AudioContext pode ser encontrado no seguinte endereço https://github.com/shinnn/AudioContext-Polyfill

Além de ser flexível nas tecnologias de multimídia jogos 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.

4.15 ARMAZENAMENTO

Uma das grades 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 ode as aplicações nativas detinham grande vantagem sobre as aplicações web. O HTML5 solucionou este problema introduzindo várias formas de armazenamento de dados («Smart Phones Application development using HTML5 and related technologies: A tradeoff between cost and quality»).

Armazenamento local é um recurso importante para jogos, tanto por diminuir a latência da persistência na rede, quanto para possibilitar um experiência offline.

Existem algumas especificações sobre armazenamento, mas a grande parte delas não conta como 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.

4.15.1 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 megabytes, o qual pode ser estendido pelo usuário.

O grupo de trabalho do Web SQL iniciou-se em 2010 e foi suspendido 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.

Não obstante, 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.

4.15.2 WEB STORAGE

Web Storage, também conhecido como Local Storage, provê uma forma de armazenar os dados como chave valor dentro do navegador. Os dados são persistidos mesmo que o navegador seja fechado.

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.

A API do Web Storage é simples, consistindo em uma interface para buscar dados e outra para armazenar, no formato chave/valor.

```
// 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'));
```

Fig. 4.27: Web Storage na prática

Fig. 4.28: *
Fonte: https://en.wikipedia.org/wiki/Web_storage#usage

A figura 4.27 exemplifica a utilização do Web Storage, para utilizar o armazenamento de 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. Não obstante 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.

4.15.3 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. Não obstante, a versão síncrona foi removida devido a falta de interesse da comunidade. Operações assíncronas permitem que aplicativos JavaScript nunca 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 setada para *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')
};
}
```

Fig. 4.29: Adicionando um cliente em IndexedDB.

A figura 4.29 demonstra um exemplo simplificado da utilização do IndexedDB, como cada iteração com o banco de dados é construído através de uma nova requisição e o tratamento do resultado é dado dentro de eventos.

Apesar de ser desenvolvido com objetivo de ser uma solução para todas as necessidades de armazenamento no Frontend IndexedDB ainda sofre algumas limitações.

Abaixo segue uma lista com algumas das limitações do IndexedDB.

- Tem limites de armazenamento e as regras variam de navegador para navegador.
- O comportamento em abas anônimas não está especificado e os resultados também variam.
- Existe uma pequena probabilidade de os dados se perderem, no caso do Firefox a API não espera confirmação do sistema operacional para considerar um dado válido, essa foi

uma escolha em detrimento de performance.

- Não existe a possibilidade de fazer buscas em textos como o *LIKE* do SQL.
- o usuário pode configurar o navegador para não aceitar armazenamento local para determinado domínio.

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 o Web Workers.

4.16 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 seja 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.

4.17 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 da gestão da aplicação offline (*The Step-by-Step Guide to Publishing a HTML5 Mobile Application on App Stores*).

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 4.31 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 exclusivamente da rede e serão buscado todas as vezes.

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/backgound.png

NETWORK:
server.cgi
```

Fig. 4.30: Exemplo de arquivo de manifesto offline

Fig. 4.31: *
Fonte: http://www.w3.org/TR/offline-webapps/

4.18 ENTRADA DE COMANDOS

Na construção da grande maioria dos jogos é muitas vezes imprescindível grande flexibilidade na gestão de entrada comandos. Esta necessidade amplia na criação de jogos multiplataforma, em determinadas plataformas a entrada de comandos pode-se dar través 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 o 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 (*Introductory Guide to Building Your First HTML5 Game*).

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 4.32 demostra a captura 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);
```

Fig. 4.32: Utilização dos eventos do teclado

Fig. 4.33: *
Fonte:
http://nokarma.org/2011/02/27/javascript-game-development-keyboard-input/

4.18.1 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 (*Gamepad*).

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.

Exitem 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 principal *Gamepad*. Cada objeto botão contém um atributo *pressed* que pode ser utilizado para saber se foi pressionado.

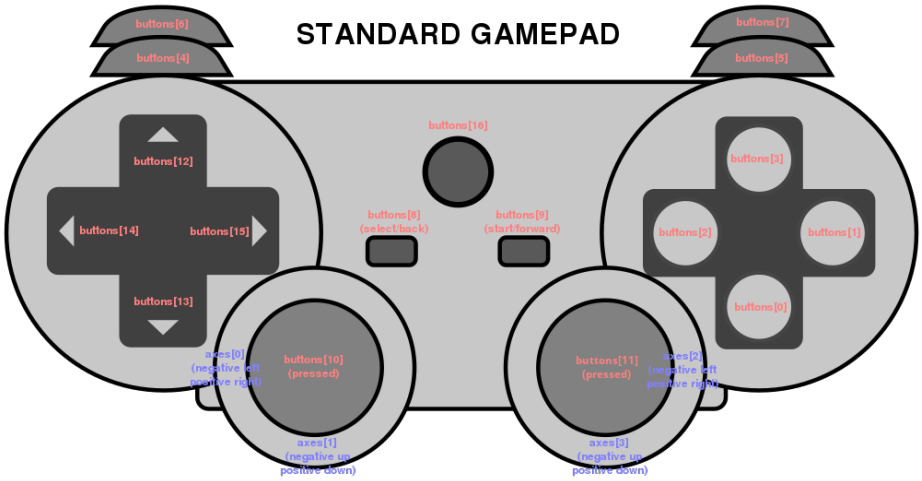

Fig. 4.34: Objetos de um Gamepad

Fig. 4.35: *
Fonte: https://w3c.github.io/gamepad

A figura 4.34 contém os objetos tradicionais de um Gamepad.

4.19 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* prove a orientação do dispositivo expressa como uma série de rotações a partir de um ponto de coordenadas locais (*DeviceOrientation Event Specification*). Para colocar claramente, a 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.

4.20 HTTP/2

4.21 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.

Não obstante, 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.

The Step-by-Step Guide to Publishing a HTML5 Mobile Application on App Stores descreve os passos que são feitos pelo PhoneGap 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.

4.22 TRABALHOS SIMILARES

Barnett 2014 elaborou uma revisão de aspectos do HTML5 através da construção de um jogo. O autor foca muito nos aspectos de criação de jogos e feedback do desenvolvimento. Troca de tecnologias e não especificamente nas limitações conforme o meu trabalho. Em outras palavras seu escopo é mais genérico e não tão preciso quanto este

Powell e 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 de jogos. Outra diferença substancial é que o autor foca apenas na viabilidade não ressaltando as limitações da plataforma.

Pirttiaho 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 projeto se propõe.

Morony 2013 estuda a viabilidade de aplicações comerciais multiplataforma em HTML5 através da construção de um aplicativo comercial em Sencha 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 aplicações genéricas.

5 METODOLOGIA

O primeiro passo consiste na definição das plataformas alvo do trabalho. Estas devem ser 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 comummente ligados à limitações multiplataforma. Segue-se uma pesquisa para aprofundar teoricamente cada um dos recursos, possivelmente elegendo novos.

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

Com o protótipo concebido, o passo que segue é a enumeração, e descrição das limitações detectadas no processo de desenvolvimento e testes do jogo. Este detalhamento deve responder perguntas como:

- Quais as limitações são comuns desenvolvimento de jogos em HTML5?
- Em quais plataformas?
- Sob quais circunstâncias?
- Quais limitações podem ser contornadas? Como?

6 PROJETO

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

6.1 MECÂNICA

Para a análise prática das limitações foi escolhido um jogo de matemática simples. Consistindo na geração de equações 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 quão 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 HTML.

Jogos como o Math Workout e o Countdown para Android tem uma temática similar. Não obstante, 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 à 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 o jogo. A figura 6.3 demonstra a interface contendo os botões descritos.

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

6.2 Requisitos

6.2.1 Requisitos funcionais

As funcionalidades que o sistema deve apresentar estão descritas abaixo.

- 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;

6.2.2 Requisitos não funcionais

Outros aspectos requisitados mas que todavia 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.

6.3 Modelagem

A figura 6.1 apresenta o diagrama de classes simplificado [1].

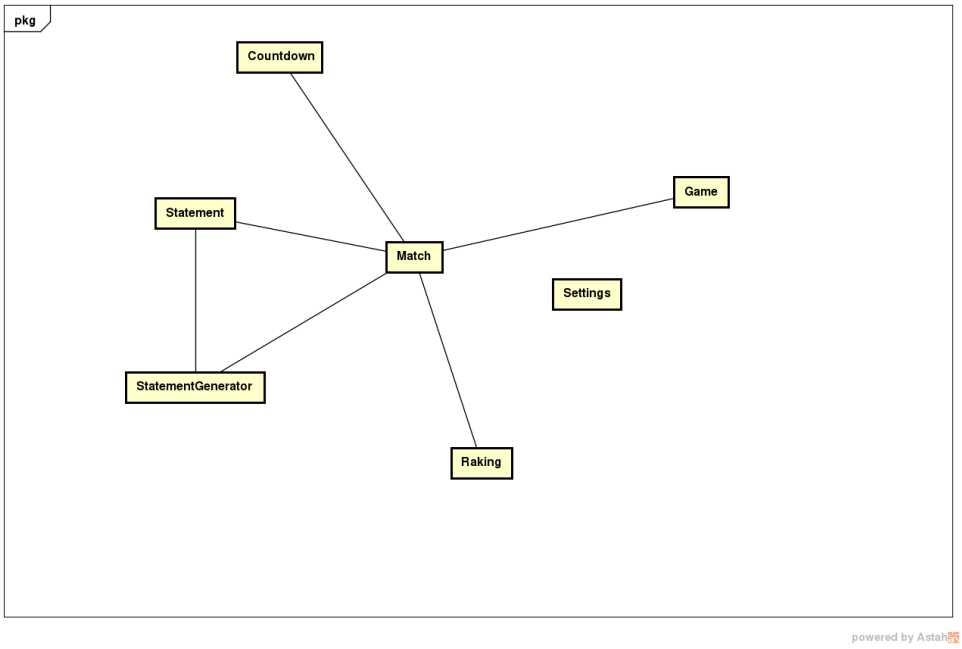

Fig. 6.1: Diagrama de classes simplificado

6.4 Desenvolvimento

O desenvolvimento se deu com uma postura de melhoria progressiva criando primeiramente a versão mais simples possível para atingir as requisitos funcionais. A partir dessa versão, novos

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

recursos foram sendo adicionadas para melhorar a experiência do usuário. Iniciei o desenvolvimento criando para a plataforma Desktop pois esta contém um grande número de ferramentas de desenvolvimento nativas que não requerem integração especial.

O primeiro passo foi a criação do documento HTML, foram utilizados elementos div para simbolizar telas do jogo. Conter todas as telas em único HTML caracteriza o jogo como uma aplicação de uma única página (*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.

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 da SPA é 98% do tamanho total disponível no dispositivo. Já o tamanho da fonte do quadro principal, representado pelo id *billboard*, 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 dela ser simples, fazer os elementos se alinharem em diversos tamanhos de telas não é fácil e pode se tornar um problema substancial para interfaces realmente complexas.

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 querem mostrar.

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

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 tela é não deixar o usuário sem feedback enquanto todos os recursos necessários para a utilização do jogo estejam disponíveis. Não obstante 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 clicar nas configurações e nos botões de certo e errado da página principal. Deixando a página pronta para receber a funcionalidade propriamente dita.

Com esqueleto da aplicação definido foi introduzido a lógica de negócio. A figura 6.1 apresenta do 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 e o tempo total da partida bem como a pontuação atual do usuário. O propósito da classe Match é iterar a cada pergunta e esperar por uma resposta. Quando a resposta é dada a classe Match computa se a resposta 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. A figura 6.3 demostra o mecanismo da classe *Match* integrado ao jogo.

O cálculo da pontuação se dá por uma operação matemática simples. Ao final de uma partida é computado o total de questões acertadas vezes 10 divido pelo total de respostas menos o total de respostas certas adicionando-se 20% do tempo da duração em segundos. A figura 6.2 apresenta o código utilizado para o cálculo acima descrito.

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

Fig. 6.2: Exemplo de utilização de funções imediatamente invocadas

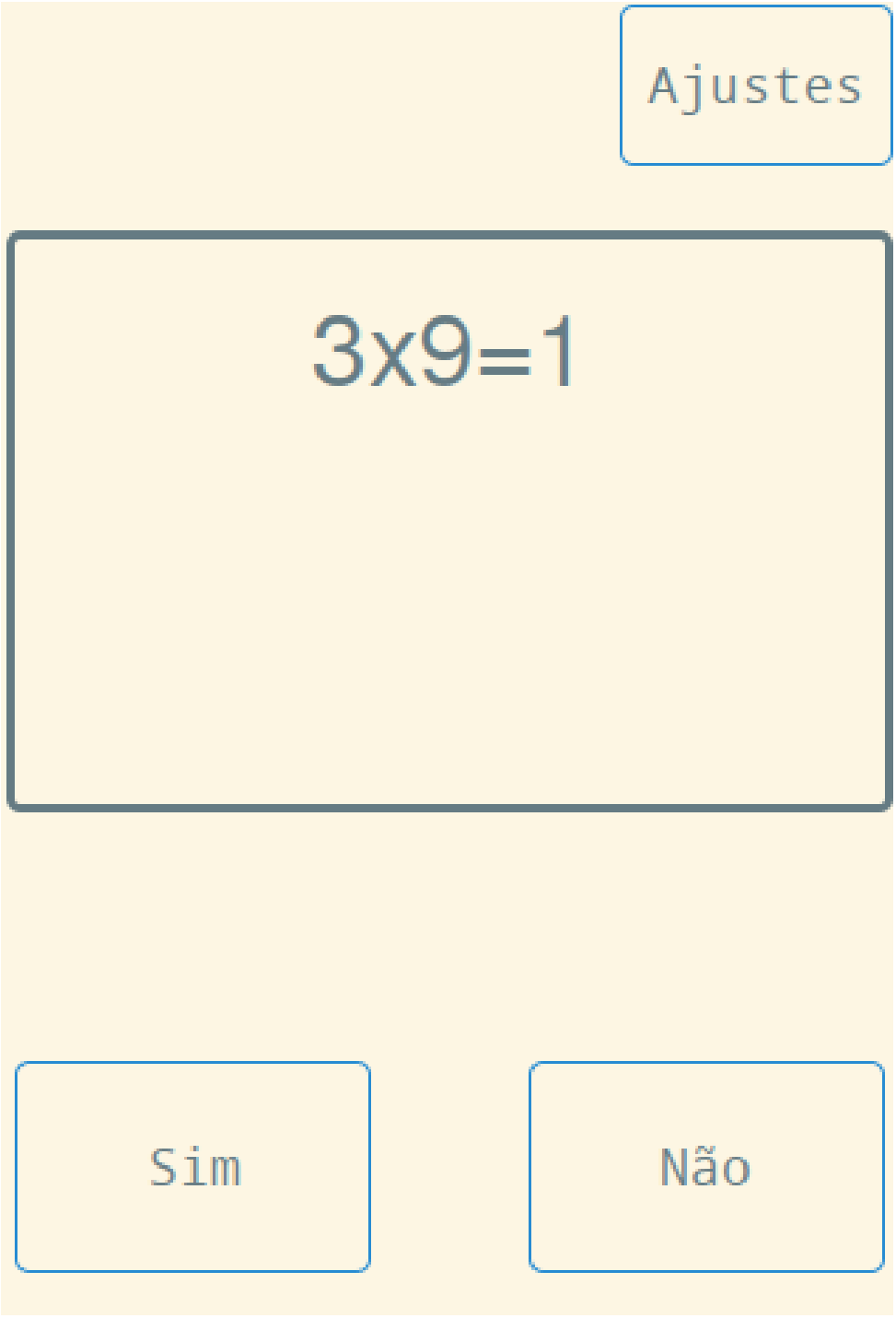

Fig. 6.3: Interface do jogo com equação sendo apresentada

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 da performance do jogador. 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. Não obstante, a interface totalmente orientada a eventos do IndexedDb introduz uma camada de complexidade desnecessária para os casos simples. Visto que os requerimentos do protótipo não demandam grande performance ou armazenamento massivo de dados a opção modesta, Local Storage, foi preferida. A classe ranking é simplesmente uma interface para converter partidas e armazenar e recuperar estas informações em Local Storage. A figura 6.4 demostra as informações do resultado de uma partida bem como a posição do ranking.

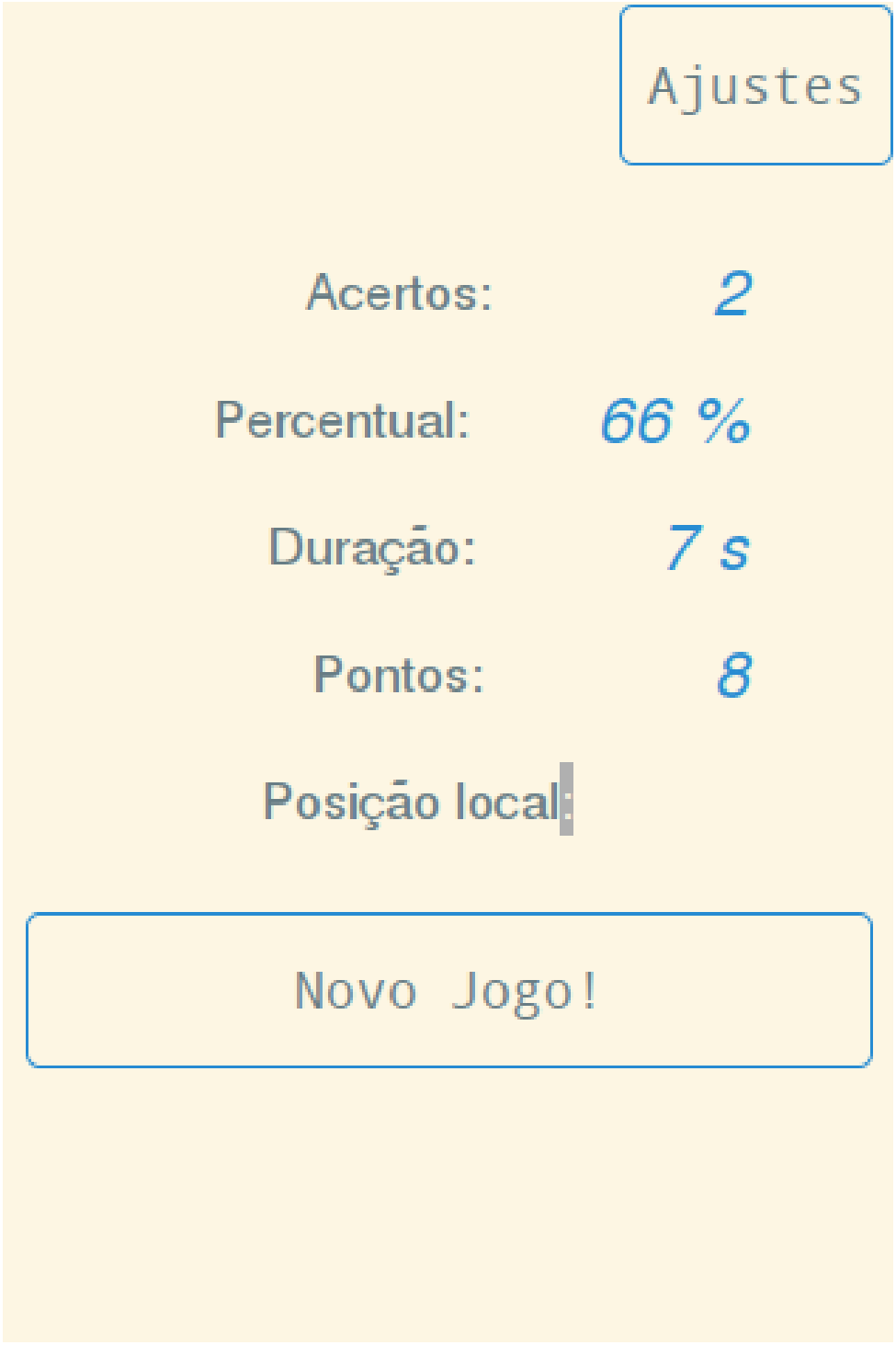

Fig. 6.4: Resultado de uma partida

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 6.5 demonstra a tela de configurações do jogo.

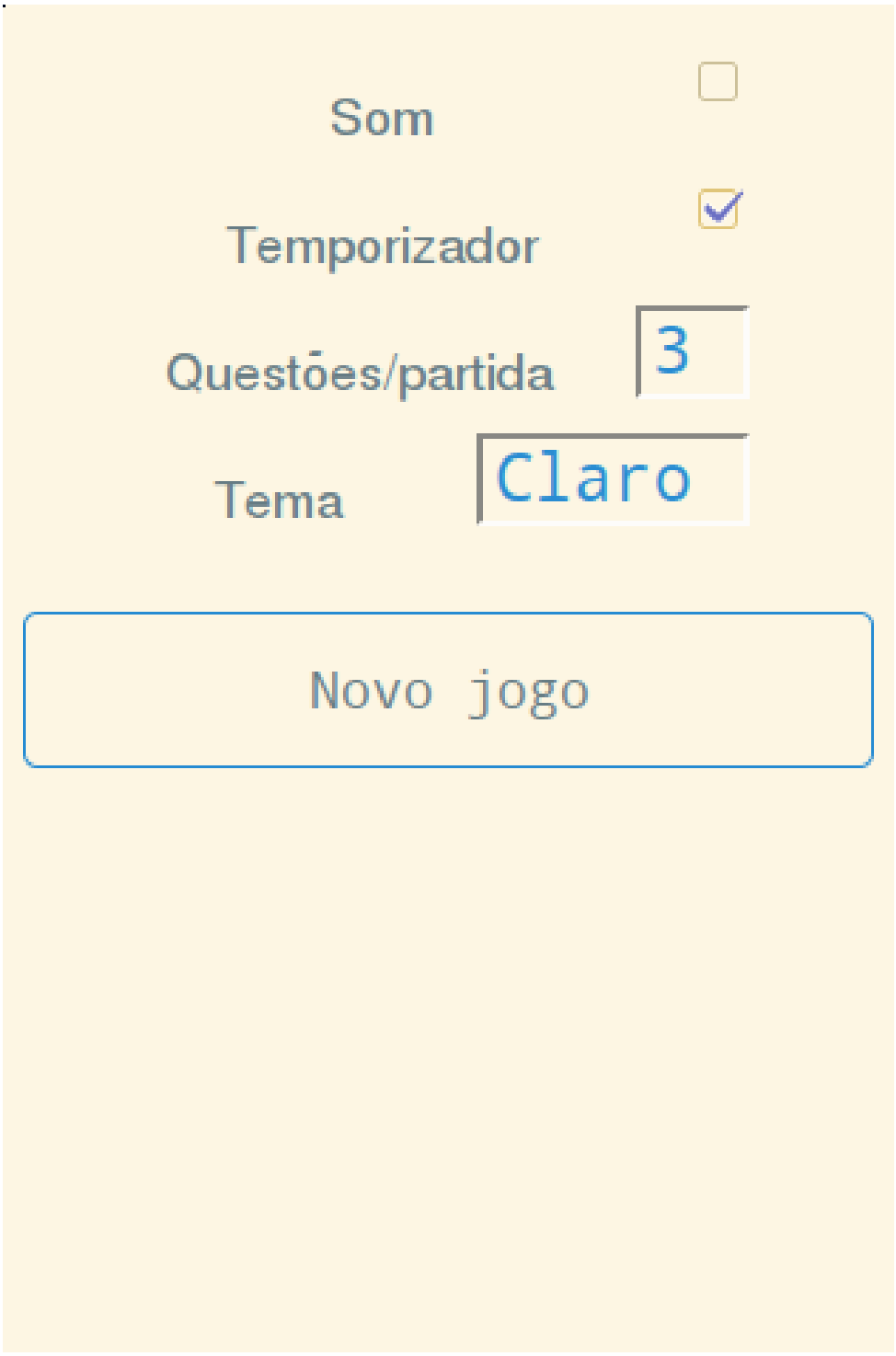

Fig. 6.5: Configurações do jogo

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 (a reposta da afirmação). O objeto *StatementGenerator* é responsável por gerar instâncias da classe *Statements*.

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 iteração do 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 qual operador será utilizado 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), seguido da 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 para a 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 circulo com uma mensagem no centro, neste caso os números do temporizador. A figura 6.6 demonstra o contador prestes a iniciar uma nova partida.

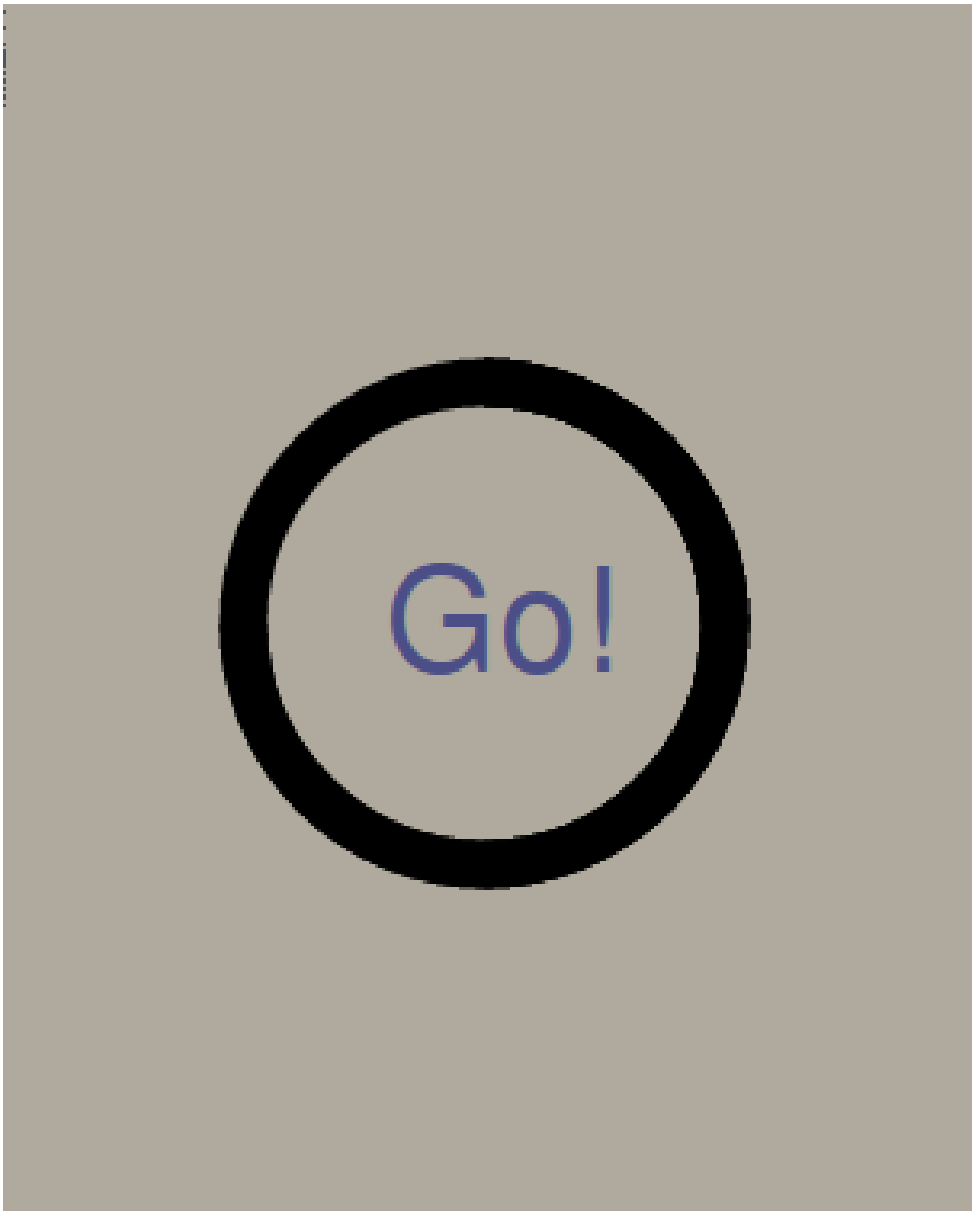

Fig. 6.6: Contador em Canvas

Para melhorar a experiência 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 ao invés acionar duas vezees o tratamento das escolhas de resposta foi uma boa estratégia pois centralizou o tratamento evitando duplicação de código.

No mobile, com intuito de melhorar a experiência, foi adicionada vibração 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/beneficio.

Durante o desenvolvimento foram utilizadas funções imediatamente invocadas para declarar as classes. Isso se demonstrou uma boa forma de separar os objetos, tornando o conflito de

variáveis globais um problema inexistente. Outro aspecto positivo foi a utilização de um meta objeto para encapsular os demais, similar ao conceito de namespaces, neste caso utilizei o nome MyMath garantindo que problemas de conflitos de nomes não aconteçam.

Ao final do desenvolvimento foi feita a integração de Grunt com plugins de minificação do JavaScript e CSS [2]. O grunt foi configurado para disponibilizar a aplicação para distribuição dentro do diretório *dist/web*.

Após de a disponibilização WEB estar funcionando, foi feita a integração para o PhoneGap build. Para tanto foi utilizado um makefile que simplesmente copia os arquivos de distribuição da WEB, gerados pelo grunt, e o arquivo de metadados do Phonegap Build e cria a árvore de arquivos padronizada que o phonegp build requer.

6.4.1 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 a velocidades de redes.

Utilizado os arquivos minificados, gerados pelo Grunt, o tempo médio de download ficou em 1.1 segundos. Uma diferença substancial dada a quantidade pequena de arquivos minificados 8 arquivos JavaScript e um arquivo CSS.

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

6.5 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 para jogos que foram identificadas através da revisão e desenvolvimento do protótipo.

6.5.1 CSS

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

A barra de endereço é outro recurso de pouca utilidade no contexto de jogos, e muitas vezes um empecilho para jogos em dispositivos móveis, devido ao limitado tamanho da tela.

[2] Para mais informações sobre o Grunt veja os apêndices

Para desabilitar a barra em dispositivos da Apple pode-se utilizar a seguinte configuração:

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

Para os demais dispositivos não existe meio oficial de esconder a barra de endereço. Não obstante, alguns sites recomendam a solução descrita abaixo:

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

Apesar de não fazer parte da especificação, a maioria dos navegadores implementa a possibilidade de desativar a seleção de elementos na tela. Em jogos essa possibilidade é útil, pois não é natural a seleção de texto neste tipo de software. Kuryanovich et al. 2014 cita que desabilitar a seleção de texto em jogos é uma otimização importante para a experiência do usuário. Para desabilitar pode-se utilizar as regras CSS demonstradas abaixo.

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

6.5.2 JavaScript

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 é variáveis não prefixadas pela palavra chave *var*.

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

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 sobrescrita de variáveis. Para contornar esse problema pode-se utilizar as funções imediatamente invocadas IFE (*Immediatly invoked function expression*).

```
(function() {
    'use strict';

    function bar() {
        return 'foo';
    }

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

Fig. 6.7: Exemplo de utilização de funções imediatamente invocadas

A figura 6.7 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 *window.settimeout* ou *window.setInterval*. Não obstante, a forma mais recomendada é utilizar o *window.requestAnimationFrame* reduz ou completamente para a execução do laço enquanto o usuário está em outra aba. Isso reduz o consumo de bateria, uma característica importante para dispositivos móveis.

HTML

Um problema que jogos sofrem em geral é a demora no carregamento da grande quantidade de recursos que precisam estar em memória para o jogo funcionar. Muitos jogos utilizam uma tela de carregamento enquanto os recursos são adquiridos.

Uma funcionalidade do HTML interessante para este tipo de situação 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, pp. 39).

7 RESULTADOS

Muitos das limitações dos jogos multiplataforma não são problemas específicos dos jogos, mas aplicam-se a todos tipos de software (Bruins 2014, pp. 3). Alguns problemas são inerentes da categoria multiplataforma Morony 2013, pp. 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 as limitaçõesd do HTML5 aplicáveis, mesmo que não exclusivamente, aos jogos encontradas 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 limitaçãos como performance e versões foram tratadas em uma partes separadas visto que se aplicam a várias das tecnologias da WEB.

7.1 CSS

É muito custoso desenvolver uma interfaces que pareçam nativas para cada dispositivo sem a utilização de plugins e ferramentas. No protótipo foi utilizada uma estilização simples, que pode se parecer natural na WEB; não obstante, nos dispositivos móveis, o layout criado é muito diferente da experiência normal nestes aparelhos.

Uma solução para este problema é utilizar frameworks como o jQuery Mobile e Kendo UI Mobile [1]. Estes frameworks permitem criar elementos típicos de interfaces mobile como listas com scroll, botões e transições com uma aparência nativa de forma relativamente fácil (*The Step-by-Step Guide to Publishing a HTML5 Mobile Application on App Stores*).

Em alguns casos o tamanho das telas pode ser um fator limitante – como no caso de jogos de estratégia. Jogadores com telas menores podem sair em desvantagem.

Para muitos caso pode-se contornar o problema de tamanhos diferenciados trabalhando com tamanhos relativos via CSS. Não obstante, há casos, como o dos botões de certo e errado, em que a proporções ficam exageradas, nesses casos utilizar controles como o *max-width* e *min-width* é uma solução conveniente.

Tecnologias experimentais do CSS geralmente levam o nome do distribuidor como prefixo. 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

[1] Mais informações sobre o Kendo UI e jQuery Mobile podem ser encontradas nos apêndices

necessários e adicionado em tempo de execução [2].

7.2 JavaScript

O JavaScript, por ser uma tecnologia desenvolvida por consenso, tem um ciclo de vida de atualizações demorado; pois necessita que todos os consumidores da especificação entrem em consenso e implementem o concordado.

Por ser uma linguagem interpretada, erros tem de descobertos rodando a aplicação. Alternativamente se JavaScript fosse compilado vários problemas poderiam ser capturados e informações úteis reveladas antes de se testar (Morony 2013, pp. 12).

Aplicações para produção muitas vezes necessitam ser minificadas por performance e ofuscação. Sem a utilização de automatizadores como o Grunt Watch essa tarefa é entediante e propensa a erros.

O sistema de tipos do JavaScript também é problemático. Erros numéricos resultam no valor NaN (*not a number*). Todas as operações com NaN como operadores irão retornar outro NaN. Isso torna a depuração de erros desnecessariamente complexa (Kuryanovich et al. 2014).

Ao testar o tipo (*typeof*) de uma variável vazia o JavaScript retorna como se fosse um objeto.

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

Fig. 7.1: Função para testar tipos que funciona como o esperado.

Fig. 7.2: *
Fonte: `http://www.slideshare.net/fdaciuk/javascript-secrets-front-in-floripa-2015`

A figura 7.3 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 - utilizando-se deste valor pode-se deduzir o tipo da variável corretamente.

Segundo «HTML5 Research Journal», um problema do JavaScript é que não é possível transferir métodos de objetos através de sistemas, via WebSocket, somente dados. 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.

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 otimizados e acelerados via hardware (Kuryanovich et al. 2014).

[2] Mais informações sobre a biblioteca pode ser encontradas nos apêndices

O WEB Assembly pode resolver este problema, substituindo o JavaScript para os casos onde grande performance é necessária. Mas para isso o Web Assembly precisa evoluir na especificação e implementações.

7.2.1 Fullscreen

Não existe forma padronizada de detectar se uma aplicação está em tela cheia ou não através de JavaScript. O IOS suporta a variável *navigator.standalone* para identificar se a aplicação está em tela cheia. **homescreenwebapps** recomenda a utilização do trecho de código para determinar se está em modo tela cheia nos demais navegadores.

```
navigator.standalone = navigator.standalone || (screen.height-document.documentElem
```

Fig. 7.3: Teste de tela cheia

Fig. 7.4: *
Fonte:
http://www.mobilexweb.com/blog/home-screen-web-apps-android-chrome-31

7.3 SVG

Uma das grandes vantagens do SVG é que é definido via linguagem de marcação e se integra bem com as demais tecnologias da WEB. Não obstante, isso também implica em problemas de performance para arquivos muito grandes. Segundo Kuryanovich et al. 2014 a grande desvantagem do SVG é que quão maior o documento mais lenta a renderização.

Controle refinado também é um problema do SVG Kuryanovich et al. 2014 cita que um aspecto negativo do SVG é que é muito difícil atingir a perfeição na posição dos pixels, por ser uma linguagem vetorizada

7.4 Canvas

A literatura aponta várias limitações distintas na API do canvas. Segundo (Kuryanovich et al. 2014), os aspectos negativos do canvas é que a performance varia de plataformas para plataformas e não existe implementação nativa para animações.

O problema de performance mencionado 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. Não obstante, o FastCanvas não suporta a especificação do canvas completamente, e existem umas diferenças no comportamento de de ambos.

A implementação de animações de fato não existe no canvas, similarmente carece-se de integração com as demais tecnologias da WEB. 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 no 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, poderia se utilizar os eventos do DOM para capturar interações do usuário através de elementos utilizados no canvas. Controle similar é implementado atualmente acessando as coordenadas do canvas onde uma interação aconteceu e fazendo o processamento com aquela determinada área.

Não obstante 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 as tecnologias cada vez úteis e dinâmicas.

Uma 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 do documento escala a superfície de desenho para preencher o elemento, o que pode gerar resultados inesperados.

7.5 WebGL

Como WebGL é baseada na versão otimizada para dispositivos móveis do OpenGL não é possível utilizar muitos recursos especiais disponível para os ambientes desktop.

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). Não obstante a dificuldade de utilização ainda persiste, forçando a maioria dos desenvolvedores a utilizarem abstrações criadas por bibliotecas de terceiros.

É importante 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. Existe também uma lista de placas gráficas com drivers bloqueados por não funcionarem corretamente no Firefox (Matti e Arto 2011, pp.42).

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)

CocoonJS é uma aplicativo híbrido que preenche a fraca implementação de WebGL nos dispositivos móveis possibilitando se desenvolver em WEBGL, CSS. Conta com suporte a dispositivos legados à partir do Android 2.3 e IPhone 5.

7.6 Codecs

7.6.1 ÁUDIO

Segundo Kuryanovich et al. 2014 a limitação do elemento de áudio do HTML5 é que seu propósito é para executar apenas um som, como o som de fundo dentro de um jogo.

Durante a concepção do protótipo foi utilizado o elemento *audio* para emitir sons na utilização dos botões de resposta do aplicativo. Não obstante, sons consecutivos tiveram problemas no Android.

Depois de migrado para a API de áudio os problemas foram minimizados , mesmo assim, algumas vezes o som não é executado ou demora até executar de modo de o que é executado ao mesmo tempo que o próximo som.

A experimentação do protótipo relativo a API de áudio confirmou as afirmações de Kuryanovich et al. 2014 a API de som é boa se você deseja apenas tocar alguma música, mas se você está lançando eventos em um jogo ela é problemática.

Os navegadores variam na disponibilização de formatos aceitáveis Somente um áudio pode ser tocado no Navegador do Android

Não é possível trocar o volume no IOS.

Alguns navegadores favorecem formatos OGG (vorbis) e outros, como o Safari, favorecem o MP3.

Segundo «Smart Phones Application development using HTML5 and related technologies: A tradeoff between cost and quality»

> 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.

7.6.2 VIDEO

O suporte a vídeo, apesar de estar melhorando, ainda é rudimentar. No Silverlight existem uma coleção de possibilidades como aplicar shaders diretamente no vídeo e streaming suave de 1080px HD (Huang 2011, pp. 8).

O projeto *Vídeo for Everybody* `http://camendesign.com/code/video_for_everybody` é um polyfill que recorre à flash quando o vídeo não é suportado pelo navegador.

Não é possível controlar a entrada em tela cheia através de script pois é considerado uma violação de segurança. Não obstante, os navegadores tem a opção de deixar os usuários escolherem ver vídeos em tela cheia através de controles adicionais (Lubbers, Brian e Frank 2010, pp. 68).

As tecnologias de mídia são restringidas pelo política de mesma origem (Lubbers, Brian e Frank 2010, pp. 68).

7.7 Armazenamento

IndexedDB é uma API interessante mas para projetos menores é um tanto complexa.

7.8 OFFLINE

Nos arquivos de cache via manifestos, quando o download falha, o navegador emite um evento mas não há indicação de qual problema aconteceu. 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.

Refresh duplo para ver assets cacheados. Ver: http://buildnewgames.com/game-asset-management/

7.9 Orientação

Visto que a especificação o suporte está longe de completo, até o momento da pesquisa o único navegador popular que implementa a especificação completamente é o Edge da Microsoft.

Outrossim, existem diferenças substanciais entre navegadores, por exemplo: O Firefox e Google Chrome não manipulam angulos da mesma forma, por conseguinte alguns eixos se comportam de maneiras opostas (*Detecting device orientation*). 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.

7.10 Detecção de recursos

Grande parte da detecção de funcionalidades é feita através de JavaScript, isso força os desenvolvedores a criarem pelo menos parte da marcação em JavaScript, isso pode ser um fator limitante para o uso generalizado de HTML5 (Pilgrim 2010).

7.11 Debug

Com o depurador do Google Chrome remoto foi observado uma falta de sincronia na atualização da imagem no computador. Os efeitos do CSS não são bem apresentados e algumas vezes a tela foi completamente alterada (como na mudança de páginas) e no computador o estado anterior é apresentado.

O inspetor de canvas nativo do Google Chrome foi removido pois continha comportamentos indesejáveis para os desenvolvedores do navegador (*HTML5 canvas inspector?*). Os desenvol-

vedores 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 a tecnologia não apareceu no navegador (*Introducing the Canvas Debugger in Firefox Developer Tools*).

Atualmente a área de ferramentas para depuração do canvas 2D estão comprometidas por não existir um plugin de terceiro tão bem consolidado como o WebGl Inspector. O projeto Canvas Interceptor é uma iniciativa para preencher esta demanda permitindo capturar alguns eventos do contexto 2d e tirar *snapshoots* 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 nativo no navegador ou via extensão. [3]

7.12 Entrada de comandos

(Powell e Li 2013, pp. 9) cita que: Fazer scroll juntamente com gestos de ações similares é uma área onde o HTML ainda é fraco. Para contornar este problema pode-se utilizar bibliotecas como o iScroll, TouchScroll, GloveBox, Sencha, jQuery Mobile, entre outros.

7.12.1 Gamepad

Chrome does things differently here. Instead of constantly storing the gamepad's latest state in a variable it only stores a snapshot, so to do the same thing in Chrome you have to keep polling it and then only use the Gamepad object in code when it is available.

7.13 Disponibilização

O serviço PhoneGap Build não contém a última versão do PhoneGap e não é possível utilizar plugins através deles. Embora no protótipo isso não tenha sido um problema, em projetos maiores estes tipos de requerimentos podem ser vitais.

7.14 Recursos

Apesar da grande maioria dos recursos oferecidos nativamente nos dispositivos estar presente em HTML5 ainda existem algumas funcionalidades faltando para este tipo de aplicação.

«Smart Phones Application development using HTML5 and related technologies: A trade-off between cost and quality» cita algumas limitações no contexto geral

[3] Mais informações sobre o projeto Canvas Interceptor podem ser encontradas no seguinte endereço https://github.com/Rob–W/canvas-interceptor

> 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.

Vahatupa 2014 apresenta as seguintes limitações no contexto de jogos de navegador.

- Não podem ser instalados em um dispositivo móvel como um aplicativo separado
- Não tem acesso a funcionalidades específicas dos dispositivos e recursos como notificações, use de hardware nativo ou comunicação entre aplicativos.

7.15 Monetização

É muitas vezes difícil encontra oportunidades financeiras em jogos para navegador, visto que não existem pacotes de jogos para vender (Vanhatupa 2010, pp. 44).

Dado este 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, pp. 44). Outra saída é a venda de recursos extra, Vanhatupa 2010, pp. 44 afirma que a venda de moeda virtual é uma forma comum monetização.

Não obstante jogos WEB podem ser integrados pelos desenvolvedores a sistemas de pagamento - possivelmente aumentado a complexidade da aplicação. Ou serem integrados a algum mercado como o do Android, se optarem pela arquitetura híbrida.

7.16 VERSÕES

A grande maioria dos dispositivos atualmente no mercado utilizam obsoletas de seus softwares.

Além de obsoletas, as implementações podem conter erros.

Segundo Barnett 2014

> 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, todavia essa solução não é prática.

O Crosswalk é uma tecnologia que pode resolver parcialmente o problema de versões de software antigas para Android. Visto que a aplicação é disponibilizada juntamente com uma versão recente do navegador que a irá rodá-la. Não obstante, o Crosswalk só é suportado para Android [4].

[4] Nos apêndices o Crosswalk é tratado com mais detalhes

A tabela abaixo demonstra é um comparativo das tecnologias pesquisadas, relevantes aos jogos e seu suporte. As informações de suporte foram extraídas do site Can I Use e os navegadores populares em questão são: Internet Explorer (11), Edge (13), Firefox (43), Google Chorme (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
Gamepad	Não	Sim
WebGL	Parcial	Sim
WebSockets	Sim	Sim
Video	Sim	Sim
Audio	Sim	Sim

7.17 PERFORMANCE

Otimizações de performance dependem do ambiente em que estão sendo feitas. E aquelas que hoje tem um impacto positivo hoje podem se tornar inúteis, ou mesmo prejudiciais, amanhã (Kuryanovich et al. 2014, pp.131).

Com o HTTP/2 minificar arquivos pode não ser mais benéfico, visto que todos passam pelo mesmo canal de comunicação.

7.17.1 ASSETS

Trafegar muitos arquivos deixa o sistema lento. Pode-se contornar este problema utilizando páginas de carregamento e/ou cache.

Outro problema relativo aos assets é descrito por Garisel e Irish 2011 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.

8 CONCLUSÕES

Neste trabalho revisamos tecnologias relevantes no desenvolvimento de jogos e algumas das limitações a elas relacionadas.

Apesar dos problemas mencionados neste trabalho, tomando uma perspectiva mais abrangente as tecnologias de desenvolvimento de jogos evoluiu muito e existem muitas coisas boas para se dizer à respeito.

É seguro afirmar que ainda não podemos afirmar que as tecnologias da WEB podem substituir as nativas na criação de jogos. Não obstante as tecnolgias da WEB podem ser aplicadas ao desenvlvimento de jogos em uma vasta gama de casos.

A manipulação de erros é bastante consistente nos navegadores mas incrivelmente não faz parte da especificação (Garisel e Irish 2011). Não pude testar todos os métodos e ferramentas e versões à disposição, um trabalho completo demandaria esforços conjuntos de muitos indivíduos ou um período de tempo bem mais extenso.

Existem funcionalidades interessantes para jogos que não funcionam em todos os navegadores. Por exemplos a possibilidade de criar um ícno da aplicação WEB através do Safari.

Se uma empresa deseja produzir jogos nativos elas precisarão de vários desenvolvedores. Eu sozinho fui capaz de produzir um jogo em tempo razoável trabalhando com a plataforma WEB.

Por não utilizar frameworks e bibliotecas estou me distanciando dos casos da vida real.

Só poderemos considerar o HTML como uma especificação pronta quando for possível fazer tudo o que se faz nativamente com os dispositivos através de uma API WEB padronizada.

Muitas das limitações do HTML5 são contornáveis através de JavaScript.

O futuro dos jogos em HTML5 parece brilhante.

Este trabalho surgiu da minha vontade de aprender mais sobre a WEB....

O suporte ao HTML vem crescendo com o tempo, o site HTML5Test `http://html5test.com/about.html`, oferece um placar atualizado dinamicamente, conforme utilização dos navegadores, sobre os recursos do HTML.

Apesar de ser difífil de prover a mesma experiência em duas plataformas diferentes, é possível utilizar algumas plataformas com funcionalidades limitadas enquanto outras recebem a expeirência completa dos jogos (Bruins 2014, pp. 1).

8.0.1 TRABALHOS FUTUROS

Trabalhos que explorem os benefícios mercadológicos do HTML5 em comparação com alternativas nativas. EMACSCRIPT 7.

HTTP2 também traz boas perspectivas em relação a performance.

Um trabalho com criação de protótipos em WebGL seria interessante.

A utilização de plugins foi interacionalmente ignorada trabalhos que analisem a construção de um jogo comercial utilizando plugins, da forma mais aproximada possível da realidade mercadológica seria igualmente interessante.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Joshua Barnett. «Building a Cross-Platform Mobile Game with HTML5». Em: *University of East Anglia* (2014).

[2] John Bristowe. *jQuery UI vs Kendo UI*.

[3] Stefan Bruins. «The current state of cross-platform game development for different device types». Em: (2014).

[4] Chris Cevelleja. *A Simple Guide to Getting Started With Grunt*.

[5] *Chrome OS vs. Android: What's the difference?*

[6] Matteo Ciman, Ombretta Gaggi e Nicola Gonzo. «Cross-Platform Mobile Development: A Study on Apps with Animations». Em: *University of Padua* ().

[7] *Detecting device orientation*.

[8] *DeviceOrientation Event Specification*.

[9] *DOM Events*.

[10] Regina Dorokhova e Nickolay Amelichev. «Comparison of Modern Mobile Platforms from the Developer Standpoint». Em: *8TH CONFERENCE OF FINNISH-RUSSIAN UNIVERSITY COOPERATION IN TELECOMMUNICATIONS* ().

[11] *Drawing shapes with canvas*.

[12] Jon Duckett. *HTML and CSS - Design and Build Websites*, pp. 1–514.

[13] *ECMAScript 2015 Language Specification*.

[14] *Gamepad*.

[15] Tali Garisel e Paul Irish. «How Browsers Work: Behind the scenes of modern web browsers». Em: (2011).

[16] Isabela Granic, Adam Lobel e Rutger C. M. E. Engels. «The Benefits of Playing Video Games». Em: *Radbound University Nijmegen* (2014).

[17] Yousuf Hasan et al. «Smart Phones Application development using HTML5 and related technologies: A tradeoff between cost and quality». Em: ().

[18] Ian Hickson. *HTML is the new HTML5*. Available at `https://blog.whatwg.org/html-is-the-new-HTML5` (2015/11/11).

[19] *HTML HyperText Markup Language*.

[20] *HTML5 canvas inspector?*

[21] Jun Steed Huang. «Research on Html5 Development». Em: *Jiangsu University* (2011).

[22] *Introducing the Canvas Debugger in Firefox Developer Tools*.

[23] *Introduction to game development for the Web*.

[24] Marek Janiszewswi. Tese de doutoramento. Vrije Universiteit Amsterdam, 2014.

[25] Egor Kuryanovich et al. *HTML5 Games Most Wanted: Build the Best HTML5 Games*. 2014.

[26] David de Oliveira Lemes. Tese de doutoramento. Pontifícia Universidade Católica de São Paulo, 2009.

[27] Håkon Wium Lie. «Cascading Style Sheets». Tese de doutoramento.

[28] Petter Lubbers, Albers Brian e Salim Frank. *Pro HTML5 Programming - Powerful APIs for Rich Internet Application Development*. 2010.

[29] Anttonen Matti e Salminen Arto. «Building 3D WebGL Applications». Em: (2011).

[30] Ciarán McCann. «HTML5 Research Journal». Em: ().

[31] Belchin Moises. *ECMAScript 6 Today*.

[32] Joshua Morony. «Viability of developing cross-platform mobile business applications using HTML5 Mobile Framework». 2013.

[33] Brand Neuberg. *What Is the Open Web and Why Is It Important?*

[34] Andreas Neumann e M. Andréas Winter. «Time for SVG - Towards High Quality Interactive Web - Maps». Em: *Institute of Cartography, ETH Zurich* ().

[35] Mark Pilgrim. *Dive into HTML5*. 2010.

[36] Joonas Pirttiaho. «Cross-platform mobile game development». Em: *Saimaa University of Applied Sciences* (2014).

[37] Mark Powell e Yuesong Li. «HTML5 - A Serious Contender to Native App Development or Not?» Em: (2013).

[38] Ely Fernando Prado. «Introdução ao desenvolvimento de Games com GWT e HTML5». Em: *Departamento de Computação, Univerisidade de São Carlos* (2012).

[39] Axel Rauschmayer. *A first look at what might be in ECMAScript 7 and 8*.

[40] Jake Rocheleau. *Introductory Guide to Building Your First HTML5 Game*.

[41] Chris Rogers. *Web Audio API*.

[42] Jacob Seidelin. *HTML5 Games - Creating fun with HTML5, CSS3, and WebGL*. 2010.

[43] *Selectors*. Available at `http://www.w3.org/TR/CSS21/selector.html` (2015/11/17).

[44] *The Step-by-Step Guide to Publishing a HTML5 Mobile Application on App Stores*.

[45] *Using CSS transitions*.

[46] Juha-Matti Vahatupa. «On the development of Browser Games - Current Technologies and the Future». Em: (2014).

[47] Juha-Matti Vanhatupa. «Browser Games for Online Communities». Em: *International Journal of Wireless and Mobile Networks* (2010).

[48] Yu Zhang. «Developing Effect of HTML5 Technology in Web Game». Em: *International Journal on Computational Sciences and Applications (IJCSA)* (2012).

A Diagrama de classes

A figura A.1 é o diagrama de classes detalhado do protótipo.

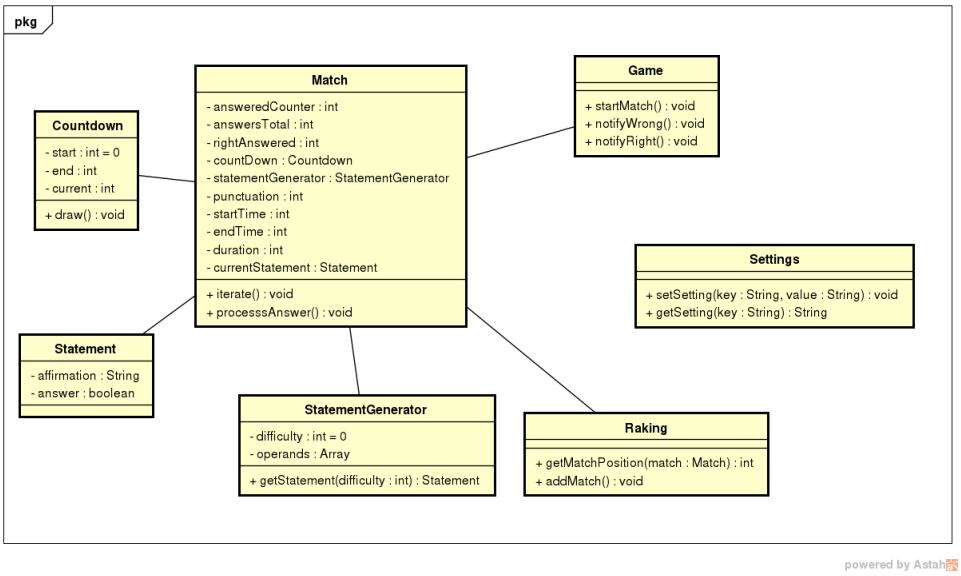

Fig. A.1: Diagrama de classes completo

B Bibliotecas relevantes no desenvolvimento de jogos WEB

A quantidade de ferramentas WEB á disposição dos usuário é enorme. Segundo (Kuryanovich et al. 2014) desenvolvedores da WEB são geralmente de mente aberta e desenvolveram uma variedade de bibliotecas e frameworks pela internet.

Sabendo disso, é uma tarefa praticamente impossível ter habilidade em todas as ferramentas disponíveis. Entretanto, é importante que os desenvolvedores ao menos conheça as ferramentas mais importantes disponíveis para que no momento necessário saibam qual aprender.

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 que se vejam livres de terem 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 CSS

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.

C CROSSWALK

Crosswalk empacota os fontes juntamente com uma versão do Chromium, a versão Open-source do Google Chrome. Isso faz com que o software se comporte da mesma forma para todas as versões de dispositivos Android.

C.1 Motores de jogos

Motores de jogos são bibliotecas que agregam várias funcionalidades usualmente úteis para o desenvolvimento de jogos (Vahatupa 2014, pp. 5). Elas podem incluir controle de usuário, cena, áudio, física, etc. Também servem como uma camada de segurança adicional (Vahatupa 2014).

Outrossim, motores de jogos não são amplamente difundidos no mercado de jogos de navegadores (Vahatupa 2014). Com o intuito de simplificar o processo para os desenvolvedores, auxiliando-os a focarem-se apenas nas soluções que estão desenvolvendo, foram criados os frameworks para desenvolvimento de jogos.

Alguns frameworks reconhecidos serão descritos abaixo:

Enchant.js, dentre suas funcionalidades constam: orientação à, orientado à eventos, contém um motor de animação, suporta WebGL e Canvas, etc three.js: considerada leve, renderiza, WebGL e Canvas, arquitetura procedural limeJs: bom para 2d quintus: especialista em jogos de plataforma 2D

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 conceitos mais primitivos.

C.2 Frameworks Multiplataforma

C.2.1 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 («Cross-Platform Mobile Development: A Study on Apps with Animations», pp. 2). A tecnologia também conta com uma IDE especializada para o desenvolvimento em Titanium, enquadrando-se também em um ambiente desenvolvimento de jogos.

C.2.2 jQuery Mobile

jQuery Mobile é um dos mais famosos frameworks mobile da WEB, isso se dá, parcialmente, pela popuplaridade od jQuery em si (Morony 2013, pp. 14).

(«Cross-Platform Mobile Development: A Study on Apps with Animations», pp. 2) cita que:

> j0uery prove uma grande game 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, apesar de funcionar bem com desktops,

jQuery Mobile não é um framework para todos as necessidades mobile. Focando-se principalmente na interface; acesso ao hardware, instalação nativas 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, pp. 14).

C.2.3 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 (*jQuery UI vs Kendo UI*).

C.2.4 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 («Cross-Platform Mobile Development: A Study on Apps with Animations», pp. 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, pp. 15). O framework geralmente é combinado com outros para obter-se este tipo de funcionalidade.

PhoneGap também conta com um serviço para empacotamento online o PhoneGap Build. Este serviço possibilita que se carregue de um arquivo compactado contendo os fontes em um

padrão especificado 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.

C.2.5 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, pp. 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.2.6 IONIC

D Ambientes de desenvolvimento de jogos em HTML

Construct 2 é um editor na nuvem focado para usuários sem conhecimento prévio em programação orientado a comportamento;

PlayCanvas é uma plataformas 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;

O 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* (Intel XDK) 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, ou seja, o desenvolvimento também é multiplataforma e devido ao fato de os binários ficarem hospedados na nuvem, possibilitou a Intel criar compiladores para cada uma das plataformas disponibilizadas pelo PhoneGap.

O site `https://codepen.io` e `https://jsfiddle.net` permitem a construção e visualização de aplicações utilizando as tecnologias da WEB. Apesar de ser possível, criação de aplicações utilizando estas tecnologias não é muito comum, servindo principalmente para visualizar trechos de códigos de exemplos na WEB.

E Tecnologias de Compilação multiplataforma

Segundo **gtw**

> O GWT é um framework essencialmente para o lado do cliente quee 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.

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, pp. 8).

A linguagem do Libgdx é Java, e é possível compilar para Android, IOS, HTML5, entre outros. Visto que a linguagem do Libgdx é Java também é possível desenvolver em todos os desktops comuns.

F ALTERNATIVAS AO JAVASCRIPT

Abaixo seguem algumas tecnologias que servem de alternativa ao JavaScript.

F.1 TYPESCRIPT

Conhecido como uma versão estendida do JavaScript que compila para JavaScript normal.

F.2 DART

Google. DartVM é uma máquina virtual que está embebido no Google Chrome. Significante melhorias em performance quando comparado ao JavaScript. Existe o dart2js que compila código em Dart para JavaScript.

G METODOLOGIAS DE CRIAÇÃO DE SOFTWARE PARA GAMES

Como o jogo é um software complexo demanda-se a utilização de metodologias de engenharia de software, dentre os processos de software mais conhecidos academicamente destacamos:

OpenUP: este é bem detalhado e de característica iterativa e incremental. Gerando assim, um levantamento mais apurado dos riscos, requisitos e outros detalhes do sistema e a criação incremental do sistema, com requisitos maleáveis;

Cascata: processo antigo, caracteriza-se por ser pouco maleável aos requisitos mapeados posteriormente ao processo de análise;

Processo ágil - SCRUM: sua utilização é flexível e sendo um método ágil especifica pouca documentação, ou como dizem, somente a documentação necessária, este processo é bem conhecido e aceito na comunidade de desenvolvimento de software. Suas principais características são: divisão do processo de desenvolvimento através uma série de iterações chamadas sprints. Cada sprint consiste tipicamente em duas a quatro semanas. É bem aplicado a projetos que mudam constantemente e que demandam rápidas adaptações;

Processo ágil – XP: tem muitas características similares ao SCRUM por este também ser um processo ágil. Dentre suas especifidades destaca-se: versões frequentes, pequenos ciclos de desenvolvimento que buscam aumentar a produtividade, introduzem checkpoints onde os clientes podem agregar novas funcionalidades;

H SISTEMAS DE BUILDING

Segundo *A Simple Guide to Getting Started With Grunt*

> 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 unitário, aplicar *linters* nos arquivos para checar erros, compilar os pré processadores de CSS (LESS, SASS), e muito mais.

O Grunt é um automatizador destas tarefas quotidianas. Para realizar a automação o Grunt conta com um grande ecossistema de plugins que podem ser compostos para realizar as 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.

Para tanto 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;

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