Joshua Barnett

Registration number 5939968

Building a Cross-Platform Mobile Game with HTML5

Supervised by Professor Andy Day

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

Creating cross-platform games is a necessity for aspiring developers. The more platforms a game supports, the wider an audience the developer can reach, and therefore their potential for success. Nowhere is this more evident than the mobile gaming market. However, the platform fragmentation associated with mobile devices impedes the development process, as tailoring an application to each different device is impractical, and in most cases extraneous. One attractive solution to this impediment is targeting the web platform that constitutes openness and inherent portability. This project aims to explore the web platform's pros and cons by researching the key technologies available for developer use. These technologies will then be assessed through the creation of a mobile game by rating the ease of development and deployment when compared to conventional native development.

Full assessment of the field would require an analysis of both native and non-native development, but the scope of this project does not allow me to do both. Therefore, the focus of this project will revolve around HTML5 and the web platform. However, while HTML5 will indeed constitute the focus of my analysis - this is what I have used to build my prototype mobile game - I will refer to native development from my personal observations and apparent evidences as a means to provide field for comparison and give depth to this project. During the early years of my education, I taught myself how to draw and animate in Adobe Flash through watching online tutorials. I then learnt ActionScript and used this new skill to make games that were sold for sponsorship on various Flash gaming portals. However, since that time, the industry has been shifting away from using third-party proprietary plugins in favour of open technologies such as HTML5. I feel this project will encourage me to research and learn these emerging technologies, and the knowledge gained will provide me with a strong footing for my desired career in industry.

Acknowledgements

I would like to thank Professor Andy Day for supporting this project with and its fluctuating nature, and my parents Jane and Paul for cultivating my creative characteristics while enabling my technical skills. As well as Elise Lanteri for her superior grasp of english grammar and the written word.

Contents

List of Figures

1 Introduction

During the past five years the popularity of mobile devices has been increasing. These devices have become one of the foremost ways in which the modern world consumes media and entertainment. Original equipment manufacturers (OEMs) such as Apple, LG, Nokia, and Samsung, have become progressively competitive in producing better mobile devices for the global market, generating a variety of them in the process. This growing market constitutes a large audience for mobile applications which conscientious developers aspire to target. However, in order for a mobile application to succeed in such a competitive and saturated market, it is a necessity that they support as many devices as possible; the more devices an application supports, the wider an audience it can reach. Now, currently, this is an arduous process, as each mobile device falls into a subset of operating systems and hardware specifications. Each operating system has a different application programming interface (API), which is made accessible through its corresponding software development kit (SDK). Hence, for a developer to target each operating system 'natively' the development of the application must vary, and be adapted. This native adaptation process is an inefficient approach to cross-compatible mobile game development. It requires more time, people, and, consequently, money. At the core of this project's research is the exploration of a solution to this deterrent that developers face.

The Open Web Platform (OWP) in essence is a collection of open (royalty-free) technologies that can be used for application development. The standards for these technologies are defined by the World Wide Web Consortium (W3C) to standardise and maximize a consensus amongst its members. Web applications built with these technologies benefit from the inherent portability of the Web, meaning they can be universally executed and distributed amongst a wide range of devices. Modern mobile devices fall into a subset of this range, as they contain essential software such as web browsers that can render and run web pages used to deliver the OWP. This makes the OWP an appealing free alternative to that of developing natively for each mobile operating systems, as the requirements for cross-compatibility are minimized.

HyperText Markup Language (HTML) is the corner stone technology of the OWP. Its purpose is to specify the content of a web page. The fifth iteration of HTML (HTML5)

has further broadened the types of content that it can deliver, now allowing for the definition of rich interactive content such as audio, video, and a graphics within the page. HTML5 and its associated technologies can be leveraged by developers to create mobile games that support the variety mobile devices. This project will focus on the usage of HTML5 and its companion technologies in mobile game development. In order to carry out my analysis, I have decided to build a game using HTML5 so that I could, along the way, assess the ease of development and deployment, as well as the difficulties arising from it.

2 Background

2.1 Primer

HTML5, as defined by the World Wide Web Consoritum (W3C, 2014), is ' ... the 5th major revision of the core language of the World Wide Web'. However, the Web Hypertext Application Technology Working Group (WHATWG, 2014) have recognised that 'the term "HTML5" is widely used as a buzzword to refer to modern Web technologies'. This usage of "HTML5" as an umbrella term has been influenced by adoption and integration of existing widespread technologies into the W3C specification, encouraged by the design principles that underlie its development (Keith, 2010). In practical terms this means that when an emerging technology becomes widely supported by the web development community and the web browser vendors such as Google (Chrome), Apple (Safari), and Mozilla (Firefox), it becomes a candidate for adoption into the specification. Notable examples of this happening early in HTML's development are the inclusion of `<style>` and `<script>` tags which are commonly used to reference or embedded Cascading Style Sheets (CSS) and ECMAScript (commonly known as JavaScript (JS)). These technologies are now key components of modern web applications.[1]

Web browsers are client-side applications which let users view and navigate web

[1] "Linux" has similarly been distorted as an umbrella term when referring to Linux based operating systems such as Debian, Fedora, and Ubuntu. The literal use of "Linux" refers to the open-source kernel which lies at the heart of these distributions, but it alone does not represent a full operating system.

content. This content is typically hosted on web servers that communicate with the web browser through the Hypertext Transfer Protocol (HTTP). Once the browsers has received an initial HTML web page, it begins the process of parsing and rendering it. The parsing process also includes locating and requesting any additional resources that pertain to the web page such as multimedia content, styles, and scripts. Once all of the relevant resources have loaded and are present client-side, the browser can begin fully rendering the page and defined contents. This basic routine of web page acquisition also applies to HTML5 apps as they often revolve around a root web page that acts as an entry point similar to that of `Main` in languages such as Java and C++. Unfortunately, this delivery method has a few drawbacks when compared to that of native mobile apps: first, the user must have Internet connectivity to access the app, and they then have to launch a browser and navigate to the web address where the app is located. These shortcomings are arduous tasks for the average mobile consumer, as they are accustomed to acquiring apps through a centralised store, where apps are nicely categorized and presented making comparison and browsing easy for the user. Once they have selected and installed an app, it can be launched through an icon created on the operating system's GUI (Graphical User Interface). 'Offline & Storage' technologies introduced as part of HTML5 are now assisting developers in circumventing the foremost of these issues, by letting them cache their app's files through the HTML5 Application Cache, as well as storing persistent data through HTML5 Local Storage. Mobile OSs such as Android and iOS are also rolling out support for the creation of web shortcuts on the native GUI, allowing them to sit alongside native apps. These strives are rapidly putting HTML5 apps on equal footing with native app presentation. Nevertheless, the barrier for entry is still too high for a casual user who is oblivious to apps outside that of the conventional centralised app stores. Until there is more support for HTML5 apps in the main appstores, another common workaround for developers is to package the HTML5 app within a native app, as this can create a native web view through the operating system's API, which in turn can be used to render the embedded HTML5 app. This helps blur the line between HTML5 and native mobile apps, and delivers the same installation and launch experience that users are acquainted with. These kinds of apps are referred to as 'Hybrid' mobile apps, as they encompass advantages of both HTML5 and native apps

features. 'Hybrid' development also helps safe guard web developers from some of the cross compatibility issues that come from supporting multiple browsers, by bundling a selected web view or browser on which their HTML5 app is executed.

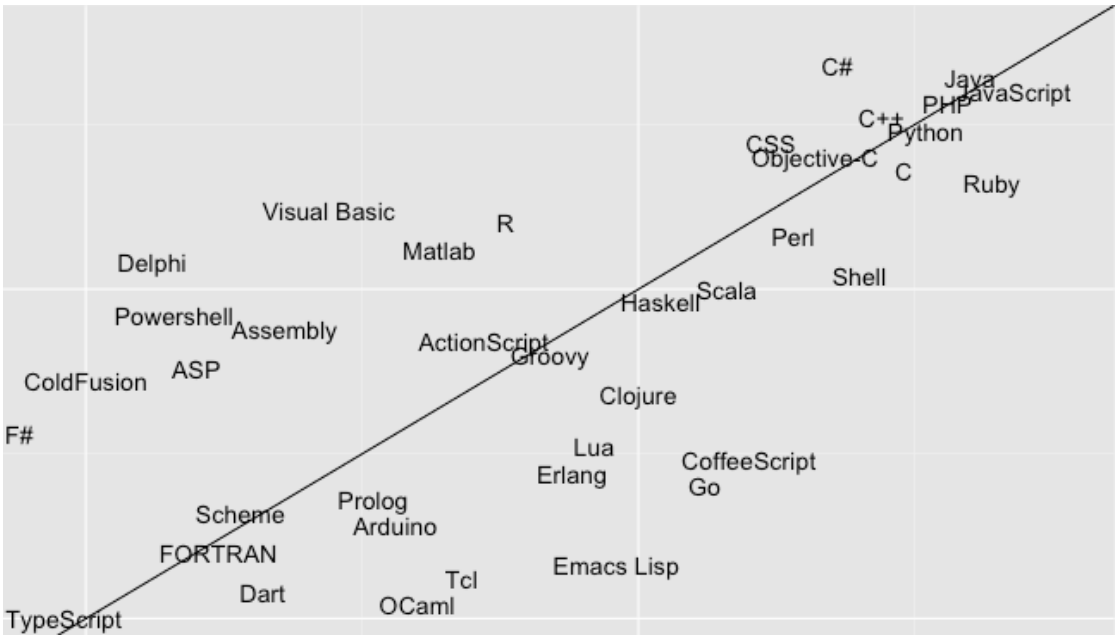
Getting a HTML app to render and function consistently across a range of different web browsers can be a huge undertaking. As noted in section 1, this task has grown exponentially with the introduction of mobile devices and their combinations of OS and web browsers. Each unique variation has the potential to produce bugs and quirks. While the HTML5 specification was being developed, much of the newly introduced features were implemented and experimented with at the discretion of the browser vendor. This has led to inconsistent support for these features across web browsers, as each vendor interprets the specifications differently while implementing them at varying rates. Developers must be aware of the idiosyncrasies between web browsers to ensure compatibility. Often, a developer will either limit their browser support scope and take full advantages of new features, or compromise and use polyfills, also known as a shims, to provide legacy support. A polyfill is often referred to as a code snippet that substitutes the lack of a future API by utilising existing features to mimic that of their final implementations Lawson and Sharp (2012). Nevertheless, even after polyfilling feature gaps, there will still likely be unforeseen bugs that will prompt further testing and debugging. As I allude to in my introduction, the most reliable way to ensure compatibility, is to physically test all browsers and mobile devices which the developer intends to support, but this solution is often impractical, especially with time, funds, and workforce limitations. The next best alternative is device emulation or investing in established cloud services that specialise in this method of testing.

2.2 Technologies

The technologies used during the development of the HTML5 mobile game were researched and discovered as they were required. Instead of recounting them during section 3, this section will be dedicated to explaining them independently, first to keep the report organised, and second, so they can be referred to when necessary, without extended explanation.

2.2.1 JavaScript

JavaScript is the most popular programming language to date (see figure 2.1). Every web browser today ships with a JavaScript interpreter, making it extremely portable and widely executable. JavaScript has been standardised under the alias ECMAScript (ECMA-262), the current version of which is 5.1[2]. However, the JavaScript pseudonym (a hangover from its forerunner) has become the preferred way to reference the language. JavaScript was originally created by Brendan Eich at Netscape as a way of making the newly added Java support in Netscape Navigator more accessible to non-Java programmers (Champeon, 2001). Since then it has grown to be the most widely supported language on the Web and is a core building block of modern web development.



x-axis: Popularity Rank on GitHub (by # of Projects)
y-axis: Popularity Rank on Stack Overflow (by # of Tags)
Source: *http://redmonk.com/sogrady/2014/01/22/language-rankings-1-14/*

Figure 2.1: The RedMonk Programming Language Rankings: January 2014

In its current state - ECMAScript 5.1 - JavaScript is an underwhelming language,

[2] *http://www.ecma-international.org/publications/standards/Ecma-262.htm*

especially when compared to more mature languages such as C++ or Java. Indeed, it lacks native support for object-orientated features such as inheritance, generics, and abstraction. As fore mentioned JavaScript was never intended for use in large-scale development projects, but due to its inherent portability, it has become widespread, and has had to bolt on improvements to keep up with trends. Until its apparent deficiencies are rectified and standardised in its forth coming iterations (ECMAScript 6/7), developers have to over utilise its strengths, such as being loosely typed, functions as objects, dynamic objects, and literals (Julian, 2009). There are many development tools such as transpilers designed to alleviate some of these gaps in feature support, as well as full blown alternatives such as Google's Dart. These technologies can iterate faster than that of heavily standardised languages such as JavaScript. The lengthy process of web language standardisation is a side effect of being open and having many parties contribute to the design and development of the language specification, this long cycle does help ensure a consensus and long term stability.

One big hindrance of utilising JavaScript in large scale development is the lack of a modular design pattern. Typically JavaScript is created as separate `.js` text files that are then referenced in a `.html` document by `<script>` tags. When the web browser finds scripts linked by these tags, it will request them from the web server and inject them into the document once loaded. This presents a few problems for large and complex web applications which often require many scripts, each having to be requested and loaded synchronously. Every one of these request has an overhead, which is compounded when a vast quantity are being made. This can lead to suboptimal load times.

To combat this problem a developer could attempt to write all their code in one `.js` file. This file can quickly become colossal, making it difficult to manage and refactor, especially on large scale projects. Alternatively, the developer can split their code into several `.js` files that will all have to be referenced by a corresponding `<script>` tag. Therefore, the more organised the codebase becomes, the more requests the browser has to make. The developer also has to micromanage these `<script>` tags to ensure their ordering is correct. The ordering is important because one script could be referencing another that has not yet been loaded, thus causing runtime errors. This approach also presents issues when attempting to introduce software engineering best practices such

as unit testing, as the interdependencies between these scripts are not clearly defined.

There are a variety of libraries that offer solutions such as RequireJS[3], HeadJS[4]and yepnope.js[5]. After testing and evaluating some of these libraries, Browserify[6]presented itself as the simplest and the most supported amongst the Node.js community. Browserify enables developers to write their JavaScript code in numerous files, and when one file requires another it can be injected via the `require('example');` function. Once a developer has written all their code, they can recursively bundle all their modules starting with a root module similar to that of a main class in conventional programming languages. This provides the developer with a good compromise as all their code will be built into a single `.js` file (including their external libraries), while allowing for easy management and refactoring of their core codebase through individual source files.

Most might think that this will in turn cause complications when debugging. As when the code breaks, it will do so in the colossal `.js` file that the browser is running. This is where source maps come in. Source maps provide linkage between the original `.js` source files and their bundled counterparts making breakpoints and stepping through code painless (Seddon, 2012). This process can also be made seamless by Watchify, which automatically recompiles the Browserify bundle whenever a module file has been updated. This allows for the developer to work on their original source files, and, as soon as the file has been saved it has rebuilt the bundle ready to be reloaded in the browser for testing. After the modules have been bundled, a further step, called 'minification', which involves taking legible JavaScript and compressing it down into its most minimal form by removing unnecessary characters such as a whitespace and shrinking instance names. This process can further decrease the `.js` file size making load times marginally faster while also partially obfuscating the original code from prying eyes.

Hanselman (2013) stated an intriguing opinion that states JavaScript has become akin to an assembly language for the web, which is the impression a lot of people get when taking a peek at the source of popular website such as Google or Facebook. This is somewhat understandable, given the that JavaScript is as 'low-level' as developers can

[6] *http:// requirejs.org/*
[6] *http:// headjs.com/*
[6] *http:// yepnopejs.com/*
[6] *http:// browserify.org/*

get when programming for the web. However, a fortunate side effect of being closely tide with the web is it has become now one of the most portable languages to date, because every web enabled device has a browser, and every browser has a JavaScript virtual machine. This makes JavaScript a very cross-platform language to develop games with which, since I have explained that reaching a wide audience is key when attempting to maximise chances for success, is a strong advantage.

2.2.2 Google Dart

Google's Dart "is a new platform for scalable web app engineering" which comes in the form of several tools. First there is Dart the language which is an alternative to writing plain JavaScript with a host of additional built-in and libraries language features akin to Java Fortuna and Cherem (2013). Second there is DartVM a virtual machine that comes as a standalone program and also happens to be embedded in the Google Chrome browser. The DartVM works similarly to a typical JavaScript virtual machine with the difference being it interprets `.dart` code instead of `.js` with significant performance advantages Schneider (2013). Lastly there is the dart2js compiler, which is effectively a backwards compatibility tool to ensure any application created on the Dart platform will still work through conventional means.

2.2.3 Microsoft TypeScript

Microsoft's TypeScript by its own admission "is a typed superset of JavaScript that compiles to plain JavaScript." which makes it very flexible alternative. This effectively means that you can continue to write plain JavaScript and no restrictions will be imposed by the TypeScript transpiler. This helps greatly if a developer already has a large JavaScript codebase which they wants to utilise during the development of a large scale application. TypeScript offers some of the inherently missing language features of JavaScript such as type annotations, classes, interfaces and modulesṪhere are also plans to support ECMAScript 6 features once it becomes the standard. Another positive remark about the output produced by the TypeScript transpiler is it produces rather legible JavaScript making it easy to see the relationships between the source and target languages.

Because the compile target of these transpilers is JavaScript it is possible to take advantage of them when developing mobile web applications such as video games. When project development first started, TypeScript was the target language because it was syntaxitcally similar to that of ActionScript 3, where my background lies. However, due to the immaturity of the project at the time and a lack of core JavaScript knowledge on my part, it was later ditched as time constraints prevented the investment of fully learning its quirks. Also it felt like working at this level of abstraction could potentially lead to delays when debugging later in development. However, as of *April 2014* close to the time of writing, the specification for TypeScript 1.0[7]has been finalised and its current status is *stable*. Going forward after this project I am eager to re-evaluate it now that it has had time to mature.

2.2.4 The Document Object Model (DOM)

'The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page.' (W3C, 2005). A web browser will create a DOM when parsing the elements and tags found in static HTML documents. This allows the browser to easily interpret relations between the content in the web page and use this information to render it appropriately. While in memory, these page elements, now represented as objects, are no longer static, which allows technologies such as JavaScript to manipulate and interact with them. Many web applications rely on the DOM's JavaScript API in order to deliver rich interactive user experiences. For instance Gmail which is a SPA (single-page application) that makes heavy use of this API. SPAs have significant advantages over that of traditional web pages as they can provide seamless user experiences akin to that of desktop applications. Instead of navigating between web pages through links that each request separate web pages, they manipulate the initial web page and load its required content and data in the background, displaying it when ready by weaving it into the current page context. This technique can help shrink load times by only requesting necessary data and reusing

[7] *http:// www.typescriptlang.org/ Content/ TypeScript%20Language%20Specification.pdf*

existing visual elements (Takada, 2012).

2.2.5 HTML5 Canvas

Canvas element is probably the most significant feature that was introduced in HTML5, especially for game developers. It finally enables developers to do interactive and dynamic graphics natively in the browser, without third-party plugins.

The new `<canvas>` element defines a graphics layer in a HTML document that can be manipulated and drawn to through a JavaScript API. Although this has been used in the context of desktop web applications to great effect it has suffered from performance issues on mobile browsers. This is often due to being limited to software rendering that only makes use of the device's CPU. However, the performance of mobile devices has been increasing in recent years, with an emphasis on GPU development Lin (2014). This has led to browser vendors adding support for GPU accelerated rendering on mobile devices in places that use complex effects such as CSS3.

Utilisation of these faster processors with the canvas element has only just started to become common place. There are many ongoing software projects with a focus on accelerating the HTML5 canvas using the mobile device's GPU. Unfortunately there is a lack of cross platform open-source projects that tackle this issue. Ejecta is an is one such project but it current only supports iOS and the hybrid mobile application framework Apache Cordova[8]also provides an accelerated canvas through a plugin called FastCanvas[9].

2.2.6 WebGL (Web Graphics Library)

WebGL is a distillation of the popular OpenGL, which makes GPU accelerated rendering in the browser a reality. WebGL, a JavaScript API, is based on the OpenGL ES (Embedded Systems) subset, and, as the name implies, it was designed and optimized for embedded systems such as mobile devices. One streamlined modification OpenGL ES made was the removal of the fixed-function API introduced in OpenGL 1.0, enabling the use and compilation of modern shaders.

[9] *https:// cordova.apache.org/*

[9] *http:// plugreg.com/ plugin/ phonegap/ phonegap-plugin-fast-canvas*

WebGL has been in development for the past three years and came to maturity as of early last year, when the WebGL version 1.0.2[10]specification was published, which 'clarifies interactions with the encompassing HTML5 platform' (Verry, 2013). The use of WebGL in application development, one notable application of the technology is in a new game console; PlayStation 4's user interface. In a revealing presentation given by Olmstead (2014) it was explained that they altered their stack to include WebGL with the intention of future-proofing it via cross platform support. This possibly hints at Sony's future plans to bring their games to multiple platforms including mobile devices through their *PlayStation Now* streaming service.[11]

Nevertheless, the support for WebGL on stock mobile browsers is currently non-existent, with partial support just starting to surface in browsers like Chrome and Firefox. One company striving to provide mobile developers with an attractive solution to this lapse in support is Ludei, have recently official rolled out a hybrid mobile application framework called 'CocoonJS'. CocoonJS lets developers package and publish HTML5 applications on mobile devices via their platform. One key feature of CocoonJS is the facilitation of WebGL, which is made possible through GPU acceleration; a feature accessible through the native APIs. Many of the demos that they provide demonstrate near native performance. This makes WebGL a feasible avenue for game development on mobile devices, with the added bonus of being cross platform ready. Once WebGL is implemented across stock mobile browsers in the near future, it is very likely that many developers well see this as a viable alternative to native mobile game development.

2.3 Node.js

'Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.'[12]Essentially, it enables developers to program applications in JavaScript that are executed outside of a web browsers similar to other

[11] *https://www.khronos.org/registry/webgl/specs/1.0/* - Accessed: 2014-05-08
[11] *http://us.playstation.com/playstationnow/* - Accessed: 2014-05-08

interpreted programming languages like Ruby and Python. Applications are written in the form of modules that rely on other modules. Core modules come bundled with the platform's binary and provide an API to common services such as HTTP and the File System. The NPM registry is where open-source and community created modules can be published alongside their dependencies. These self contained modules provide developers with even more utilities such as local development servers and build tools.

2.4 weinre

Weinre[13](web inspector remote) is a debugger application created by Patrick Mueller[14]who is also a contributor to the Apache Cordova project. This application works similar to the built-in desktop browser debuggers and inspectors with the added bonus of having a remote target. By including a remote JavaScript file hosted on a local weinre server the mobile device can be linked letting it send and receive debugger data from the server. This lets developers inspect and manipulate the mobile devices web browser. The key issue that prevented the further use of this application in the project's development was that in combination with Apache Cordova it was not able to log any information before the script was loaded and the console was initialized. So when it came to analysing the initial portion of an application's lifecycle the data generated was restricted by this dependency.

2.5 Hybrid Mobile Application Frameworks

Hybrid mobile apps are mobile web apps that run inside a native app container

3 Development

The development process of HTML5 applications and games like the platform they are built on, is in constant flux with new methods, techniques, and tools surfacing all the

[12] *http://nodejs.org/* Accessed: 2014-05-05
[14] *http://people.apache.org/~pmuellr/weinre/docs/latest/Home.html*
[14] *http://muellerware.org/*

time. Developers must ultimately choose a tooling stack that enables them to express their creativity, by letting them work productivity and efficiently. This undoubtedly will vary based on their personal preferences, background, and opinions on best practices. I have had no previous experience with HTML5 development, which encouraged me, during this project, to seek out the popular solutions to the problems I was encountering. This section of the report will detail the problems I faced, my found solutions, and what they let me accomplish.

3.1 Design

I have had much experience with game design and development, but because of my inexperience with the HTML5 platform, I felt that the core concept of the game had to remain simplistic and allow for expansion. This choice was meant to ensure that the project remained feasible, as my self-education would likely consume the majority of this project's allotted time. After initial deliberation, I decided to base the game upon the simple card game of *Concentration*[15] (also known as Pairs). The gameplay mechanics consist of the player initially being presented with several cards, each marked with a symbol linking it to a set. These symbols are only on the front-faces of the cards and are briefly visible to the player at the start of the game. When the cards are flipped the symbols are obscured, making each card indistinguishable from the rest. The player's objective is to identify the matching cards from memory. When designing a game it is important to be conscious of the player's perspective, as well as the flexibility it allows the developer. Hence, the following aspects of this game concept also made it an attractive candidate.

1. The mechanics are simple, making it rapidly accessible for players of all ages to recognise and understand the core gameplay.

2. It has depth, allowing for additional and more complex gameplay mechanics to be introduced gradually to the player.

[15] *http://en.wikipedia.org/wiki/Concentration_(game)*

3. It will translate well to touch screens, as the fundamental interaction the player will have is selecting cards.

4. It engages the player's brain in pattern recognition which is one of its inherent strengths.

Other mechanics such as restricting the player to matching sets of cards in a particular order were later introduced to further challenge the player and keep them interested with an increasing difficulty curve. This engages other strengths of the player's brain, as they will have to chunk their memories of card positions and prioritize them so they can be retrieved later in a particular order. Subtle changes in level design such as introducing additional shapes and symbols in various symmetrical and asymmetrical patterns were also added to push the player's cognitive abilities.

3.2 Implementation

To get a head start on the project implementation I started researching and acquiring tools that provided a familiar development experience to what I was accustom to. I felt doing so would make my transition to JavaScript programming much faster than learning it from scratch. Much of my game development experience is from several years of building games for Adobe Flash using its ActionScript programming language, so I began looking at some of the popular transpiler languages that translate to JavaScript once compiled. Microsoft's TypeScript presented itself as a close alternative to ActionScript, with familiar syntax and features such as classes, interfaces, and typed variables, staples of conventional high-level languages. I installed the compiler using NPM (Node Packaged Modules), which is accessible through a command-line interface that comes as part of the *Node.js* platform. After settling upon TypeScript as my intermediate language of choice I began testing various IDEs with TypeScript support. WebStorm a commercial JavaScript IDE featured, informative auto-completion, support for TypeScript with minimal configuration, and watchers (background processes that would compile TypeScript into JavaScript upon detecting changes). As a result of having my project supervisor fill out a request form, I was able to obtain an educational license

for this IDE that would last the length of a year. Once comfortable with the development workflow of programming in TypeScript and compilation via WebStorm, I began implementing and testing the basic game logic using a tile map and object models that represented each card. After game logic was functional I began researching JavaScript graphics libraries that could offer a similar hierarchical pattern that I was acquainted with in AS3. This hierarchy revolves around subclasses of `DisplayObject` and `DisplayObjectContainer` that can be manipulated and organised through methods such as `addChild`, `removeChild`, and `setChildIndex`. These functions enable the grouping and layering of multiple visual elements such as images in local and global contexts.

CreateJS, is a suite of modular JavaScript libraries developed by Grant Skinner, sponsored by Adobe. This suite contains a library called EaselJS, that emulates the aforementioned pattern, while also providing other useful APIs for asset loading, visual effects, masking, transforms, and mouse interaction. EaselJS can render this display list hierarchy setup via programming and renders its children to a HTML5 Canvas. During this period of research, I also came across the Apache Cordova project which also provided a command-line tool through *NPM*. Using EaselJS and Cordova I managed to develop and deploy a rough prototype of the game from placeholder art and the initial game logic I had already programmed in TypeScript. Unfortunately, working with the CreateJS suite in combination TypeScript proved to be more challenging than I initially thought. Moreover, the performance of HTML5 Canvas was slow and stuttering in Cordova, which further made me reconsider this approach. In order to fully utilise the features of TypeScript in combination with a typical JavaScript library, TypeScript definitions are required. TypeScript definitions outline a JavaScript library's API with type annotations in `.d.ts` files similarly to `.h` header files in C++. These `.d.ts` files can be referenced, which help the compiler map between TypeScript and JavaScript enabling features such as compile-time error detection and auto-complete in the IDE. There is an open source repository maintained by the TypeScript community on GitHub called DefinitelyTyped[16], which does contain definition files for the CreateJS libraries. However, upon further inspection I found the definition files were obsolete and incompatible with the latest version of CreateJS. The time it would have taken for me to

manually correct the definitions to get them to up-to-date would have been a lengthy process. I also felt that working at this level of abstraction above the true underlying JavaScript could potentially hinder my ability to debug later in development. This ultimately led to the abandonment of EaselJS and TypeScript in favour of migrating to JavaScript and DOM orientated approach.

While educating myself in JavaScript I found the lack of built-in structure daunting, and was worried it could plague my project's development as I lacked experience with the language. After further research I chose to employ the use of a JavaScript *MV** framework as it provided a structured pattern in which to develop a web application. I selected Backbone.js[17]because of its flexibility, large community following, and modular design. As described in a presentation given by Bull (2013) Backbone.js is the equivalent of a *knife/spoon/fork* on a camping trip. This analogy emphasises its modularity, implying that there is no imposed exclusivity between its internal components. This allowed me to gradually implement the use of its Model, View, Collection, and Router classes throughout this project's development, without being forced to learn everything at once but only when it was required. I began using Backbone, first, by porting my game logic code into appropriate *Model* classes and my EaselJS/HTML5 Canvas code to *View* classes that instead employed the use of the DOM.

Several improvements arose after the switch from rendering on Canvas to utilising the DOM. For instance, the original implementation of the card flip effect was achieved by simply tweening the width of the images to 0 and back. This technique produced a very flat effect. After switching from this Canvas implementation to using DOM elements, I was able to make use of CSS3 transforms. These transforms allow for 3D manipulation one area that the EaselJS library was lacking in. This allowed me to add perspective to the flip effect, which made it more aesthetically pleasing and helped highlight the user's interaction. The transition could be applied through adding and removing a CSS class selector to and from a particular card's DOM element. CSS3 transforms are hardware accelerated on many mobile devices, making the animation smoother than with the previous implementation. Also, because the browser handled process of rendering, it applied anti-aliasing by default making the edges of the images less jagged.

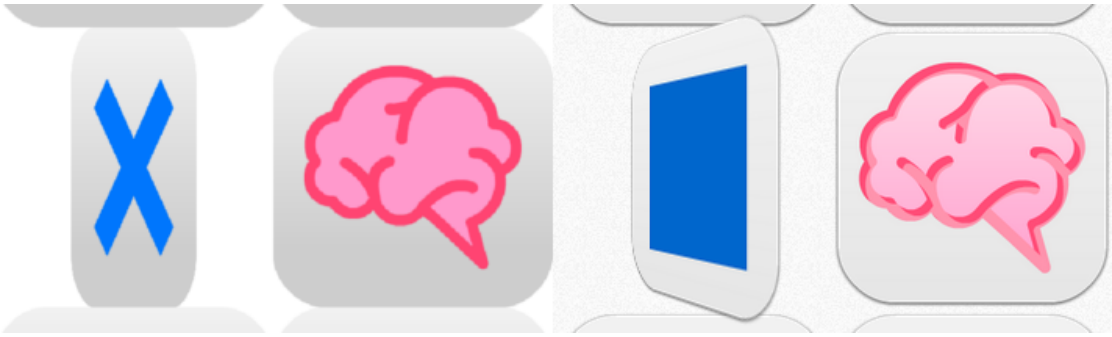


Figure 3.1: Comparison of original flip effect and current flip effect (left-to-right)

Scaling the game's visual elements to fit the varying screen sizes of my mobile devices proved more challenging in the DOM than it had been with the use of a Canvas element. The original scaling process consisted of setting the Canvas element's dimensions to match that of the window upon it firing a `resize` event, then updating the scale properties of the EaselJS `stage` instance appropriately. However, when working with the DOM, every element is rendered and positioned in the page's layout relative to its style attributes (such as `display`, `position`, and `float`), which means that almost all the rendering of the game's visuals could only be influenced indirectly. I first attempted to scale my game using *viewport units*, which lets you set elements style attributes relative to the current viewport's dimensions. For example, `width:50vh` would make an element's width 50% of the viewport's height. However, I found that the majority of stock mobile web browsers had not yet implemented this feature, with it only being present in versions 4.4 of Android and 6.0 of iOS (see Figure 6.3). This led me to seek an alternative solution that could give me a similar effect.

Remote debugging was a task I frequently had to perform in order to analyse problems that occurred while running on mobile. It was important to find a solution that was efficient and compatible with Cordova hybrid mobile applications. After searching for tools that fitted this criteria, I discovered the weinre project. I have explained in a section **??** how this software works in detail. I stuck with this solution for a large chunk of development because nothing better was available at the time, although launching the local weinre server, connecting my mobile devices, and then navigating to the inspector's interface through a browser on my desktop computer, was an arduous process.

= Supported = Not supported = Partially supported = Support unknown

iOS Safari	Opera Mini	Android Browser	Opera Mobile	Blackberry Browser	Chrome for Android	Firefox for Android	IE Mobile
		2.1					
		2.2					
3.2		2.3					
4.0-4.1		3.0	10.0				
4.2-4.3		4.0	11.5				
5.0-5.1		4.1	12.0				
6.0-6.1		4.2-4.3	12.1	7.0			
7.0	5.0-7.0	4.4	21.0	10.0	33.0	26.0	10.0

Source: *http://caniuse.com/#feat=viewport-units*

Figure 3.2: A compatibility table of support for viewport units

Later in development, I discovered that Google Chrome comes with built-in 'DevTools' that feature remote inspection of an Android Web View, providing it was using the latest OS (4.4 KitKat). This solution significantly improved my development process making it rapidly iterative. First I would debug the game in Chrome on my desktop computer. If all went well, I would then build, deploy, and launch the application on mobile through Cordova's command-line interface, which automatically connected to the familiar Chrome's DevTools I had just used, to test the desktop version. Chrome also later added mobile emulation to DevTools which facilitated the simulation of touch events, device orientation, and screen density/resolution. This was preferable to deploying the Cordova package everytime I wanted to quickly test mobile functions, as it loaded quicker than the time it took to build and deploy the package to mobile. However, I still tested on mobile at regular intervals to ensure overall compatibility.

As the project progressed, the addition of numerous tools and libraries meant that the structure and build processes grew overly complicated, which diminished my ability to implement efficiently. Many of these tools and libraries the project relied upon were NPMs in the Node.js ecosystem, which led me to the conclusion that I should also format my project as a NPM. NPMs are configured through their `package.json` file in the project's root directory. In this file, I was able to specify the dependencies of my project and define scripts that ran a variety of tasks such as launching the local development server or deploying the mobile game through Cordova.

As I added more levels to the game I decided to implement the locking of future

levels similar to that of most popular mobile games. This unlocking of levels acts as a reward system that the player can become invested in, encouraging them to come back. However, this came with the overhead of storing persistent data that would maintain their progress between play sessions. HTML5 Local Storage provided a quick cross-platform solution that allowed for the storing of key value pairs on the player's device without interacting with native APIs. I used this to store level data, that included a boolean value representing the locked/unlocked state, as well as all the scores obtained by the player. Once deployed on the centralised app stores I plan to sync this data with a player account so they can compare scores with others, adding a competitive element to their progression.

After reaching the end of implementation I had integrated several more project development tools. The final project structure relied upon using NPM to manage all the development tools such as Apache Cordova, Gulp.js, and Bower. The Bower package manager was used to organise all my front-end libraries used by the HTML5 application such as Backbone.js and jQuery. Then finally everything was tied together by Gulp.js a streaming build system that managed all my project tasks such as bundling the game modules and front-end libraries, minifying and obfuscation of the bundles, building the Cordova application from the HTML5 application source, then deploying the built application to mobile devices.

- Asyncronous Script Loading - CSS3 Flip Transform - Remote Debugging - Features Detection - Button Responsiveness - Sound Latency - Native Integration - Package Managment - Scaling Problem - Modules - Git Version Control + Overlay - Save Game Data - Touch Leave Shim - Gulp.js - WebGL

3.3 Feedback

Throughout the development of my game, I considered it essential to obtain feedback and reactions from players to gauge its performance first hand. Therefore, whenever willing participants were available, such as friends, family, and classmates, I presented them with a device containing the latest stable build, and observed their behaviour to get third-party perspective. To test the intuitiveness, I gave them no explanation or hint

[17] *https://github.com/borisyankov/DefinitelyTyped*

as to how the game should be played. Indeed, the process a player goes through in order to understand the mechanics is a vital component of player satisfaction. If they do not understand the game's rules almost instantaneously, many people will lose interest shortly after picking it up.

The first playable prototype received a lot of feedback from players relating to 'lag'. The 'lag' they were referring to was the latency between their interaction and visual/audible feedback. This gap detracts from the responsiveness and tactile impression required to inform the player. After some research I found an article written by Croft (2012) on creating responsive buttons with PhoneGap (Cordova), which encouraged me to test different JavaScript events. I found that `touchstart` and `touchend` events fired much faster than the `click` event, I was currently using. This is intentional because mobile browsers delay the firing of the `click` event, in order to differentiate it from the initial press which is represented by the `mousedown` and `touchstart` events. There are plans of unifying these two sets of events under the a pointer events specification, but until this is implemented by browser vendors developers have to deal with this fragmentation. I solved this problem for both platforms by employing a feature detection library that let me choose which events to listen for upon initialisation. Another optimization I made was to keep the style of the buttons to a minimum, as complex effects take longer to render which happens every time the button changes state. These modifications solved the latency of visual feedback. I tackled the audible latency by analysing the waveform of the sound effects that were sourced from *freesound*. I found many of the sound effects had a lot of trailing blank space at the beginning and ends of the audio files, so after trimming, fading, and normalizing all the effects, this decrease a significant portion of the latency. However, after further play tests it was apparent there was still a minor delay between the event firing and the audio playing. I thus decided to switch from HTML5 Audio to a Cordova's official Media plugin that gave me access to native audio capabilities. Web Audio API would have been a preferable alternative but most mobile browsers had yet to implement it. Unfortunately, the offical Media plugin still had quite a high latency. After searching I found a custom *LowLatencyAudio* plugin that had been developed by (Trice, 2013) for the PhoneGap framework (the predecessor of Cordova). Since its debut, the plugin has been ported to for Cordova by (Xie, 2013),

but only for Android and iOS, which finally brought the audible latency to a reasonable level.

When repeatedly testing, developers can become accustomed with the game, which leads them to underestimate the game's accessibility to new players. This often takes the form of unintentionally assuming prior knowledge on part of the player. In some cases, this can have a positive effect, allowing them to skip explanation of common tropes by drawing parallels to that of other popular games. A good example is the match-three game popularised by *Bejeweled* in the 2000s. After its debut, this same mechanic has surfaced again in the popular mobile game *Candy Crush Saga*, which has innovated, adding further complexity to a familiar concept (Juul, 2007). Leveraging existing tropes can act as a shortcut when teaching a player new or similar game systems. Nonetheless, these can also be stumbled upon accidentally from a developer's perspective, as their preconceptions will often differ vastly by comparison to the average player. This was the case of the introductory level in my initial draft. At the start, the first two symbols the player encountered were a red nought and a blue cross on a three-by-three grid. During the first play test of this level, I discovered the player's natural instinct was complete the level by producing a winning *Tic-tac-toe* game state. This was a glaring flaw in my game's design, because the symbols are common to a pre-existing game (namely *Tic-tac-toe*). The player receives mixed signals about how to play the game from the offset, leading to confusion and ultimately, frustration. I thus later decided to try a simpler approach, using more abstract symbols (such as circles, squares, and triangles) in my initial set of levels. These symbols are more generic and therefore do not have any strong associations that can mislead the player.

When repeatedly testing, developers can become accustomed with the game, which leads them to underestimate the game's accessibility to new players. This often takes the form of unintentionally assuming prior knowledge on part of the player. In some cases this can be used to their advantaging allowing them to skip explanation of common tropes by drawing parallels between their game's mechanics and that of other popular games. A good example is the match-three game popularised by *Bejeweled* in the 2000s, this same mechanic has surfaced again in the popular mobile game *Candy Crush Saga* who have innovated adding further complexity to a familiar concept (Juul, 2007).

Leveraging existing tropes can act as a shortcut when teaching a player new or similar game systems. However, these can also be stumbled upon accidentally from a developer's perspective as their preconceptions will often differ vastly by comparison to the average player. This was the case of the introductory level in my initial draft. At the start, the first two symbols the player encountered were a red nought and a blue cross on a three-by-three grid. During the first play test of this level, I discovered the player's natural instinct was complete the level by producing a winning *Tic-tac-toe* game state. This was a glaring flaw in my game's design, because the symbols are common to a pre-existing game (namely *Tic-tac-toe*), the player receives mixed signals about how to play the game from the offset, leading to confusion and ultimately frustration. I thus later decided to try a simpler approach, using more abstract symbols, such as circles, squares, and triangles in my initial set of levels. These symbols are more generic and therefore do not have any strong associations that can mislead the player.

4 Conclusion

After

5 Future

The key issue HTML5 is tackling on the mobile platform is portability. OEMs will continue to improve and innovate their product lines at different rates relative to what is profitable and marketable. As the cost of the technology required to construct these devices plummets so does the cost to improve and innovate. The technology life-cycle of mobile devices is short in contrast to that of web specifications. If developers desire cross platform compatibility there are many moving targets they have to hit when it comes to native development; hardware specifications, operating systems, store requirements, and official software development kits. Hitting these targets in an ever fluctuating landscape demands much of a developer's focus, which could be better spent more effectively improving a game's overall quality and design. The web has proved itself as a stable and universal platform on desktop computers and eventually this will

be implemented fully on mobile devices.

Currently the platform on mobiles is lacking full support of accelerated canvas and WebGL in stock mobile browsers. However, this does not have to discourage developers as the support is on its way. In the mean time developers can use hybrid mobile application frameworks such as Apache Cordova and CocoonJS to get a head start on utilising edge features. Once native support for these features is introduced their original source code will be instantaneously be compatible not even requiring compilation. As processing power on mobile devices increases other concerns such as performance will eventually become irrelevant and then a developers focus will skew towards the architecture and accessibility of their code rather its over engineered optimisation.

The future of JavaScript looks bright according to the creator Eich (2014). ECMAScript 7 intends to introduce language features such as overloadable operators, observers, value objects, and SIMD (single instruction, multiple data) intrinsics. The use of SIMD and low-level value objects in JavaScript will lead to near native performance while being widely compatible across processor architectures such as Intel's SSE, AMD's AVX, and ARM's NEON popularly used in mobile devices. He goes on to promote his extensible web manifesto[18]that notably focuses on adding safe and secure low-level capabilities to the web platform as well as simplifying and streamlining the standardisation process by tightening the feedback loop between committees and developers.

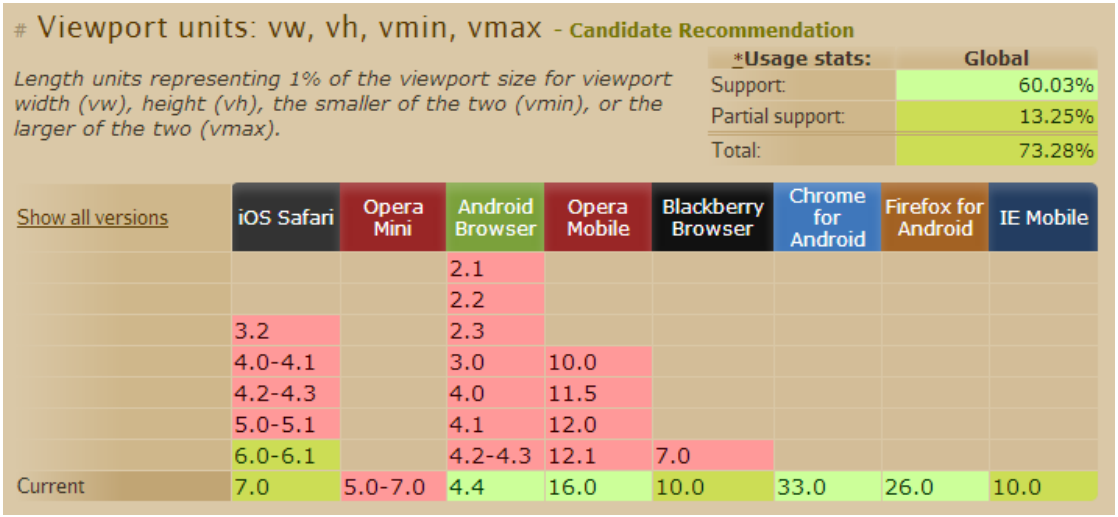
The video games industry will continue to grow in scope. Consoles such as the *Nintendo Wii* have had widespread success across demographics in recent years. Victories such as these are bringing this creative medium to mainstream society. I feel mobile devices will similarly perpetuate the medium as they become common place in our daily lives. With the web growing alongside these devices it is a natural progression that they become intertwined. This is already happening in projects such as Ubuntu for Phones and Firefox OS. The web platform presents attractive prospects for game developers offering lasting portability which as performance issues become less of a problem will become their dominant priority. This project overall has been successful and has encouraged me to seek out and research this platform thoroughly. I feel my newly acquired

[18] *http:// extensiblewebmanifesto.org/*

skills and experience will hugely useful going forward with my career in industry.

6 Old Stuff

Depending on the project specification a developer can rapidly become limited by what features of HTML5 they can utilise. For instance one such feature that was initially being used to develop *MindFlip* was *viewport units*. This feature allows for the scaling of elements based upon the current viewport's dimensions which is especially useful on mobile devices for scaling visual elements relative to the screen's aspect ratio and pixel density. However the stock web browsers on mobile devices have only recently added support for this feature with it being introduced in version 4.4 of Android and 6.0 of iOS.



Viewport units: vw, vh, vmin, vmax - Candidate Recommendation

Length units representing 1% of the viewport size for viewport width (vw), height (vh), the smaller of the two (vmin), or the larger of the two (vmax).

*Usage stats:	Global
Support:	60.03%
Partial support:	13.25%
Total:	73.28%

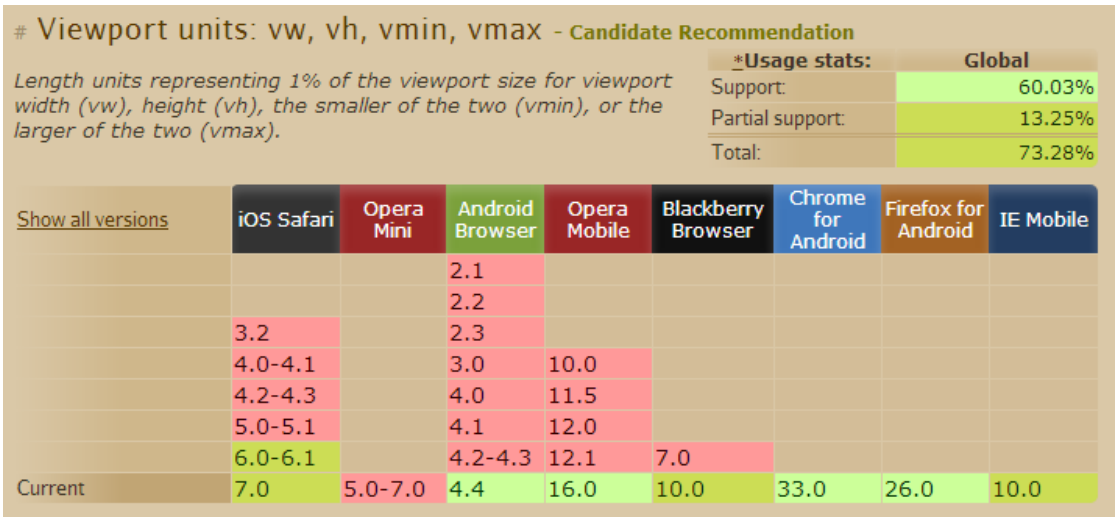
Show all versions	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Blackberry Browser	Chrome for Android	Firefox for Android	IE Mobile
			2.1					
			2.2					
	3.2		2.3					
	4.0-4.1		3.0	10.0				
	4.2-4.3		4.0	11.5				
	5.0-5.1		4.1	12.0				
	6.0-6.1		4.2-4.3	12.1	7.0			
Current	7.0	5.0-7.0	4.4	16.0	10.0	33.0	26.0	10.0

Figure 6.1: A compatibility table for viewport units

In cases like these a developer can either limit their platform target range and make use of the fringe features or use a polyfill also known as a shim. A polyfill is often referred to as code that substitutes the lack of a future API (application programming interface) by utilising existing features to mimic that of newer fully fledged implementations Lawson and Sharp (2012). In the case of this project the lack of *viewport units* support in older mobile browsers present an issue. After searching for a solution the

presentation slides of Kadrmas (2012) detailed a technique using *em* units in HTML5 mobile games. *em* units were originally intended for scaling DOM elements relative to the current font size. This technique was implemented in the project and allowed the game to correctly scale across differing screen sizes. When a window `resize` event is dispatched the `resize` function in `viewport-model.js` uses the window's current dimensions to set its values appropriately. These changes in the model are then listened for by `viewport-view.js` then applies them to the `viewport <div>` element. By changing the font size of the root (viewport) element relative to the window's width, all the contained elements also inherent this `font-size` CSS property which can then be used through the use of *em* to scale individual elements. This approach also made it easy to retain a specified aspect ratio, preventing any oddities when displaying on landscape displays.

For instance one such feature that was initially being used to develop *MindFlip* was *viewport units*. This feature allows for the scaling of elements based upon the current viewport's dimensions which is especially useful on mobile devices for scaling visual elements relative to the screen's aspect ratio and pixel density. However the stock web browsers on mobile devices have only recently added support for this feature with it being introduced in version 4.4 of Android and 6.0 of iOS.



Viewport units: vw, vh, vmin, vmax - Candidate Recommendation

Length units representing 1% of the viewport size for viewport width (vw), height (vh), the smaller of the two (vmin), or the larger of the two (vmax).

*Usage stats:	Global
Support:	60.03%
Partial support:	13.25%
Total:	73.28%

Show all versions	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Blackberry Browser	Chrome for Android	Firefox for Android	IE Mobile
			2.1					
			2.2					
	3.2		2.3					
	4.0-4.1		3.0	10.0				
	4.2-4.3		4.0	11.5				
	5.0-5.1		4.1	12.0				
	6.0-6.1		4.2-4.3	12.1	7.0			
Current	7.0	5.0-7.0	4.4	16.0	10.0	33.0	26.0	10.0

Figure 6.2: A compatibility table for viewport units

In cases like these a developer can either limit their platform target range and make use of the fringe features or use a polyfill also known as a shim. A polyfill is often referred to as code that substitutes the lack of a future API (application programming interface) by utilising existing features to mimic that of newer fully fledged implementations Lawson and Sharp (2012). In the case of this project the lack of *viewport units* support in older mobile browsers present an issue. After searching for a solution the presentation slides of Kadrmas (2012) detailed a technique using *em* units in HTML5 mobile games. *em* units were originally intended for scaling DOM elements relative to the current font size. This technique was implemented in the project and allowed the game to correctly scale across differing screen sizes. When a window `resize` event is dispatched the `resize` function in `viewport-model.js` uses the window's current dimensions to set its values appropriately. These changes in the model are then listened for by `viewport-view.js` then applies them to the `viewport <div>` element. By changing the font size of the root (viewport) element relative to the window's width, all the contained elements also inherent this `font-size` CSS property which can then be used through the use of *em* to scale individual elements. This approach also made it easy to retain a specified aspect ratio, preventing any oddities when displaying on landscape displays.

```
resize: function (e) {

    var xwindow = window.innerWidth;
    var ywindow = window.innerHeight;

  // e.g. 16:9 or 4:3 screens.
    var xratio = this.get('xratio');
    var yratio = this.get('yratio');

    var landscape = window.innerWidth / xratio;
    var portrait = window.innerHeight / yratio;

    if (landscape < portrait) {

        this.set('width', Math.round(xwindow));
        this.set('height', Math.round(xwindow * (yratio / xratio)));

    } else {

        this.set('width', Math.round(ywindow * (xratio / yratio)));
        this.set('height', Math.round(ywindow));

    }

  // used by 'viewport-view.js' to set css('font-size').
    this.set('scale', Math.floor(this.get('width') / 3));

}
```

However, even when polyfilling potential feature gaps in browsers, issues can still occur under a variety of circumstances. One of the most reliable ways to ensure compatibility is to actually test on all the mobile devices in the target range. This solution most of the time is impractical, especially funds are limited. The next best alternative is to either emulator such devices or use online tools such as BrowserStack[19]. Figure 6.4 shows thumbnails of *MindFlip* on Android and iOS browsers this is a helpful utility that assists in highlighting obvious incompatibility issues at a glance. The blank thumbnails quickly point out where my game is not loading, which allows for selection and prioritizing of what devices need in depth debugging. However, limited workforce, budget, and time meant this project could really only afford to comprehensively debug devices at hand such as the Samsung Galaxy S4 and the Google Nexus 7 (2012).

Hybrid application frameworks currently serve developers by providing access to specific device APIs that are either inaccessible or have not yet been implemented to a satisfactory standard. However, as browsers and web views on mobile devices improve and shape web standards these will most likely serve as a stop gap until better support for HTML5 applications exists. When transitioning from a hybrid application to solely HTML5 the underlying source code will still be compatible, and if engineered sensibly

it will also require minimal changes.

HTML5 is the fifth iteration of the hypertext mark up language. During its previous iterations such as HTML 4.01 and XHTML 1.1 there was minimal support for multimedia content. This resulted in a void that was promptly filled by third-party plugins such as Adobe Flash and Microsoft Silverlight. Such plugins allowed for easy presentation of media such as video, audio and even allowed for interactivity that was used to great effect in games.

Unfortunately many of these plugins are proprietary and closed-source which prevents developers from fixing and debugging inherent issues in the technology. Instead they are dependant on the technology owner for solutions while being limited to reporting them or finding "temporary" workarounds. Plugins often require some form of installation process as they are "add-ons" and exist outside of scope of the W3C standards.

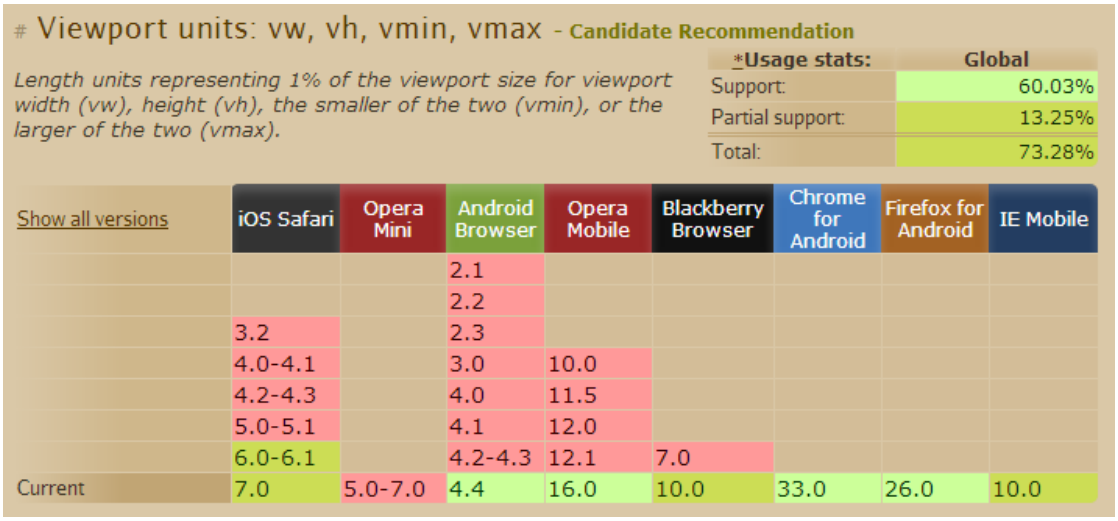
Jobs (2010) famously professed his concerns about Flash and presented sound reasoning as to why Apple had no intentions of supporting or integrating Flash into their iOS devices. Flash was also briefly available for Android but was short lived and eventually discontinued by Adobe. Adobe is continuing to making strides into the mobile platform with their new runtime AIR[20]. HTML5 is still currently a work in progress but the W3C (2012) have made plans to stabilize the specification and help it reach *Recommendation*[21] status as of this year.

Getting a single web application to render and function consistently across a range of browsers can be a huge undertaking. This task has grown exponentially with the introduction of mobile devices and their endless combinations of hardware, operating systems, and mobile browser applications. Each combination of these variables has the potential to produce unique bugs and quirks specific to a particular test case. As the specification of HTML5 was being developed much of the new features have been implemented in browsers at the discretion of the vendor. This has quickly led to cross compatibility issues as older browsers will support less of these new features and not necessarily be consistent with the other browsers that were available at the time.

[21] *https://www.adobe.com/aboutadobe/pressroom/pressreleases/201002/021510FlashPlayerMWC.html*

[21] *http://www.w3.org/2005/10/Process-20051014/tr.html#rec-publication*

Depending on the project specification a developer can rapidly become limited by what features of HTML5 they can utilise. For instance one such feature that was initially being used to develop *MindFlip* was *viewport units*. This feature allows for the scaling of elements based upon the current viewport's dimensions which is especially useful on mobile devices for scaling visual elements relative to the screen's aspect ratio and pixel density. However the stock web browsers on mobile devices have only recently added support for this feature with it being introduced in version 4.4 of Android and 6.0 of iOS.



Viewport units: vw, vh, vmin, vmax - Candidate Recommendation

Length units representing 1% of the viewport size for viewport width (vw), height (vh), the smaller of the two (vmin), or the larger of the two (vmax).

*Usage stats:	Global
Support:	60.03%
Partial support:	13.25%
Total:	73.28%

Show all versions	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Blackberry Browser	Chrome for Android	Firefox for Android	IE Mobile
			2.1					
			2.2					
	3.2		2.3					
	4.0-4.1		3.0	10.0				
	4.2-4.3		4.0	11.5				
	5.0-5.1		4.1	12.0				
	6.0-6.1		4.2-4.3	12.1	7.0			
Current	7.0	5.0-7.0	4.4	16.0	10.0	33.0	26.0	10.0

Figure 6.3: A compatibility table for viewport units

In cases like these a developer can either limit their platform target range and make use of the fringe features or use a polyfill also known as a shim. A polyfill is often referred to as code that substitutes the lack of a future API (application programming interface) by utilising existing features to mimic that of newer fully fledged implementations Lawson and Sharp (2012). In the case of this project the lack of *viewport units* support in older mobile browsers present an issue. After searching for a solution the presentation slides of Kadrmas (2012) detailed a technique using *em* units in HTML5 mobile games. *em* units were originally intended for scaling DOM elements relative to the current font size. This technique was implemented in the project and allowed the game to correctly scale across differing screen sizes. When a window `resize` event is

dispatched the `resize` function in `viewport-model.js` uses the window's current dimensions to set its values appropriately. These changes in the model are then listened for by `viewport-view.js` then applies them to the `viewport <div>` element. By changing the font size of the root (viewport) element relative to the window's width, all the contained elements also inherent this `font-size` CSS property which can then be used through the use of *em* to scale individual elements. This approach also made it easy to retain a specified aspect ratio, preventing any oddities when displaying on landscape displays.

```
resize: function (e) {

    var xwindow = window.innerWidth;
    var ywindow = window.innerHeight;

  // e.g. 16:9 or 4:3 screens.
    var xratio = this.get('xratio');
    var yratio = this.get('yratio');

    var landscape = window.innerWidth / xratio;
    var portrait = window.innerHeight / yratio;

    if (landscape < portrait) {

        this.set('width', Math.round(xwindow));
        this.set('height', Math.round(xwindow * (yratio / xratio)));

    } else {

        this.set('width', Math.round(ywindow * (xratio / yratio)));
        this.set('height', Math.round(ywindow));

    }

  // used by 'viewport-view.js' to set css('font-size').
    this.set('scale', Math.floor(this.get('width') / 3));

}
```

However, even when polyfilling potential feature gaps in browsers, issues can still occur under a variety of circumstances. One of the most reliable ways to ensure compatibility is to actually test on all the mobile devices in the target range. This solution most of the time is impractical, especially funds are limited. The next best alternative is to either emulator such devices or use online tools such as BrowserStack[22]. Figure 6.4 shows thumbnails of *MindFlip* on Android and iOS browsers this is a helpful utility that assists in highlighting obvious incompatibility issues at a glance. The blank thumbnails quickly point out where my game is not loading, which allows for selection and prioritizing of what devices need in depth debugging. However, limited workforce, budget, and time meant this project could really only afford to comprehensively debug devices at hand such as the Samsung Galaxy S4 and the Google Nexus 7 (2012).

[22] *http:// www.browserstack.com/*

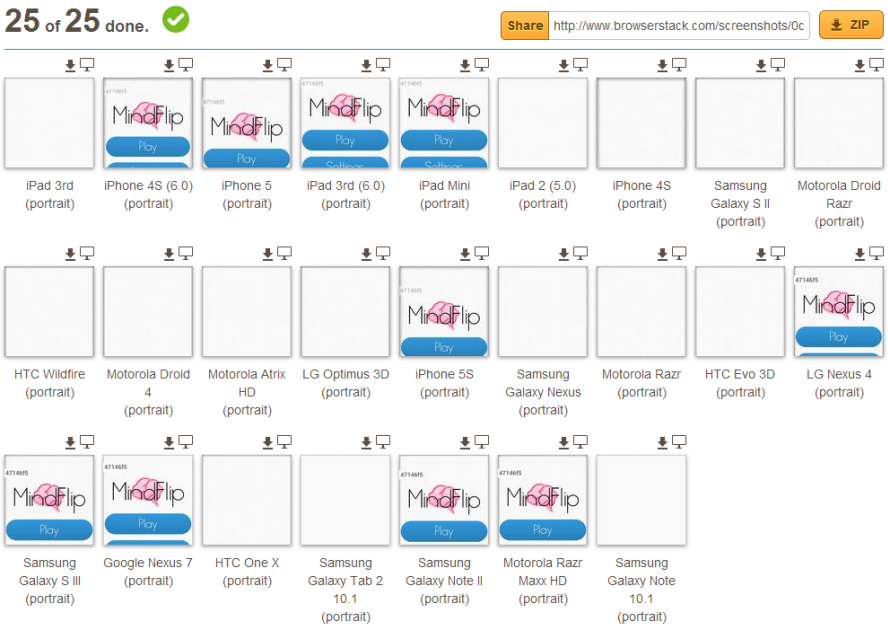


Figure 6.4: Screenshots of MindFlip on various mobile device browsers

Rendering graphics for video games through HTML5 is possible through a variety of ways. DOM based games use elements and styles effectively and take advantage of the browser's long standing rendering conventions. However, most game developers are more familiar with rendering through the rasterization of a graphics layer which can be manipulated at runtime. Both of these approaches are now possible on mobile devices as of recent developments but there is still room for improvement.

The fundamentals of web development are that the content of a website is typically marked down in a `.html` document and the style of this content is embedded alongside or defined within a `.css`. The browser parses the rules found in the stylesheet identifying referenced document elements in the process. This information then instructs the browser how to render the selected document elements often improving their otherwise bland presentation. Although HTML has been getting a much needed update, this begs the question of what its companion CSS has been doing in order to enhance the web experience.

The latest iteration of CSS often referred to as CSS3 provides new and interesting ways in which to style these elements. Notable features include rounded corners, shadows, gradients , transitions or animations , as well as new layouts like multi-columns , flexible box or grid layouts[23].

The support for these on mobile browsers has been around for quite awhile with much of the older versions supporting them through the use of vendor prefixes. These are a side effect of specification CSS3 still being a draft as such features were classified as experimental at the time. However, these are starting to be phased out as browser vendors start to agree on specifics and it also provides a means in which to use them providing legacy support for older browsers.

The state of HTML5 audio support in mobile browsers at this present time is still very limited and plagued with issues. Audio is a key component in delivering a satisfying game experience as it provides stimuli and feedback to the player. This can help further immerse the player in the game experience especially in main stream AAA games. However, audio can play a part in mobile games too emphasising and cuing player interaction that would otherwise be difficult to prompt without the use of popups

[23] *https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3*

and explanatory text. These latter techniques are often avoided because they can bore and frustrate the player while breaking their immersion in the process. Audio can also serve as a brand mechanism for mobile games providing a strong theme that could be associated with good quality experiences.

Commonly the first action most players will take when playing a mobile games to instant mute or lower the audio. Players have a very low tolerance for disturbing sound design and due to earlier mobile games have been conditioned in this way. This stresses that audio must be used sparingly and in the appropriate context in order to deliver the full intended game experience Thomas (2013).

The `<audio>` element is another significant addition from the HTML5 specification. It provides developers a way to define sound content within a HTML document which can then be loaded and played back by web browser. Because of this element web applications can now finally deliver an audio experience that before was impossible to achieve without the use of a third-party plugin.

Support for this element in mobile browsers is widespread however there are many discrepancies between them. Here is a list of some of the known issues that currently effect mobile browsers taken from *caniuse.com*[24].

- Audio played from the element in iOS always plays in a full screen player.
- Playback rate not supported on the stock android browser.
- Only one Audio file can be played at one time in iOS and Android browsers which forces you to use sprites for multiple audios.
- Audio in iOS can not be auto played. It even starts downloading after a user triggered event.
- Volume is read-only on iOS.

Much of these restrictions imposed by mobile web browsers make it awkward to utilise `<audio>` as an competent solution for dynamic sound effects and background music. This also poses a problem when trying to compete with native mobile games

[8] *http:// caniuse.com/ #feat=audio*

as features such as multichannel audio and equalizers come standard with their relative SDK.

The Web Audio API is an experimental high-level JavaScript API that is currently being drafted and developed by Google and the W3C[25]. This API provides richer methods to manipulate and generate audio through signal processing and synthesis. This also includes the ability to load and playback audio files similar to the `<audio>` element. A similarly the MediaStream API[26]developed by Mozilla is trying to achieve the same functionality focusing on the streaming of both audio and video related data.

This definitely appears to be a more appropriate technology for the implementation of game related audio. However, these technologies are much newer than the `<audio>` element and have yet to be implemented in any of the stock mobile browsers with the exception of iOS Safari versions `6.0-7.0`[27].

With the advent of hybrid mobile applications frameworks in some cases it is now viable to bridge to the mobile operating system's native media player. This presents much fewer headaches to developers as these APIs have been around for much longer and are therefore adequately stable.

The game utilises Apache Cordova which has an official plugin that bridges the gap from JavaScript to most naive mobile media players. This allowed me to make calls to the Media plugin's API and have the native media player load and playback audio files that were in my web applications directory. Unfortunately Cordova's official Media plugin had quite high latency which detracted from the tactile feel that improves the user's experience. Research yielded a custom LowLatencyAudio plugin that had been developed by (Trice, 2013) for the PhoneGap framework (the predecessor to Cordova). This plugin was since ported to Cordova by (Xie, 2013) for Android and iOS which was used to solve the tactility issue.

When designing any game it is pertinent that the player enjoys their participation in the game experience. Enjoyment can take many forms, but the majority will boil down to the player obtaining satisfaction from learning. Typically when successfully emulating a mental model akin to that of the game's underlying flow and structure. This

[26] *http://www.w3.org/TR/webaudio/*

[26] *https://developer.mozilla.org/en-US/docs/WebRTC/MediaStream_API*

[27] *http://caniuse.com/#feat=audio-api*

allows for the prediction of cause and effect that takes place within the game's system, Cook (2012). Yet when the terms of engagement with the game change drastically from the norm such as the means of interaction, such is the case with mobile games. It creates an added layer of design complexity that is required for the system to provide sufficient accessibility and feedback to the player.

The choice of interface should be highly weighted relative to the context of the game system. For instance a real-time game system would not be suited to that of a touch interface that simulates a conventional input system such as buttons (unless highly simplified). This is because it lacks tactile feedback and indirectly effects gameplay, creating a sense of incoherence in the user experience. This method however could be applicable in the context of non-gameplay sections such as menu navigation and non real-time game systems such as turn-based game genres, Xu and Bradburn (2011). In these systems because there is no emphasis on player reaction time it lends itself to such interface methods. One advantage to this approach is the omission of a tutorial as most players are accustomed to such conventions.

A better route still is to provide an intuitive means of interface with the game system that could enhance the game experience. Good examples of touch interfaces often take a minimal approach, consisting of few touch gestures, this has substantial benefits over that of the simulated. Angry Birds[28] is a classic example of an intuitive touch interface because the gestures made by the player have direct correlation to visual feedback on the screen. Once the gesture is made the game system simply uses your interaction as the seed of entropy that is then played out (using a physics engine). All the while allowing the user to clearly observe the cause and effect in the game system. Their interaction only formed the initial section of the game loop and for the rest was not obscuring the visual feedback.

User-centred design (UCD) is an interesting approach to mobile game design, instead of relying solely upon the experience and vision of designers it considers the potential future players of the game to be the core design informant. At the core of any game design process is objective to create a meaningful game experience for the player. Two key factors in achieving this are discernibility; providing the player feedback on their

[28] *http:// www.angrybirds.com/*

actions, and integration; showing the player their actions and outcome in the larger context of the game system.

One positive effect of this perspective on game design is it avoids the potential pitfalls that stem from designers. Designers can have a tendency to "fall in love" with their own game ideas as it often originates from their own desires as to what they look for in a game experience. Such approaches may misrepresent what the user base wants from their game experience, as it suffers from lack of wide initial feedback and criticism. The design process is a means to combine logic and information to form satisfactory choices to base a game's blueprint on. The information on which the logic is built can be yielded from a variety of sources; research literature, statistical data, player feedback and the designer's own input, Ermi and Mäyrä (2005).

Dependant on what root is taken when developing an application for the mobile platform most software engineering strategies can be applied through some form. A popular strategy is unified modelling language (UML) which can be applied in the context of mobile game development, Quan-yin et al. (2011). However the advantages presented in this method of this approach may be better suited to that of a business software context. It has the potential hinder development of an evolving interactive application by restricting the ability to create through quick iterative prototyping. Especially in the context of a small development team with a limited development time frame.

"Game mechanics are rule based systems or simulations that facilitate and encourage a user to explore and learn the properties of their possibility space through the use of feedback mechanisms."Koster (2004) Games are engineered to provide enjoyment and a sense of satisfaction for the user. They do this through recording a player's performed action, which in turn causes an effect in the game space. The player then receives feedback often in the form of rewards or new tools which can then be used in the same or newly introduced game mechanics. Human behaviour has made this a good model because we have evolved and survived due to our increasing learning capabilities. The brain's reward centre will release chemicals (dopamine) that encourage such actions during the learning process.

Skinner (1948) proved with his operant conditioning chamber that not only is it possible to condition human reactions but it is also possible to condition human volition and

change the way users make choices. Many successful games that exist in today's market are built on this principle and can sometime exploit its power over their audience. Operant conditioning experiments have shown that to get a player to engage in an activity there must be some sort of feedback that the player takes pleasure in such as a reward. However it has also shown that the best way to get the player to repeat the activity is not to provide feedback consistently. Instead having the feedback be delivered at random intervals or at set time periods has shown to be far more effective.

Games such as World of Warcraft[29] (WoW) and various social games have manipulated this human characteristic beyond their initial novelty. An example of that would be when a player has reached the "end-game" (reaching the maximum character level) in WoW , the key activities that are left to engage in are "raids". These are large-scale game scenarios that require several participants to progress through various virtual dungeons. Most of these dungeons have character class specific rewards that are delivered upon the defeat of a boss. These rewards however, have a randomised probability of occurring which bares significant similarities with that of the operant conditioning experiments. This game design methodology is not exclusive to that of large scale experiences it can also be applied in a very simplistic context. Examples such as Solitaire[30] and Candy Crush Saga[31] have infrequent reward systems for the player which condition them in a similar way. Nevertheless such experiences may be engaging for the player but this does not make them compelling forms of play, Portnow et al. (2013).

The core concept of the game had to be simplistic with room for expansion and modification. This led to the decision of basing it upon simple card game of *Concentration*[32]. The game's core mechanics would consist of the player initially being presented with a set of cards each labelled with a symbol relating to a set. These symbols will only be briefly visible to the player at the start of the game before they are then turned over to obscuring their front-face. This then makes each card indistinguishable from the rest. The player's objective was then to identify matching cards from memory within limitations. This core concept was chosen consciously for a few reasons.

[29] http://eu.battle.net/wow/en/

[30] http://en.wikipedia.org/wiki/Solitaire

[31] http://about.king.com/games/candy-crush-saga

[32] *http://en.wikipedia.org/wiki/Concentration_(game)*

1. It is relatively simple, making it quick and easy for players of all ages to recognise and understand the core gameplay mechanics.

2. It has depth, allowing for additional and more complex gameplay mechanics to be introduced gradually to the player.

3. It will translate well to touch screens, because the fundamental interaction the player will have is selecting cards (whether it be individually or concurrently).

Other mechanics such as matching sets of cards in a specified order were later introduced to put a unique spin on the gameplay. This means players will effectively have to chunk their memories of card positions in particular orders so they can be retrieved later in the reveal stage of the game. Additional shapes and symbols in various placement symmetrical and asymmetrical patterns were also added making players improve their memory capacity and pattern recognition skills.

Throughout the development and prototyping of MindFlip it was important to take into consideration the reactions and feedback of players. So whenever the chance arose I would hand over my phone containing my latest stable build of the game and observe others playing it from a third-party perspective.

Preconceptions can be a double edged sword when used in game design. Often they can be used to draw parallels between activities in contemporary games such as my own. This acts as a shortcut when teaching a player new or similar gameplay mechanics. However, these can also be stumbled upon accidently from a developer's perspective as their own preconceptions about games will likely be vast by comparison to the average player.

This was the case with the initial draft of introductory level. In the beginning the first two card symbols the player encountered were a red nought and a blue cross on a three-by-three grid. After showing the game to my sister while providing no tutorial, the first actions she made were to produce a winning *Tic-tac-toe* game state. This made it apparent that there was a flaw in my game design, because the symbols are common to a pre-existing game namely *Tic-tac-toe* the player receives mixed signals about how to play the game from the offset leading to frustration and confusion. I later decided to go for simpler more abstract symbols such as circles, squares, and triangles in my initial set

of levels do to their generic associations they imply simplicity leaving the player open to learn the game mechanics.

When creating a game targeting HTML5 it can be difficult to know what workflow to invest in. Developers each have their own preferences and beliefs as to what is the best way to work with their priorities often being similar but achieved in different ways. It also has useful tools such as watchers which will automatically build and update your preprocessors code

in its current state (ECMAScript 5.1) is an underwhelming language, especially when compared to more mature languages such as C++ or Java. It lacks native support for object-orientated features such as inheritance, generics, and abstraction. Originally JavaScript was never intended for use in large-scale development projects however, due to its inherent portability it has become widespread. Until its apparent deficiencies are rectified and standardised in its next iteration (ECMAScript 6), developers have to over utilise its strengths, such as being loosely typed. To help aleve JavaScript's growing pains many organisations have been developing alternatives and middle ware to deal with its short comings.

Heavily standardised languages such as JavaScript often have very long iteration cycles, making it difficult for them to adapt swiftly however, this does help to ensure and maintain a language's stability in the long term. By comparison *transpilers* (source-to-source compilers) can transform a better suited custom language into plain JavaScript, by shimming the missing functionality at compile time. They also make it easier for developers to transition to web development without having to learn the semantics of JavaScript. In figure 2.1 are the top thirty-nine most popular programming languages as of January 2014. These statistics were calculated based upon their frequency of occurrence on GitHub (x-axis) and Stack Overflow (y-axis); popular community websites with a focus on programming. By comparing it to one from last year[33]it is clear to see their has been a rise in popularity. Some such preprocessors include CoffeeScript[34], Clojure[35], Dart[36]and TypeScript[37]. The full graph[38]is cumbersome so this snippet is

only of the top quadrant.

The advantages of transpilers often mean that less can be written in order to achieve the same functionality in its targeted language. Simple common activities such as array iteration, class extension, and appending vendor prefixes can become unnecessarily ceremonious especially in the case of JavaScript when shimming non-existent functionality. These transpilers can offer an alternative means in which to achieve the same results with less code and effort required on the part of the developer.

Google's Dart "is a new platform for scalable web app engineering" which comes in the form of several tools. First there is Dart the language which is an alternative to writing plain JavaScript with a host of additional built-in and libraries language features akin to Java Fortuna and Cherem (2013). Second there is DartVM a virtual machine that comes as a standalone program and also happens to be embedded in the Google Chrome browser. The DartVM works similarly to a typical JavaScript virtual machine with the difference being it interprets `.dart` code instead of `.js` with significant performance advantages Schneider (2013). Lastly there is the dart2js compiler, which is effectively a backwards compatibility tool to ensure any application created on the Dart platform will still work through conventional means.

To keep my project structure and programming semantics organised the MVC (model view controller) design pattern was to be employed in development. However, because of the inherent structure of web applications a controller often is not a "true" controller and mislabelled which has led to the term MV* framework implying a substitution of the controller with a more appropriate alternative. Development began initially using TypeScript and then later moving to plain JavaScript, inexperience with the language rapidly led to jumbled spaghetti code. A MV* frameworks looked like an appealing solution that would assist with retaining a legible codebase. Which became imperative when frequently switching between other coursework projects that also required prolonged development in differing programming languages.

[38] *http://redmonk.com/sogrady/2013/07/25/language-rankings-6-13/*
[38] *http://coffeescript.org/*
[38] *http://clojure.org/*
[38] *https://www.dartlang.org/*
[38] *http://www.typescriptlang.org/*
[38] *http://redmonk.com/sogrady/2014/01/22/language-rankings-1-14/*

As with most JavaScript frameworks there are countless variants on offer which made it daunting to select one. It was important to make a firm decision as limiting time constraints prevented me from investing time in learning the pros and cons of each. TodoMVC[39]is a project with the intent of assisting developers select a JavaScript MV* framework. TodoMVC provides developers with a simple todo web application constructed with each of these varying MV* frameworks, allowing them to quickly browser the source code of each and determine whether it suits their needs.

Backbone.js[40]was chosen because of its flexibility, large community following, and modular design. As described in a presentation given by Bull (2013) Backbone.js is the equivalent of a *knife/spoon/fork* on a camping trip. This analogy represents its modularity implying that there is no imposed exclusivity between its internal components. This allowed me to gradually implement the use of its Model, View, Collection, and Router classes throughout my development process not forcing me to learn everything at once but only when it was required.

During this project development a slue of various tools and applications were evaluated for use features and work flow efficiency. The most fundamental tool required is that of a text editors and/or IDE (integrated development environment). They help to maximise the output of a developer by offering efficient procedures such as autocompletion, refactoring and build automation.

Brackets[27] is an open source code editor for web designers and front-end developers constructed with HTML, CSS and JS and maintained by Adobe. It has a comprehensive suite of plugins[28] providing developers a great way to customise it to suit their needs. As the project is open source it allows developers to create and submit their own plugins if their niche has yet to be filled. One of the most useful features is the live preview mode letting me type code while viewing its results instantaneously in the browser. Tweaking the aesthetics of *MindFlip* was easy, changes to colours and styles declaration were instantly visible through live preview which made decisions as to whether they were appropriate swift. The nested nature of this editor being built with the languages it is providing the means to edit also benefits from the inherent portability of these languages

[39] *http:// todomvc.com/*
[40] *http:// backbonejs.org/*

making it avaliable cross platform on Windows, Linux and OSX.

Sublime Text is a sophisticated text editor for code, markup and prose [41]. A popular editor amongst the developer community with is broad coverage of languages and clever short-cuts. The editor is very veritable and can be customised down to the minutiae with packages and plugins many of which can be found on Package Control[42]. It has a very DIY approach expecting users to setup their own build commands and identify the packages that are applicable to their workflow. This has a large learning curve which can discourage some developers that work from a high level of abstraction. Nevertheless, developers looking for such fine control over their development experience will be pleased with the variety of configurations options on offer. Sublime Text is free for evaluation with no time constraints and with the only difference from the full version being a faint *"UNREGISTERED"* water mark.

JetBrains WebStorm is a comprehensive IDE for web development. It has support for many of the preprocessor languages talked about in section 6 right out of the box without plugins and add-ons having to be seeked out by the developer. This was used for the initial stretch of development and boosted the project's productivity by not having to spend lengthy periods of time ironing out the details of the development stack should function. Later in development this IDE was ditched in favour of Adobe Brackets because of performance and incompatibilities with the desired development stack.

One of the challenges that faces mobile developers is how to package their applications for the mobile platform, which spans such a wide variety of hardware and operating systems (iOS[43], Android[44], Windows Phone[45], Ubuntu[46]). This poses a problem especially for the smaller companies as to develop native variants of an application for each platform while retaining a common vision is demanding on both financial and chronological frontiers. Larger companies that facilitate and accommodate such resources, can however frequently benefit from more seamless experience. Most native applications built with their corresponding software development kits (SDKs) which have direct access to device specific application programming intefaces (APIs). This

[40] *http:// brackets.io/*

[40] *https:// brackets-registry.aboutweb.com/*

[42] *http:// www.sublimetext.com/*

[42] *https:// sublime.wbond.net/*

often results in native applications having faster performance and better system user interface integration. Native applications compile to binary machine code that is then interpreted via its relative hardware architecture making it vastly faster than JavaScript which is interpreted at run-time via a browser or web-view. This is most apparent when it comes to developing 3D gaming experiences as these can often be most the most performance taxing, Kulloli et al. (2013).

When comparing HTML5 based mobile applications to native there was another significant disadvantage as there was no way to access device APIs. Such APIs are required to make use of the most hardware features on mobile devices such as the camera, device orientation and battery status. Especially considering that each API varies dependant on operating system and then again on hardware specification. For instance not all mobiles have front facing cameras or high-dpi (dots per inch) screens but a vast majority do. So it becomes quite a challenge when trying to build an application that adapts from phone-to-phone without excluding a particular sector of an audience from the core user experience, Charland and Leroux (2011).

A project with the sole purpose of remedying such issues is that of PhoneGap a mobile development framework originally created by a company called Nitboi. After PhoneGap began to make strides the company was purchased and enveloped by Adobe in 2011, Adobe (2011). Then after the project matured under Adobe's supervision the project was donated to the Apache Software Foundation. This was to ensure that the project was properly maintained when made open source under the Apache License Version 2.0[47]. A formality due to this hand over was that the open source variety of the project had to operate under a different name to avoid trademark ambiguity, Leroux (2012). Adobe's PhoneGap still functions as a separate entity as a distribution of Apache Cordova offering services such as cloud compilation[48].

Apache Cordova[49]is a platform for building native mobile applications using HTML, CSS and JavaScript. It achieves this by providing developers with a set of relative device

[46] *http:// www.apple.com/ ios/*
[46] *http:// www.android.com/*
[46] *http:// www.windowsphone.com/*
[46] *http:// www.ubuntu.com/ phone*
[48] *http:// www.apache.org/ licenses/ LICENSE-2.0.html*
[48] *https:// build.phonegap.com/*

APIs that allow for access of functions previous obfuscated from web technologies by the native system. It also assists with handling a lot of the cross-platform concerns previously mention when targeting multiple platforms and device hardware.

CocoonJS is a comprehensive HTML5 app and game development platform created and developed by Ludei. During the course of this project the platform was functional but still improving some of its vital features. Earlier this year in version `1.4.7` they introduced [50]GPU accelerated WebGL across a widespead of mobile platforms. This feature was innovative at the time because until then because no equivalent existed. Since earlier this year they have made huge strides in improving their platform as well as partnering with similar companies and developers. The disadvantages of CocoonJS at the moment are its compilation service is closed source and although it is currently free Ludei have made no commitments to keeping it that.

It took me awhile to finally settle upon the debugging solution of my development stack and I tried a few approaches with varying success. It was important to be able to quickly prototype on my desktop computer as well as my mobile devices. On desktop computers this is relatively easy as most desktop browsers applications have built in development tools with a variety of features. However, understandable this is not the case commonly on mobile browsers as screen space is limited and it would be difficult to convey the vast streams of data are under the hood of a typical web application. Even the bare minimum tools such as a console are often absent in mobile browsers which are a necessity when trying to diagnose runtime errors in generated by JavaScript as well as reading the output `console.log()`.

Apache Ripple[51]is a companion project of Cordova consists of a web based mobile environment simulator design to emulate the conditions of a packaged Cordova application. It has options for manipulating mobile specific features such as GPS (global positioning system) geolocation and device orientation. This can be useful for developers when rapidly prototyping an application that makes use such mobile specific features. This solution was used for a while until the lack of up-to-date documentation and dwindling support hampered progress during early development.

[35] *http:// cordova.apache.org/*
[50] *http:// blog.ludei.com/ cocoonjs-1-4-7/*
[36] *http:// ripple.incubator.apache.org/*

Adobe Edge Inspect[52]previously known as Adobe Shadow is a tool built on top of Weinre that lets developers preview and inspect across multiple devices concurrently. When it was being assessed for use in this project's development the free version was limited to testing on one device, since then it has become exclusively a paid product. Another disadvantage that came with using this solution is the web application had to be viewed through their Adobe Edge mobile application similar to that of a browser. This made it impossible to debug the game through its targeted environment bundled within a Cordova application.

Chrome DevTools[53]was definitely the most comprehensive and painless solution discovered during development. DevTools encapsulates mobile emulation[54]as well as remote debugging[55]of Android's WebView on versions 4.0 and above which Cordova uses to display the embedded web content. During development some of these features were classified as "experimental" but still accessible through Chrome Canary[56], a custom variant of Google's browser designed for developers and early adopters of fringe web technologies. DevTools provided a quick and effective way to iteratively test the game first on the desktop platform, a mobile emulation, and finally when deployed on mobile devices.

[39] *http://html.adobe.com/edge/inspect/*

[56] *https://developers.google.com/chrome-developer-tools/*

[56] *https://developers.google.com/chrome-developer-tools/docs/mobile-emulation*

[56] *https://developers.google.com/chrome-developer-tools/docs/remote-debugging*

[56] *https://www.google.co.uk/intl/en/chrome/browser/canary.html*

References

Adobe (2011). Adobe Announces Agreement to Acquire Nitobi, Creator of PhoneGap. *Adobe Press Releases*. *http://www.adobe.com/aboutadobe/pressroom/pressreleases/201110/AdobeAcquiresNitobi.html*.

Bull, A. (2013). Snappy Performance Apps with Ember.js. *The SF HTML5 User Group*. *https://github.com/floatinghotpot/cordova-plugin-lowlatencyaudio*.

Champeon, S. (2001). JavaScript: How Did We Get Here? *O'REILLY WEB DEVCENTER*. *http://www.oreillynet.com/pub/a/javascript/2001/04/06/js_history.html* Accessed: 2014-05-03.

Charland, A. and Leroux, B. (2011). Mobile Application Development: Web vs. Native. *Communications of the ACM, Volume 54 Issue 5, May 2011*. *http://delivery.acm.org/10.1145/1970000/1968203/p20-charland.pdf*.

Cook, D. (2012). Building Tight Game Systems of Cause and Effect. *LOSTGARDEN*. *http://www.lostgarden.com/2012/07/building-tight-game-systems-of-cause.html*.

Croft, S. (2012). Responsive button feedback in PhoneGap apps: a better alternative to -webkit-tap-highlight-color. *http://samcroft.co.uk/2012/alternative-to-webkit-tap-highlight-color-in-phonegap-apps/* Accessed: 2014-05-06.

Eich, B. (2014). JavaScript Taking Both the High and Low Roads. *O'Reilly Fluent 2014*. *http://youtu.be/aZqhRICne_M*.

Ermi, L. and Mäyrä, F. (2005). Player-Centred Game Design: Experiences in Using Scenario Study to Inform Mobile Game Design. *The International Journal of Computer Game Research, Volume 5, Issue 1, October 2005*. *http://www.gamestudies.org/0501/ermi_mayra/*.

Fortuna, E. and Cherem, S. (2013). Dart: HTML of the Future, Today! *Google I/O 2013*. *http://youtu.be/euCNWhs7ivQ*.

Hanselman, S. (2013). JavaScript is Web Assembly Language and that's OK. *Scott Hanselman's Blog*. *http://www.hanselman.com/blog/JavaScriptIsWebAssemblyLanguageAndThatsOK.aspx*.

Jobs, S. (2010). Thoughts on Flash. *Apple*. *http://www.apple.com/hotnews/thoughts-on-flash/*.

Julian, J. (2009). "JavaScript: The Good Parts" Summary. *http://www.slideshare.net/jonathanjulian/javascript-the-good-parts-summary*
Accessed: 2014-08-06.

Juul, J. (2007). Swap Adjacent Gems to Make Sets of Three: A History of Matching Tile Games. *Artifact journal*, 2. *http://www.jesperjuul.net/text/swapadjacent/*
Accessed: 2014-05-06.

Kadrmas, J. (2012). Building HTML5 Games. *cf.Objective()*. *http://dl.dropboxusercontent.com/u/21521496/cf.objective/index.html*.

Keith, J. (2010). Design Principles. *HTML5 For Web Designers*, pages 9–10.

Koster, R. (2004). A Theory of Fun for Game Design. *AGC 2003*. *http://www.theoryoffun.com/resources.shtml*.

Kulloli, V. C., Pohare, A., Raskar, S., Bhattacharyya, T., and Bhure, S. (2013). Cross Platform Mobile Application Development. *International Journal of Computer Trands and Technology (IJCTT), Volume 4, Issue 5, May 2013*.

Lawson, B. and Sharp, R. (2012). Introducing HTML5, Second Edition. *New Riders*.

Leroux, B. (2012). PhoneGap, Cordova, and what's in a name? *PhoneGap Blog*. *http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/*.

Lin, E. (2014). 2014 global mobile application processor market forecast. *DIGITIMES Research*. *http://www.digitimes.com/news/a20140129RS400.html*.

Olmstead, D. (2014). Optimizing WebGL Applications. *Google Developers*. *http://youtu.be/QVvHtWePQdA*.

Portnow, J., Floyd, D., and Theus, A. (2013). The Skinner Box. *Extra Credits*. *http://youtu.be/tWtvrPTbQ_c*.

Quan-yin, Z., Yinaand, J., Chengjiea, X., and Ruia, G. (2011). A UML Model for Mobile Game on the Android OS. *International Conference on Advances in Engineering*. *http://www.gamestudies.org/0501/ermi_mayra/*.

Schneider, D. F. (2013). Compiling Dart to Efficient Machine Code. *Lecture at ETH*. *https://www.dartlang.org/slides/2013/04/compiling-dart-to-efficient-machine-code.pdf*.

Seddon, R. (2012). Introduction to JavaScript Source Maps. *HTML5 Rocks*. *http://www.html5rocks.com/en/tutorials/developertools/sourcemaps/*.

Skinner, B. F. (1948). Superstition in the Pigeon. *Journal of Experimental Psychology, Volume 38, 168-172*.

Takada, M. (2012). Adventures in Single Page Applications. *HTML5 Developer Conference*. *http://youtu.be/BqDJqKGfIiE*.

Thomas, N. (2013). The Rise of Mobile Gaming, and Fighting Back Against The Mute Switch. *Casual Connect USA*. *http://youtu.be/3ptqdzf6tLA*.

Trice, A. (2013). Low Latency & Polyphonic Audio in PhoneGap. *Trice Designs*. *http://www.tricedesigns.com/2012/01/25/low-latency-polyphonic-audio-in-phonegap/*.

Verry, T. (2013). Khronos Group Announces WebGL 1.0.2 Specification and Extensions, Formal Release Expected in April. *PC Perspective*. *http://www.pcper.com/news/General-Tech/Khronos-Group-Announces-WebGL-102-Specification-and-Extensions-Formal-Release-Expe*
Accessed: 2014-05-08.

W3C (2005). Document Object Model (DOM). *Architecture domain*. *http://www.w3.org/DOM/*
Accessed: 2014-05-03.

W3C (2012). Plan 2014. *World Wide Web Consortium (W3C)*. *http://dev.w3.org/html5/decision-policy/html5-2014-plan.html*.

W3C (2014). Abstract. *HTML5*. *http://www.w3.org/TR/html5/*
Accessed: 2014-04-30.

WHATWG (2014). Is this HTML5? *HTML Living Standard*. *http://www.whatwg.org/specs/web-apps/current-work/multipage/introduction.html#is-this-html5?*
Accessed: 2014-04-30.

Xie, R. (2013). LowLatencyAudio Plugin. *GitHub*. *https://github.com/floatinghotpot/cordova-plugin-lowlatencyaudio*.

Xu, S. and Bradburn, K. (2011). Usability Issues in Introducing Capacitive Interaction into Mobile Navigation. *Human Interface and the Management of Information, Part IV: Mobile and Ubiquitous Information, p430-439*.