Vrije Universiteit Amsterdam

Master Thesis

HTML5 Cross-Platform Game Development

Author:
Marek Janiszewski

Supervisors:
Dr. Frank Nack
Prof. Dr. Anton Eliens

A thesis submitted in fulfilment of the requirements for the degree of MSc in Computer Science

November 2014

This page intentionally left blank.

Abstract

The noticeable technological advancements during last few years have had impact on many aspects of our everyday life: mobile phones have become small entertainment centres. Since many use them as gaming consoles, the game development market has started to rise. The crucial aspect of a successful title lies within its substantial coverage throughout different devices. Reaching such successful coverage is a matter of further game development process, towards which this paper aims to contribute. In this paper, I present and compare a number of cross-platform compilers. While the overview alone indicates which cross-compiler may be appropriate for the needs of a given project, the study in this paper reaches beyond that. I also present tools to support the development process and using them I create a working title. It leads to a series of tests measuring the quality of experience and showing that even very simple methods may have impact on the end-user.

Acknowledgements

First of all I would like to thank my supervisors, Frank Nack and Anton Eliens, for agreeing to look after this thesis and having to deal with my sloppy work attitude.
Special thanks to Anton for helping me during these 2 years at the VU and also changing the way I look at code.
Specjalne podziękowania dla moich Rodziców i Siostry. Za wsparcie i wiarę do końca!
Last but not least I would like to thank all my friends I have met in Amsterdam: Lea, Matteo and many others, for their help,support and all the great moments we had during our adventure!

Contents

Figures

1 Introduction

Due to widespread use of smartphones, they can be seen surrounding us almost everywhere now. It is very difficult not to notice more and more users and owners looking at a small screen held in their hand. The technological progress that advances the possibilities of a smartphone and makes them more and more popular can be observed each year without possessing a specific knowledge about faster processors, bigger amounts of memory, different screen sizes etc..
According to KantarWorldpanel (2013), the top 3 mobile operating systems are Android (85%), iOS (11%), and Wndows Phone (3%). Niche products are BlackBerry Os, Tizen, Firefox Os, Sailfish Os And Ubuntu touch.
The three major operating systems own their prominent places on the top of the sales to the different platforms, that they offer one of the most important factor of their growing popularity - the apps they provide.
Apps are usually small applications, which extend or add new functionalities to the device. The applications can be obtained through shops available exclusively for the following operating system platforms (J.Haag, 2012):

- iOS - Appstore
- BlackBerry - Blackberry world
- Android - Google Play
- Firefox OS - Firefox Marketplace
- Tizen - Tizen Store
- Windows Phone - Windows Phone Store

From rich variety of available apps they can be divided in different types Statista (2014). In June 2014 the 5 most popular categories were:

- Games

– Education

– Business

– Lifestyle

– Entertainment

"Games" lead the most popular types as their total share of all downloaded apps in that period was 19.06%.
With the evolution of smartphones, their increase in calculating power, memory, graphics possibilities etc, people started using them as mobile handheld gaming consoles – 80% of the total time spent using mobile phones is for using apps, and 32% for playing games.

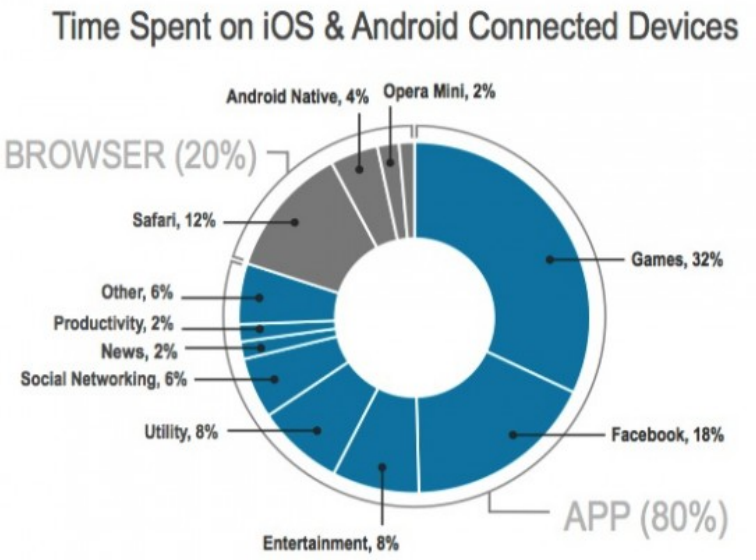

FIGURE 1.1: Time spent on mobile devices Bosomworth (2014)

With such a significant part in the mobile market share, games seem to be a very tempting area for investors and development studios alike. Great number of users is likely to generate substantial revenue. As checked in August 2014, top mobile games like "Clash of clans"[1] , "Candy crush saga"[2] or "Game of war: fire age"[3] have set their daily revenue level above 1 million dollars ThinkGaming (2014). What is worth mentioning is that all these titles are freemium games - they can be downloaded and played for free, but to get extra bonuses and perks, players need to pay micro-transactions.
Shipping an AAA title process consists of the following steps J. Busby (2004):

– planning

[1] http://www.supercell.net/games/view/clash-of-clans
[2] http://www.candycrushsaga.com/
[3] http://www.gameofwarapp.com/

- design phase
- development
- testing
- marketing

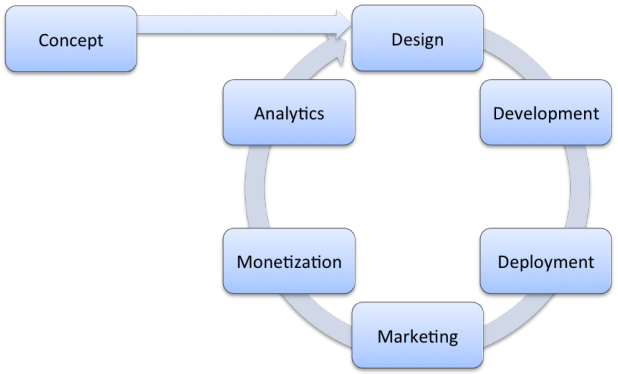

FIGURE 1.2: Game development process

Each process, depending on the size and complexity, requires a significant amount of specialists from each field involved in order to ship the final product. To cover as big market as possible it is necessary to provide a product for different platforms. Overcoming the differences with multiple development environments and various code-bases, may be possible by designing the game first and later on deploying it using cross-platform compilers.
Observations of the global widespread of the mobile market and game apps popularity brought up the following research question into focus:

1. Which cross-platform compilers to use for fast and efficient game development?

The hardware differences between mobile devices also derived further research problems:

2. Can "building" tools help in overcoming the different efficiency between various devices?
3. 3. If so, what is the impact on the quality of experience of the end-user?

This paper is organised as follows: chapter 2 presents the background information on cross platform approach and describes the technologies used to achieve it in mobile environment. Chapter 3 contains a comparison of cross platform compilers and presents tests checking their efficiency. Chapter 4 describes the tools helpful in game development,

the architecture used in the HTML5 game created for the purpose of this thesis and a method for optimizing the software with regards to a different hardware. It is followed by a discussion on the results and a short summary.

2 | Background

Reaching nowadays as vast amount of targeted users as possible across such a variety of available phone models and platforms can be achieved by implementing cross platform support for the product. Cross platform means that the software concepts are implemented and operated on multiple platforms PC-Mag (2013).
Such kind of software may be divided into two types:

- requiring individual building or compilation for each supported device
- directly running on any platform without special preparation, e.g. a software written in an interpreted language or a pre-compiled portable bytecode for which the interpreters or run-time packages are common (or standard) components of all platforms PC-Mag (2013).

With current state of mobile platforms for delivering the first type of cross platform software the team needs multiple skills. Most of the platforms require their applications to be developed using specific programming languages and to be followed by certain standards.
Using different languages requires various development environments. For example iOS requires Objective-C[1] with Xcode[2] and on the other hand Android requires JAVA[3] with supported IDEs. Least popular platforms like Tizen, Firefox OS and Ubuntu Touch use HTML5 and Javascript A. Charland (2011).
Such big variety of languages may require few development teams working simultaneously on the same product, requiring even the smallest changes to be implemented by different developers. It may also lead to different release dates for various platforms.
The other cross-platform approach can be divided into two types H. Heitkotter (2012):

- employing the runtime environment

[1] https://developer.apple.com/library/mac/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html
[2] https://developer.apple.com/xcode/
[3] http://www.java.com/

– cross-platform compilers, generating platform-specific apps from a common code base at compile time.

2.1 | Runtime Environments

The employed runtime environment interprets the app's code at runtime and executes it. It has to be specific for each mobile platform, while the app's source code is platform-independent.
Three different kinds of environment can be identified: the Web browser, a hybrid of Web and native components, and self-contained environments H. Heitkotter (2012).

2.1.1 | Web Browser

Applications using the web browser as runtime environment are usually called mobile web apps. Taking advantage of standardised mobile internet browsers support, there is usually a single version required for different kind of devices. Most important feature of this approach is to implement a fluid layout fitting various screen resolutions.
As such applications are accessed via the browser, they act just like regular websites. This results in two main problems:

1. they cannot be installed on the phone as a separate app (in some devices there may be a shortcut to a website saved as an icon, the iPhone is an example)
2. they do not have access to phone specific functionalities and widgets like notifications, usage of built-in hardware or cross application communication.

2.1.2 | Hybrid Applications

Hybrid approaches emerged from the lack of native functionalities Mahemoff (2011). They work like the previously mentioned web apps using the browser runtime environment, but in addition are wrapped in a native engine. This extra layer allows making API call specific for the device allowing taking advantage of hardware possibilities. Because of the native package they cannot be retrieved via a url address, thus, have to be installed like regular apps in the operating system.

2.1.3 | Self Contained Runtime Environments

Self contained runtime environments are built from scratch and not based on any other previously existing engines. There are designed and developed according to the needs of apps they will be used at. An example of such an environment is Adobe Integrated Runtime[4], developed by Adobe Systems for building Rich Internet applications (RIA) that can be run as desktop applications or on mobile devices. The RIA are programmed using Adobe Flash[5] , Apache Flex[6] (formerly Adobe Flex), HTML, JavaScript and XML Adobe (2014a).

2.2 | Cross Platform Compilers

Cross platform compilers are the tools that concentrate on translating a single source code into native applications. These tools bridge the requirements of the native device APIs with the chosen programming language of the single source code Research2guidance (2014). They allow wrapping up an earlier developed HTML5 application and output a package available for upload to different platforms.

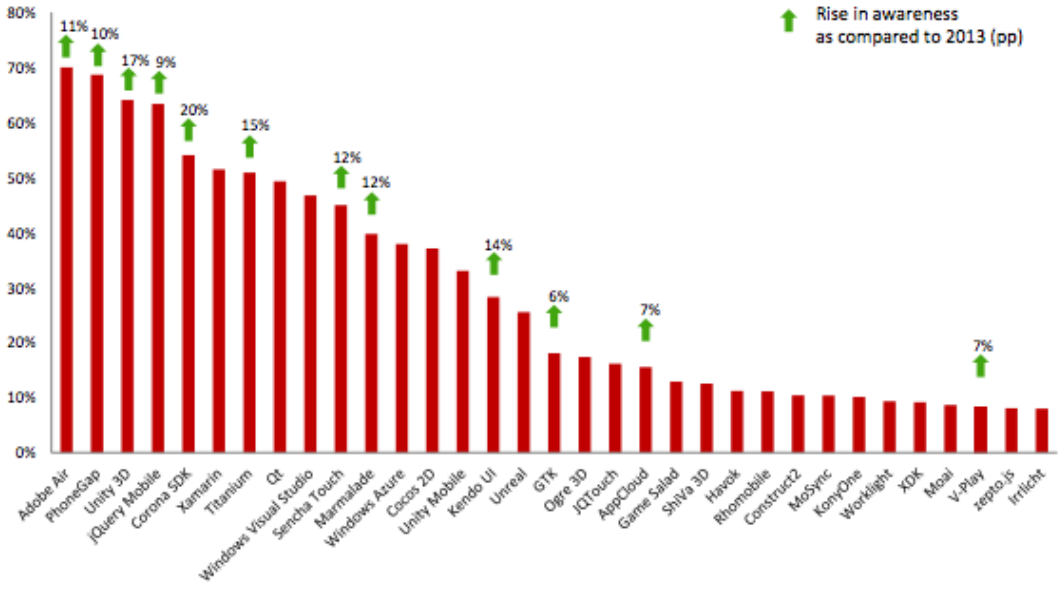

FIGURE 2.1: Cross-platform tools awareness among developers B. Lawson (2012)

[4] http://get.adobe.com/air/
[5] http://get.adobe.com/pl/flashplayer/
[6] http://flex.apache.org/

2.3 | HTML5

HTML5[7] (HyperText Markup Language) is a mark-up language used to prepare websites to be rendered by a web browser. It describes every element contained on a single page by set of given specific tags, allowing to manipulate them and set up as desired by the user. After almost 10 years of HTML4[8] , which was mostly focused on static presentations, in 2008 W3C (World Wide Web Consortium) presented a working draft of HTML5 (final release is planned for 2014). The main difference between those two versions is the focus on web applications. With the evolution of the Internet the pages have begun to get more interactive and concentrate on the user generated content B. Lawson (2012). Introduction of APIs for audio and video control, web storage, cross-document messaging, etc allows developers to include such services without need of any third party plugins or tools installed on the users' device.

Among this rich variety of features, one needs special attention: canvas. It is an element that allows for dynamic shapes and bitmaps rendering while using JavaScript. With its image manipulation functionalities it facilitates the generation of graphs, presentations and games directly in the browser, not requiring a plugin, such as Flash. It is very important for the cross platform means, that canvas element is supported by almost every browser on stationary machines and mobile devices CanIuse (2012).

2.4 | JavaScript

JavaScript[9] is an interpreted programming language with object oriented capabilities. It was developed at Netscape Communications in the mid 90's. Despite some naming, syntactic, and standard library similarities, JavaScript and Java are otherwise unrelated and have very different semantics. The syntax of JavaScript is actually derived from C, while the semantics and design are influenced by Self and Scheme programming languages Adobe (2007).

It is commonly used in web browsers, and in that context, the general purpose was extended with objects allowing the scripts to interact with the user, control the web browser and alter the document content that appears within the web browser window.

It is commonly called "client-side" JavaScript to emphasize that scripts are run by the client computer rather than on the server.

In the mid 2000's node.js[10] was released, which introduces JavaScript also as a server

[7] http://www.w3.org/TR/html5/
[8] http://www.w3.org/TR/html4/
[9] https://developer.mozilla.org/en/docs/Web/JavaScript
[10] http://nodejs.org/

side language.
The language evolved quickly and has been standardised by the European Computer Manufacturer's Association (ECMA). Official name of JavaScript, according to ECMA-262 standard is ECMAScript[11].

Technologies described in this chapter tend to help covering the multiplatform environment. With their usage I will try to present answers to the previously stated research questions. In the next chapter I will focus on the first one, regarding the choice of a cross-platform compiler for mobile game development, by comparing available tools and performing a set of tests checking their efficiency.

[11] http://www.ecmascript.org/

3 | Frameworks Comparison

To determine which of the cross platform compilers may increase the efficiency of HTML5 game development I have created a simple game prototype checking their capabilities. It allows testing the entire multiplatform development process with tasks (i.e. building, setting up the work environment etc.), which may happen during the development phase of a professional title served to an app store.

From the available group of cross platform development kits, as outlined in figure 6, I have chosen two very popular compilers dedicated for HTML5 and JavaScript: CocoonJS[1] and Phonegap[2]. The two were chosen because of their popularity and rich support found across web pages. These compilers subject to many communities based around programming boards such as Stackoverflow[3] , which means that a plethora of helpful hints and tutorials were created to move forward with app development.

In the tables below can be found a short comparison of features offered by chosen frameworks, as both of them present a slightly different approach. Evaluation of efficiency between these two tools is described afterwards.

As for cross platform game development, it may seem strange to omit the Unity3d. As seen on figure 6, it is a very popular framework among developers. Many successful games have been developed using Unity during 2014, including the multiplatform title Hearthstone: Heroes of Warcraft[4] , by Blizzard Entertainment. The decision to omit Unity was the lack of canvas element, and also the price packages. Instead of canvas it uses its own API, which is programmable using JavaScript or C#[5] . The difference between premium and free version are quite significant and can be seen under the following link: http://unity3d.com/unity/licenses. Unity has also its own "appstore" but instead of games it offers tools, models or shaders, which come in handy during the development phase.

[1] https://www.ludei.com/cocoonjs/
[2] http://phonegap.com/
[3] http://www.stackoverflow.com/
[4] http://us.battle.net/hearthstone/en/
[5] http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

The following criterion were taken under consideration

1.License and costs

This criterion examines if the developer / company has to pay any extra costs regarding to usage of framework, licensing issues or distribution

2.Available platforms

Which platforms are supported

3.Ease of development

Is the documentation sufficient enough for a quick start with the development process? What is the quality of these documents? Are there any noticeable problems users were facing?

4.Testing

How to test the app? What is necessary?

5.Deployment

Where can the app be distributed? What is the process?

3.1 | Phonegap

1.License and costs

Phonegap is free, open source and built on open standards. In 2014 Phonegap opened Phonegap Build[6] service in collaboration with Adobe. This service offers one private app for free. From 9.99$ it is possible to extend the amount of apps to 25. Otherwise they are hosted on a public GitHub repository. Adobe ID's are also usable in Phonegap Build.

2.Available platforms

iOS, Android, Blackberry OS, Windows Phone, Ubuntu, Firefox OS. Unfortunately not all native features are supported. The table provided at Adobe (2014c) presents a compatibility chart.

[6] https://build.phonegap.com/

3.Ease of development

All apps are developed using HTML5, CSS and Javascript. First a proper SDK needs to be installed, which in my case took some trouble and time. It requires working in the command line and some features were flawed. To set up the environment it took me some time before properly starting the development.
Documentation is available to the public (see Adobe (2014b)). Each part contains step by step instructions, but as mentioned earlier, even with such hints I had few problems with setting the SDK correctly.
A much appreciated feature is the possibility for users to write their own native plugins in a language corresponding to the platform they want to support, and inject them to their app.

4.Testing

Testing requires extra IDE's for the given environment previously installed. Android required Android Studio (Eclipse pre-configured with SDK) and for iOS is the X-code necessary. I have tested the app only on these two platforms, because of the devices available at the time of testing.
For Android there are two options available: testing on a device and in a simulator.
To test on a device only an USB cable connected to the phone is necessary. After few lines in the command line, if the environment is set up correctly the app displays on the mobile screen. Testing in a simulator requires some extra configuration in the Android Studio. There are already few pre-made simulator devices, but to add new ones all the parameters have to be set manually or downloaded from third party users willing to share them.
iOS testing is only available through the built-in X-code simulator, due to Apple rules regarding the apps.
[Update] After I have performed the tests and checked Phonegap again, in 2014, a simulator available for the phone was made available. It requires the proper SDK, and with one command a virtual server to which the app connects is set up. This allows to directly test apps on iOS and Android.

5.Deployment

After building the source, the app is available in proper directories. To publish, it takes the same steps as the publishing of a native app.

3.2 | CocoonJS

1.License and costs

CocoonJS is free, although not open source. An OpenSDK will be available to allow developers to create their own extensions. For this moment to use such features as push notifications, game centres, analytics, in-app purchases or advanced hardware features the users have to apply through a form to become a premium user. Ludei (2014)

2.Available platforms

iOS, Android, Blackberry OS, Windows Phone, Firefox OS, Windows 8, Intel AppUp, Tizen, nook, Amazon

3.Ease of development

CocoonJS was first designed to compile HTML5 games but with few major updates added it has now has also Document Object Model support. It offers cloud based compilations: the developer needs to only upload a directory containing the index.html file. This means that the developer does not have to prepare any special environment like in previous case, just to develop the app with the usual web manner.
As advertised "CocoonJS provides the most performant canvas 2D and WebGL implementation in the market for iOS and Android".
It offers a custom runtime environment called "Webview+", which is supposed to improve canvas performance across different devices.

4.Testing

Developer apps for Android and iOS are available. The application can be run either through the cloud based system if it was uploaded previously, or can be uploaded to the phone via USB. iOS requires iTunes to upload the files, through the app manager.
A much valued feature is a built-in debugging console. It offers the same possibilities as regular browsers like Chrome or Firefox. Messages can be logged and displayed and the same applies in the case of warnings and errors. Also a built-in frames per second display is available.

5.Deployment

After uploading the app to the cloud compiler files for selected platforms are available for download in few minutes. With regards to the publishing, it takes the same steps as while publishing a native app.

3.3 | Efficiency

From the comparison it can be concluded that there are some differences between both tools, such as testing or ease of development. Yet, a relevant issues has not yet been addressed, namely the efficiency of each environment. Rendering efficiency is a very important matter, not only for developers but also designers and finally end users.
A crucial aspect of efficiency with respect to rendering is the frame rate. It describes how many frames (refreshed images) per second are being displayed on the screen. For video games this has a significant meaning because of the heavy amount of different processes computed by the processor and visual elements being displayed on the screen. Keeping the frame rate around 30 frames per second or more can improve not only the quality of experience but also the efficiency of the player M. Claypool (2009).
I have tested the efficiency of both framework on different devices (where possible) and also simulators if necessary. The testing application was a simple drawing program, checking how many images can be rendered until the frame rate drops to 30 frames per second.
The test was based on two variants, both making use of a drawing program developed especially for this purpose. In the first variant the same image, as described in Figure 7, was used in 3 different resolutions (i.e. 64X64, 128X128, and 256X256 pixels). The programme checked for each resolution how many images can be rendered until the frame rate drops under 30 frames per second.

FIGURE 3.1: Icons used for tests

In the second variant the same application was used to generate circles. In comparison to rendering images, creating arches requires much more operations and is more complex for the rendering unit Baulig (2011). Usually during app or game development it is not necessary to draw such primitive shapes by hand but this will serve as a comparison to

image drawing possibilities.

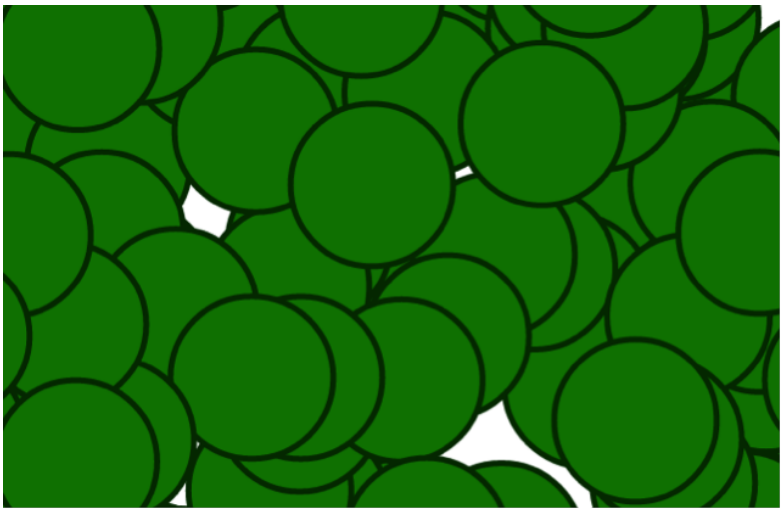

FIGURE 3.2: Circles drawing test on canvas

To measure the frame rate stats.js[7] was used. It provides a simple info box that will help monitoring the code performance.

FIGURE 3.3: Stats.js info box

It measures how much time passed from last call of the event and based on this, presents current frame rate. Setting this in the rendering loops gives accurate information on the state.
To check if the rate drops below 30 frames per second the code has been slightly updated to pop up an alert box with current number of entities and with command to stop drawing. The code is available in the appendix at the end of this paper.
Figure 3.4 - 3.7 present the results for the different hardware and software configurations of the test runs. The tests were performed in webview mode. The specification of the operating systems were:

[7] https://github.com/mrdoob/stats.js/

iPhone 4:

- system: iOS 7
- CPU: 1 GHz Cortex-A8
- GPU: PowerVR SGX535
- memory: 16GB, 512 MB RAM

iPhone 5s:

- system: iOS 7
- CPU: Dual-core 1.3 GHz Cyclone (ARM v8-based)
- GPU: PowerVR G6430 (quad-core graphics)
- memory: 16GB, 1 GB RAM DDR3

Samsung Galaxy Note S8:

- system: Android OS v4.4 (KitKat)
- CPU: Quad-core 1.6 GHz Cortex-A9
- GPU: Mali-400MP4
- memory: 16GB, 2GB Ram

iPad 4:

- system: iOS 7
- CPU: Apple A6X APL5598
- GPU: Mali-400MP4
- memory: 16GB, 1GB Ram

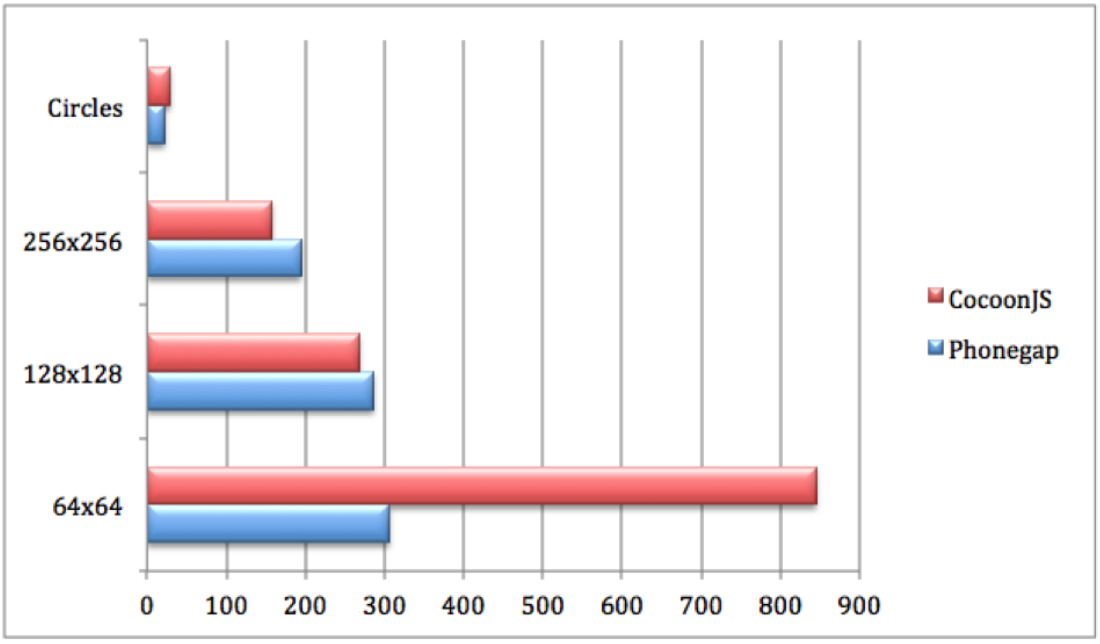

FIGURE 3.4: iPhone 4 test results

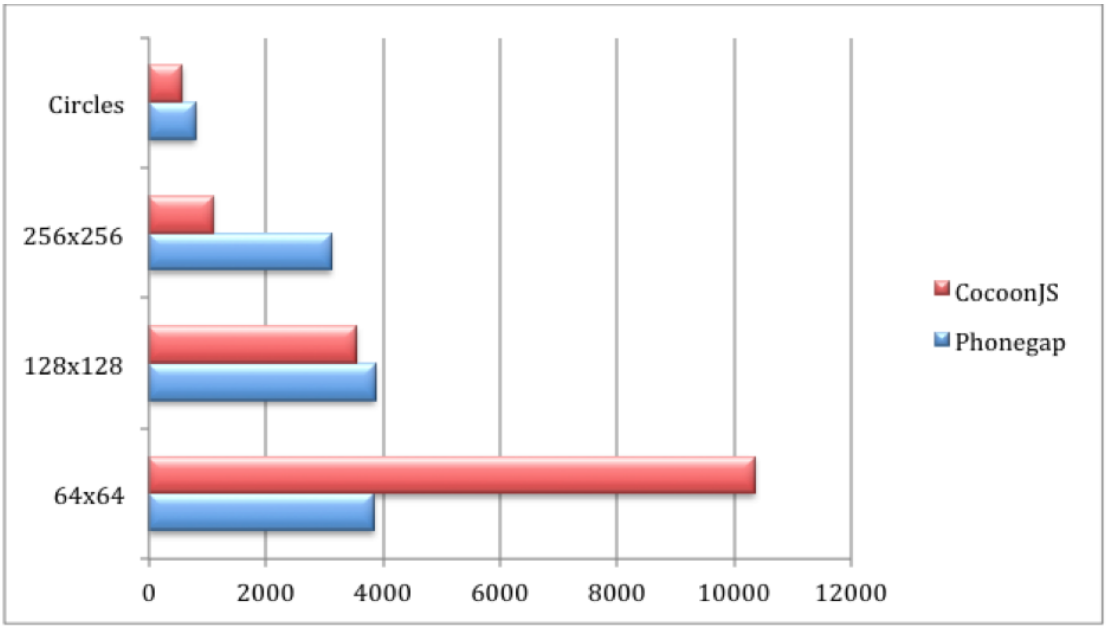

FIGURE 3.5: iPhone 5s test results

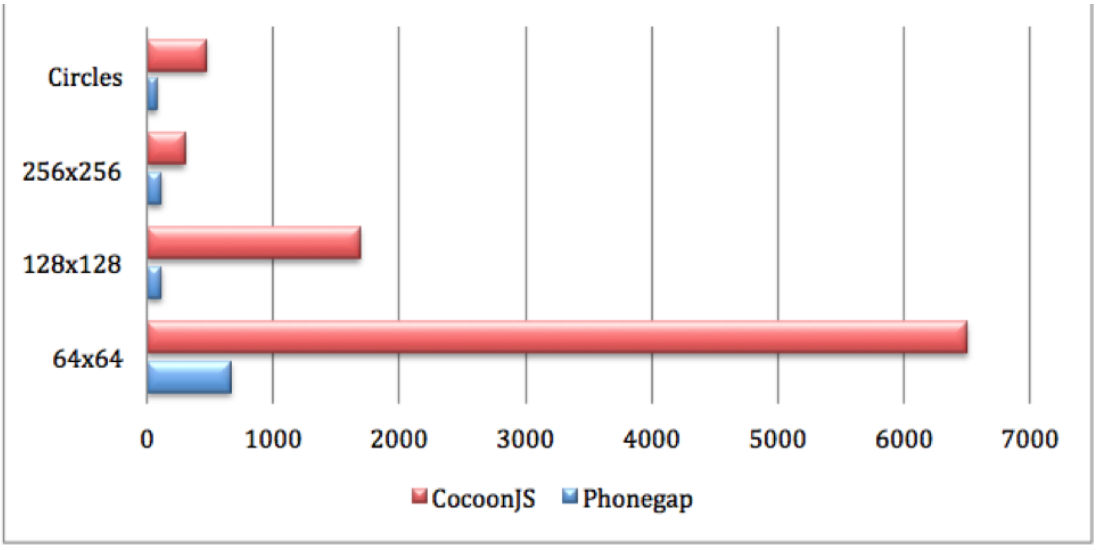

FIGURE 3.6: Samsung Galaxy Note S8 test results

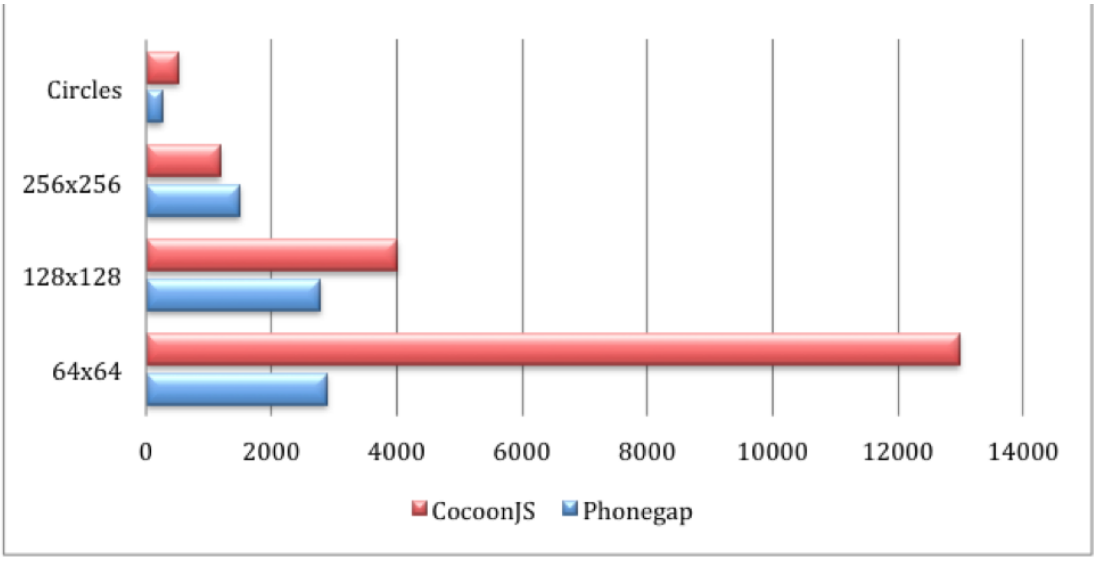

FIGURE 3.7: iPad 4 test results

Looking directly on the results and picking the winner between Phonegap or CocoonJS is not an easy choice. In most of the cases with Apple devices Phonegap is at the least in the lead. Exception is the Samsung device with Android installed, where CocoonJS behaved much better than the other test subject.
What is interesting to note is that on every device in the test using 64x64px images Cocoon declassified its opponent. The results are significantly better. Being able to render few thousand more images gives lots of opportunities in games, especially creating visual effects, consisting of huge amounts of tiny particles.
Such huge difference in efficiency as described above, combined with previously described features, such as dedicated webview+ mode, ease of development and easy deployment, directed me on choosing CocoonJS for the further development involved in this work. Except the efficiency, the deciding factor was the ease of installation and testing. Also as mentioned earlier, the test involving rendering of circles got the lowest scores because the big amount of calculations allowed only for the comparison in rough terms of quantity.
In the next chapter describing the process of creating my HTML5 game, I will take advantage of these finidngs and describe a simple method to prepare the product for different platforms.

4 | Game Development

In this chapter I describe the process of creating my game for the purpose of this paper. It consists of the workflow description, architecture and useful tools to speed up the multi-device development process.
The concluding paragraph will focus on testing a method developed for limiting the special effects for this research. It will focus on measuring users' quality of experience by playing the same game on different devices, suited for given platforms, to help answer the research questions regarding the usage of "build" tools to overcome the differences between different devices and testing the impact on end users quality of experience.

4.1 | Game Design

The idea for my game was reasonably simple: to create an engagement for casual players, allowing relaxing for few minutes while e.g. commuting.
The whole gameplay is very easy, even for those not spending their time on virtual entertainment. A user has to connect as many squares from the same colour as possible within 20 turns. The more blocks are connected, the bigger the point multiplier, resulting in getting higher scores. During each turn, the blocks that have been connected disappear and the board fixes it position by adding new ones. Random colour fillings provides for unpredictable combinations making the game more exciting. Example of the interface is presented on figure 4.1.
Such an easy gameplay does not even require any extra instructions for the user. To learn what to do at the beginning of the game, the player has only two squares on the screen, which need to be connected. This will already provide the user with enough information to carry on with the game. Short time needed for playing was also intended for carrying out tests on end users measuring their quality of experience.

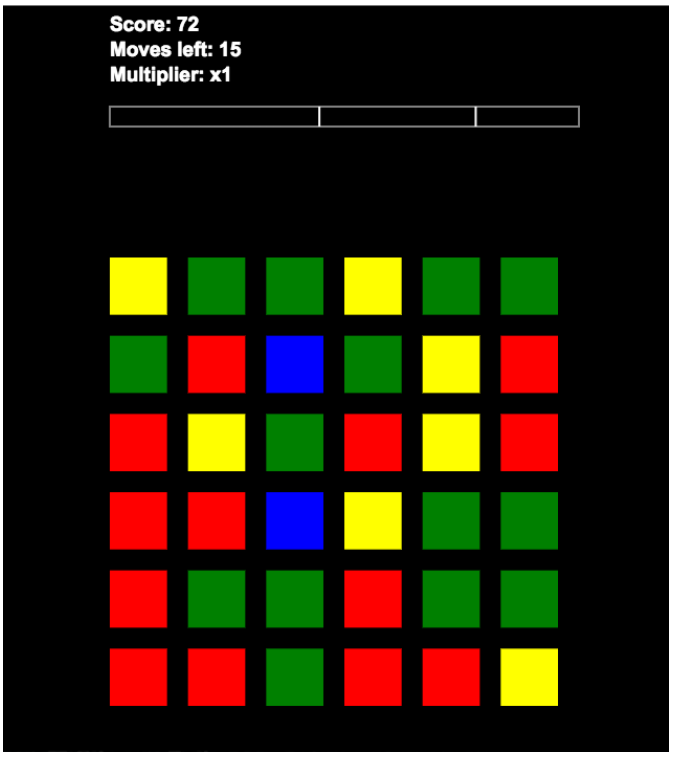

FIGURE 4.1: The games user interface (main scene)

4.2 | Technologies

For the development, earlier described CocoonJS was chosen. As the wrapper uses JavaScript, only this language could have been used for production.
The choice of cross-compiler was not only based on the tests held in chapter 3.3. As seen in the charts, CocoonJS was not in lead in all categories. Selection should be also based on the workload and needs dictated by the developed project.
To automate and speed up the whole process I used few extra tools:

- Bower
- Grunt
- TexturePacker

Each of them is free (TexturePacker can be upgraded to premium version adding extra features) and boosts up repeatable and monotonous steps being faced while working on such a project.
It is important to add, that the tools presented here have no effect on the end product (excluding plugins designed for preparing the code for different versions), and are used

only to improve the workflow in combination with the architecture described in section 4.3.

4.2.1 | Bower

As JavaScript is one of the most popular programming languages currently used (see fig. 4.2), front-end development is at its best. Huge amount of libraries to choose is available and all of them being updated frequently to fix with previous bugs or implement compatibility between different browsers.

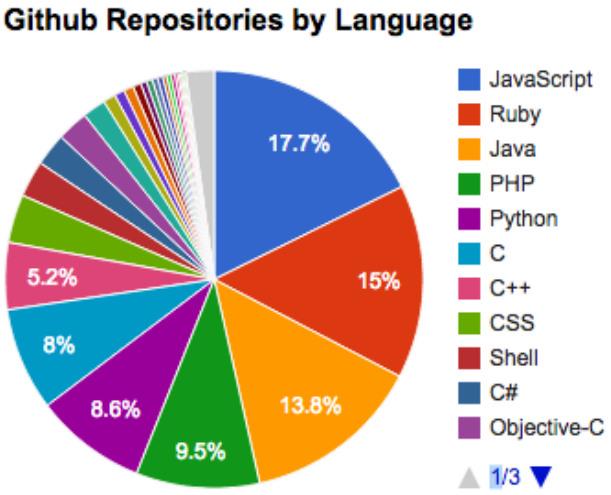

FIGURE 4.2: Most popular languages on Github in 2014 Warner (2014)

Keeping all the tools in a project up to date may be a serious problem, especially when it contains a number of different sources spread across Github or other repositories. To help solving that problem Bower was introduced. It is a package manager responsible for frontend libraries.
Built on top of node.js and npm requires just a few lines in the command line to work and a json file containing the necessary packages.
Libraries can be searched for in two ways: either using the web interface of the component library located at `http://bower.io/search/` or from the console.

Bower automatically downloads the latest available version. It is possible to download not only the newest build, but inside the bower.json file, user can determine which version should be downloaded.

```
MacBook-Pro-Marek:game marekjaniszewski$ bower search jquery
Search results:

    jquery git://github.com/jquery/jquery.git
    jquery-ui git://github.com/components/jqueryui
    jquery.cookie git://github.com/carhartl/jquery-cookie.git
    jquery-placeholder git://github.com/mathiasbynens/jquery-placeholder.git
    qunit git://github.com/jquery/qunit.git
    jquery-file-upload git://github.com/blueimp/jQuery-File-Upload.git
    sizzle git://github.com/jquery/sizzle.git
    jquery-waypoints git://github.com/imakewebthings/jquery-waypoints.git
    jquery-mousewheel git://github.com/brandonaaron/jquery-mousewheel.git
    jasmine-jquery git://github.com/velesin/jasmine-jquery
    jquery-autosize git://github.com/jackmoore/autosize.git
    jquery.ui git://github.com/jquery/jquery-ui.git
    jquery-cookie git://github.com/carhartl/jquery-cookie.git
    jquery-bridget git://github.com/desandro/jquery-bridget.git
```

FIGURE 4.3: Bower search results for âĂIJjqueryâĂİ

4.2.2 | Grunt

Grunt is a JavaScript task-runner. It allows automating processes that are repeated constantly throughout the development process. Its substantial community use and rich documentation allows easily and quickly taking advantage of this helpful tool.

```
{
  "name": "my-project-name",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-jshint": "~0.10.0",
    "grunt-contrib-nodeunit": "~0.4.1",
    "grunt-contrib-uglify": "~0.5.0"
  }
}
```

FIGURE 4.4: Example content of a package.json file

Plugins available to use are available for public browsing at http://gruntjs.com/plugins.

One of the biggest advantages of Grunt is creating tasks, which define which plugins to run. As an example it is possible to run under a single command such operations as:

1. create a build directory
2. minify all the images and copy them to the destination folder
3. check JavaScript code and if correct minify it, compress to a single file, rename the sources inside HTML files

4. run a set of tests on the destination directory using Karma or Jasmine

5. upload to an ftp server if tests passed.

Although only these five steps as described take part, the amount of time saved on building may increase drastically. Tasks like minifying images, files, uploading to an ftp usually require external third party applications and user interaction, which get very monotonous and as a result may decrease the work performance.
In subsequent part of this paper I will describe my simple plugin for preparing a configuration file to adjust the number of entities displayed inside the game.

4.2.3 | TexturePacker

While developing a game it often requires handling big amount of images. To reduce the number of files and thereby decrease the number of calls and loading times it is possible and convenient to use sprite sheets.
Sprite sheets allow any number of images to be stored in a single image, making the game swifter to load and animations to work with easier P.Retting (2012).
While rendering an image on canvas using JavaScript the starting coordinates, width and height need to be defined over the bitmap WC3Schools (2014). This allows putting multiple images in one bitmap and defining which segment needs to be drawn.

FIGURE 4.5: Animation spritesheet containing three frames

Sprite sheets are especially helpful for animations, because of their ability to put multiple frames in a single file P.Retting (2012). Manual process of creating such a set of images requires significant time investment. Proper placement, including checking positions by hand and putting them to a reasonable piece of code is a very consuming task.
TexturePacker allows loading multiple bitmaps, sorting them automatically using different algorithms and delivering a finished sprite sheet, accompanied by code for a desired platform. Currently it supports multiple game engines like Unity, Cocos2d, Flash and

many others. For HTML5 purposes it is possible to generate a JSON file containing the necessary information on all frames that were included.


```
{"frames": [

{
    "filename": "Comparison Table Mobile Development.png",
    "frame": {"x":2,"y":2,"w":681,"h":303},
    "rotated": false,
    "trimmed": false,
    "spriteSourceSize": {"x":0,"y":0,"w":681,"h":303},
    "sourceSize": {"w":681,"h":303}
},
{
    "filename": "bower.png",
    "frame": {"x":2,"y":307,"w":551,"h":239},
    "rotated": false,
    "trimmed": false,
    "spriteSourceSize": {"x":0,"y":0,"w":551,"h":239},
    "sourceSize": {"w":551,"h":239}
}]
}
```


FIGURE 4.6: JSON file generated by TexturePacker

This notation allows easy adapting and manipulating generated code inside the engine. As seen on figure 19, all filenames are given, which enables to quickly identify the wanted frame, size, width and height and to provide quick access to it without any trial and error attempts.

4.3 | Architecture

Software architecture represents the system's earliest set of design decisions. These early decisions are the most difficult to be fulfilled correctly and the hardest to change later in the development process. Also, they have the most far-reaching effects L. Bass (2003). Following architectural patterns may lead also into an increased efficiency during the coding phase by introducing modularity and re-usable elements L. Bass (2003).
For game development on purpose of this work I have chosen the concept presented by Przemyslaw Sikorski[1] , an HTML5 game developer, author of the award winning game Qbqbqb[2] . Providing clear separation of concepts, allows easy code maintenance and full control over the game. It uses two external libraries:

– underscore.js[3] - provides different heavily used "everyday" tasks out of the box, without the necessity of writing and extending object functionalities from scratch.

[1] http://rezoner.tumblr.com/
[2] http://qbqbqb.rezoner.net/
[3] http://underscorejs.org/

As an example: retrieving first or last elements from array, extending objects, cloning etc.

– canvas query[4] - provides jquery-like chaining, and manages event handling

The heart of the engine is the "Application" (application.js). Main function is to initialize the canvas, create loaders and assets (explained further). It also dispatches the events during runtime, which saves lots of effort dealing with timing events.
Next very important part is the "Scene" (scene.js). Scenes describe a state of a game such as menu screen, ending credits, main game etc.
Each element of the game: tree, cloud, player, points etc. are treated as entities. All of

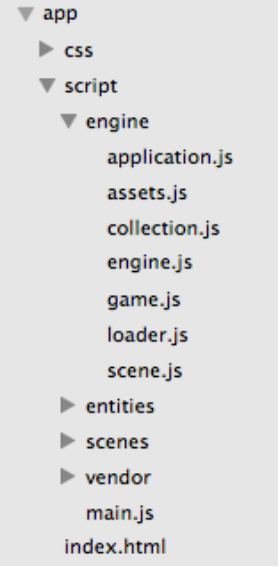

FIGURE 4.7: Folder structure

them have their own private properties but also share common methods: step and render. Step event contains the logic being updated and every frame and render describes what should be performed during the rendering phase. With this approach the game can call all entities currently created and perform given steps on them while in the game loop. It is not important what type it is, for the program it is just an entity that needs to be updated.
Assets are external media that need to be loaded, for example images, audio, video. While loading these types requires some extra code it is helpful to have a wrapper containing all the important pieces while not having to repeat redundant lines of code.
As it is easy to observe, this set of elements provides basic functionalities for quick scaffolding and prototyping. It is not as rich as ready out of the box frameworks such as Impact[5] or Phaser[6] but those actions allow for a complete control over every element.

[4] http://canvasquery.com/
[5] http://impactjs.com/
[6] http://phaser.io/

Keeping the elements modular, furthermore, allows reusing them in other projects, making future development easier Crockford (2008).

4.4 | Cross Platform Deployment

Based on previous tests from section 3.3 it is easy to observe that different devices may be missing the amount of memory required to render a big amount of objects. In case of the simple game all objects are quite small and static (section 3.1, they do not require any calculations between rendering different frames), so I fit them under the category of 64x64 pixels. To show the difference in action between iPhone 4 and iPhone 5s , after each move involving blocks of the same colour to disappear I have added a particle system effect spawning a given number of small entities moving around the screen and disappearing. It looks similar to a fireworks explosion.

Particle systems are used in computer graphics to produce animated effects such as smoke, fire, explosions, water, electricity, flocking and many other natural and imaginative special effects der Berg (2000). They are defined by multiple points in space and a set of rules describing their behaviour, appearance E. Hastings (2009). The amount of particles may vary from a single digit to thousands of entities. Very often is the case that describing their behaviour may require a lot of calculations, and especially when dealing which such big systems it may be hard for the processing unit.

Spawning particles creates after every move a big amount of entities. To stop this problem I have defined a variable inside the engine determining the amount of objects, which can be displayed on the screen.

The architecture described previously allows creating containers for entities. One container was used for "necessary objects" such as blocks, scores, buttons etc. The second container is used for keeping the extra special effects separated for easier management.

Each time an event responsible for spawning particles is triggered, the difference between the number of possible entities and the amount of objects in first container is calculated. This allows easily determining how many new objects may be created. In case where the limit is reached, a particle will be removed at random and will create space for a new particle.

To speed up the deployment and avoid any unexpected errors regarding the amount of entities a simple grunt plugin was used, allowing to determine the value of available number of entities for a given device.

Adding the plugin to the grunt pipeline generates a JavaScript file with the given name in the desired destination. According to the device type set in the task options, the output file will contain a corresponding value.

As written in the Grunt documentation Grunt (2013) it is also possible to set different targets for each plugin. Combining this with different tasks it is possible to set commands like “build-android”, “build-iphone”, “build-ipad” etc. (example in Appendix A). Such easy building configuration may increase the quality, save time, and at the same time allows avoiding manual settings inside the code each time a build for different platform has to be performed.

5 Discussion

The final goal is to test the built game in public. I let a group of 10 people in the age between 23-30 to play it on two different devices (iPhone 4 and iPhone 5s) with various setting and check their opinion on the quality of experience.
Quality of experience is the overall acceptability of an application or service, as perceived subjectively by the end-user A. Takahasi (2008). My questions asked to the testers was, whether they noticed any difference between all of these games played on different devices. I have not informed the end-users about any differences in settings, only about the models of phones.
The first test was performed on the iPhone 5s running the blocks game with a very big amount of spawning particles. Despite having lots of entities on the screen the frame rate did not drop below 30 frames per second, so the game was running seamlessly.
The second part was to play the same game with the same number of entities using the older version of iPhone. As observed on the rendering tests earlier, the possible numbers of objects to draw had a significant difference. As for this attempt, while destroying few blocks and spawning extra particles the frame rate was dropping below 30 frames per second.
The last task was to play again on the iPhone 4 but this time with the limitations of entities described in the previous chapter. Here the amount of spawned particles was limited according to the device capabilities, providing smooth animations.
From the group of testers, 9 out of 10 have noticed a difference between the two versions on iPhone 4. None of the testers has noticed a difference between versions on iPhone 5 and 4 with limitations being set up.
This test, even though performed on a very limited group of people, may show that simple manipulations may have an impact of the performance of software running on different mobile phones. In this case having a set of numbers for each type of deployment target and a preconfigured plugin for building may help to prepare the app for various platforms in a very short time.
Easy to notice, such methods should be planned during the design phase, allowing to prepare assets and possible outcomes earlier. Otherwise it may appear to be difficult to

implement them during development without significant changes affecting the end user. This method is just a basic indicator how to prepare the application. It can be explored further by adding such functionalities as:

- prioritising entities - each object may have a different level of importance, based on which if necessary they are removed in an ordered way
- better memory management - instead of removing entities every time, reuse them
- cross-platform tests - current tests were performed only on different versions of the same device due to their availability. Other devices with different operating systems should be tested as well

Also there may be different ideas how to implement the limiting of entities or add new tools to the build pipeline to achieve the goal in a more automated way. The idea of using the architecture recommended in this text is also only an indication. It is also possible to explore among many other frameworks prepared especially for game development using HTML5 such as:

- Impact
- Construct[1]
- Phaser
- Turbulenz[2]

and many others available online.

[1] https://www.scirra.com/
[2] http://biz.turbulenz.com/

6 | Conclusions

In this paper I have tried to answer the following research questions:

1. Which cross-platform compilers to use for fast and efficient game development?
2. Can "building" tools help in overcoming the efficiency differences between different devices?
3. If so, what is the impact on the quality of experience of the end-user?

Regarding the first question I have compared two popular frameworks among HTML5 game developers: Phonegap and CocoonJS. Describing their features and performing basic efficiency tests, I have chosen CocoonJS for further development. As stated earlier, CocoonJS is not the best available universal cross-compiler. The choice of the cross-compiling tool should be based on its features and project requirements, which can be fulfilled with selected compiler.
To answer the second problem, I have presented a simple Grunt plugin, allowing determining how many objects can be displayed on the screen, prepare the files for building for different devices and a simple method to limit them in case of extensive spawning.
In order to check how does the limiting of entities affect the quality of experience, a series of 10 tests was performed on real users. It showed that even such a simple method may have an impact and bring value to the project.
The research made for the purpose of this thesis may help future game developers to focus on more efficiency-driven decisions during the design and development phase. There is still much to improve and explore, like cross-platform compatibility regarding audio, 3d capabilities, user interactions, as the mobile game development process (especially regarding HTML5) is for this moment receiving more attention. Despite the noise of the attention remain numerous materials available for research as well as topics and questions, which need more in-depth analysis. One of such examples may be rendering using different JavaScript engines like V8 used by Chrome or Nitro in Safari. Such tasks may require deeper understanding of compiling engines and diving inside compilation processes of various browsers.

A | Code Samples

```
var draw = true;
var canvas = document.createElement('canvas');

var width = window.innerWidth;
var height = window.innerHeight;

canvas.width  = width;
canvas.height = height;
canvas.style.border   = "1px solid";
document.body.appendChild(canvas);

var context = canvas.getContext('2d');

var stats = new Stats();
stats.domElement.style.position = 'absolute';
stats.domElement.style.left = '0px';
stats.domElement.style.top = '0px';
stats.setMode( 0 );
console.log(stats.getFps());
document.body.appendChild( stats.domElement );

var imageObj = new Image();
imageObj.src = '64x64.png';
//imageObj.src = '128x128.png';
//imageObj.src = '256x256.png';
var images = [];
var image = function() {
  this.src = imageObj;
  this.posx = Math.random() * width;
  this.posy = Math.random() * height;

  return this;
}
stats.setFps(90);
var init = function() {
  // Store the current transformation matrix
  context.save();
```

```
38
39    // Use the identity matrix while clearing the canvas
40    context.setTransform(1, 0, 0, 1, 0, 0);
41    context.clearRect(0, 0, canvas.width, canvas.height);
42    stats.begin();
43    // Restore the transform
44    context.restore();
45    for(var i=0; i<images.length; i++) {
46      context.drawImage(images[i].src, images[i].posx, images[i].posy);
47    }
48    if(draw) {
49      images.push(new image());
50    }
51    if(stats.getFps() < 30 && draw) {
52      draw = !draw;
53      alert(images.length);
54    }
55    stats.end();
56  }
57  // shim layer with setTimeout fallback
58  window.requestAnimFrame = (function(){
59    return  window.requestAnimationFrame       ||
60            window.webkitRequestAnimationFrame ||
61            window.mozRequestAnimationFrame    ||
62            function( callback ){
63              window.setTimeout(callback, 1000 / 60);
64            };
65  })();
66
67
68  (function animloop(){
69
70    requestAnimFrame(animloop);
71    init();
72  })();
73  // place the rAF *before* the render() to assure as close to
74  // 60fps with the setTimeout fallback.
```

Listing A.1: Image rendering tests - main.js

```
1  module.exports = function (grunt) {
2      "use strict"
3
4      grunt.initConfig({
5          builder: {
6              iphone: {
7                  options: {
8                      appName: 'Engine',
```

```
                    device: 'iphone',
                      destFile: 'config',
                      destDir: 'iphone-build'
                  }
              },
              android: {
                  options: {
                      appName: 'Engine',
                      device: 'android',
                      destFile: 'config',
                      destDir: 'android-build'
                  }
              }
          },
          clean: {
              tests: [
                  "test/output.txt"
              ]
          },
          nodeunit: {
              all: ['test/*_test.js']
          },
          jshint: {
              options: {
                  curly: true,
                  eqeqeq: true,
                  eqnull: true,
                  browser: true,
                  debug: true,
                  forin: true,
                  noarg: true,
                  noempty: true,
                  boss: true,
                  loopfunc: true,
                  evil: true,
                  laxbreak: true,
                  bitwise: true,
                  strict: true,
                  undef: true,
                  unused: true,
                  nonew: true,
                  globals: {
                      process: true,
                      module: true,
                      require: true
                  }
              },
              main: [
```

```
57                  'tasks/**/*.js',
58                  'test/**/*.js'
59              ]
60          }
61      });
62
63      grunt.loadTasks('tasks');
64
65      grunt.loadNpmTasks('grunt-contrib-jshint');
66
67      grunt.loadNpmTasks('grunt-contrib-clean');
68
69      grunt.loadNpmTasks('grunt-contrib-nodeunit');
70
71      grunt.registerTask('default', ['clean', 'jshint', 'builder', '
    nodeunit']);
72
73      grunt.registerTask('build-iphone', ['clean', 'jshint', 'builder:
    iphone', 'nodeunit']);
74
75      grunt.registerTask('build-android', ['clean', 'jshint', 'builder:
    android', 'nodeunit']);
76
77      grunt.registerTask('test', ['clean', 'nodeunit', 'clean']);
78
79  };
80 }
```

Listing A.2: Example of Grunts configuration file

B | Online References

1. Clash of Clans - http://www.supercell.net/games/view/clash-of-clans
2. Candy Crush Saga - http://www.candycrushsaga.com/
3. Game of War: Fire Age - http://www.gameofwarapp.com/
4. Objective C - https://developer.apple.com/library/mac/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html
5. Xcode - https://developer.apple.com/xcode/
6. Java - https://www.java.com/
7. Adobe Integrated Runtime - http://get.adobe.com/air/
8. Adobe Flash - http://get.adobe.com/pl/flashplayer/
9. Apache Flex - http://flex.apache.org/
10. HTML5 - http://www.w3.org/TR/html5/
11. HTML4 - http://www.w3.org/TR/html4/
12. JavaScript - https://developer.mozilla.org/en/docs/Web/JavaScript
13. node.js - http://nodejs.org/
14. ECMAScript - http://www.ecmascript.org/
15. CocoonJS - https://www.ludei.com/cocoonjs/
16. Phonegap - http://phonegap.com/
17. Stackoverflow - http://www.stackoverflow.com/
18. Hearthstone: Heroes of Warcraft - http://us.battle.net/hearthstone/en/
19. C# - http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

20. Phonegap Build - https://build.phonegap.com/

21. Stats.js - https://github.com/mrdoob/stats.js/

22. Przemyslaw Sikorski - http://rezoner.tumblr.com/

23. Qbqbqb, the game - http://qbqbqb.rezoner.net/

24. underscore.js - http://underscorejs.org/

25. Canvas Query - http://canvasquery.com/

26. ImpactJS - http://impactjs.com/

27. Phaser - http://phaser.io/

28. Construct - https://www.scirra.com/

29. Turbulenz - http://biz.turbulenz.com/

C | Extra Libraries

Below is a list of extra JavaScript libraries not mentioned in this paper, but helpful during the development process.

WebGL

To support 3D graphics it is possible to use WebGL on newer devices. Libraries listed below provide wrappers and helpers not to deal with pure complicated syntax and allow faster prototyping.

- ThreeJS - http://threejs.org/
- BabylonJS - http://www.babylonjs.com/

Audio

JavaScript does not provide easy methods for audio control. Libraries presented below allow manipulating sounds, creating audiosprites etc.

- SoundJS - http://www.createjs.com/SoundJS
- Timbre.js - http://mohayonao.github.io/timbre/
- Howler - http://goldfirestudios.com/blog/104/howler.js-Modern-Web-Audio-Javascript-

Drawing

To take full advantage of drawing on the canvas element and creating different types of animations requires many lines of code. These libraries help to deal with animations, transitions etc.

- EaselJS - http://www.createjs.com/EaselJS
- KineticJS - http://kineticjs.com/

- FabricJS - http://fabricjs.com/
- Raphael - http://raphaeljs.com/
- oCanvas - http://ocanvas.org/

Utility

In this category I have added just one library, which is very popular among JavaScript developers. It helps managing tweening between values in a very easy way.

- TweenJS - http://www.createjs.com/TweenJS

Bibliography

A. Charland, B. L. (2011). Mobile application development: Web vs native. *Mobile Computing*, 9. [Accessed November 2014].

A. Takahasi, D. Hands, V. B. (2008). Standardization activities in the itu for a qoe assesment of iptv. *IEEE in Communications Magazine*, pages 78–84.

Adobe (2014a). Adobe integrated runtime. http://www.adobe.com/products/air/faq.html. [Accessed November 2014].

Adobe (2014b). Phonegap documentation. http://docs.phonegap.com/en/3.5.0/index.html. [Accessed November 2014].

Adobe (2014c). Phonegap supported features table. http://phonegap.com/about/feature/. [Accessed November 2014].

Adobe, The Mozilla Foundation, O. S. A. (2007). Proposed ecmascript 4th edition âĂŞ language overview. http://www.ecmascript.org/es4/spec/overview.pdf. [Accessed November 2014].

B. Lawson, R. S. (2012). *Introducing HTML5*. New Riders, second edition.

Baulig, D. (2011). High performance ecmascript and html5 canvas. http://www.danielbaulig.de/wp-content/uploads/2011/04/Bachelor-Thesis.pdf. [Accessed November 2014].

Bosomworth, D. (2014). Mobile marketing statistics 2014. http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/. [Accessed November 2014].

CanIuse (2012). Compatibility table for support of the html5 canvas element in desktop and mobile browsers. http://caniuse.com/canvas. [Accessed November 2014].

Crockford, D. (2008). *JavaScript: the good parts*. O'Reilly Media.

der Berg, J. V. (2000). Building an advanced particle system. *Game Developer Magazine*, pages 44–50.

E. Hastings, R. Guha, K. S. (2009). Interactive evolution of particle systems for computer graphics and animation. In *IEEE Transactions on Evolutionary Computation*. IEEE Press, New York.

Grunt (2013). Grunt documentation. http://gruntjs.com/configuring-tasks. [Accessed November 2014].

H. Heitkotter, S. Hanschke, T. M. (2012). Evaluating cross-platform development approaches for mobile applications. In *Web Information Systems and Technologies. 8th International Conference*. WEBIST.

J. Busby, Z. P. (2004). *Mastering Unreal Technology: The Art of Level Design*. Sams Publishing.

J.Haag (2012). Mobile app development and distribution options. http://www.adlnet.gov/mobile-app-development-and-distribution-options. [Accessed November 2014].

KantarWorldpanel (2013). Kantar worldpanel comtechâĂŹs smartphone os market share data. http://www.kantarworldpanel.com/global/smartphone-os-market-share. [Accessed November 2014].

L. Bass, P. Clements, R. K. (2003). *Software Architecture in Practice*. Addison-Wesley Professional, second edition.

Ludei (2014). Cocoonjs feature list. https://www.ludei.com/cocoonjs/features/. [Accessed November 2014].

M. Claypool, K. C. (2009). Perspectives, frame rates and resolutions: itâĂŹs all in the game. In *Proceedings of the 4th International Conference on Foundations of Digital Games*.

Mahemoff, M. (2011). Html5 vs native: The mobile app debate. http://www.html5rocks.com/en/mobile/nativedebate/. [Accessed November 2014].

PC-Mag (2013). Encyclopedia. http://www.pcmag.com/encyclopedia/term/40495/cross-platform#fbid=aHfb3ldkqPq. [Accessed November 2014].

P.Retting (2012). *Professional HTML5 Mobile Game Development*. John Wiley and Sons.

Research2guidance (2014). Cross platform app development tool benchmarking 2014. http://www.research2guidance.com/r2g/Cross-Platform-Tool-Benchmarking-Report-2014.pdf. [Accessed November 2014].

Statista (2014). Most popular apple app store categories in september 2014, by share of available apps. http://www.statista.com/statistics/270291/popular-categories-in-the-app-store/. [Accessed November 2014].

ThinkGaming (2014). Top grossing ios games. http://thinkgaming.com/app-sales-data/. [Accessed November 2014].

Warner, J. (2014). Top 100 most popular languages on github. https://jaxbot.me/articles/github-most-popular-languages. [Accessed November 2014].

WC3Schools (2014). Html canvas drawimage method. http://www.w3schools.com/. [Accessed November 2014].