Bachelor's thesis

Information and Communications Technology

2018

Riku Skyttä

CROSS-PLATFORM GAME DEVELOPMENT

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2018 | 32

Riku Skyttä

CROSS-PLATFORM GAME DEVELOPMENT

The purpose of this thesis was to research cross-platform game development, conduct an internet questionnaire among game industry professionals, find out their opinions about cross-platform development and document the author's experiences when working with it.

The theoretical part of the thesis researches cross-platform development in general and compares different game engines that are suitable for cross-platform development. Some general guidelines and common issues are observed. The result was a list of different cross-platform game engines and their differences, notes on what to take in to account when developing to certain platforms, e.g., peripheral compatibility and aspect ratio and an analysis of testing on different platforms and their minimal testing environments.

The survey was carried out using Google Forms and it was shared in the IGDA Finland Facebook page. In total, ten people answered the questionnaire and the majority of them were programmers. Unfortunately no graphical artists answered, so there were no answers from the graphical perspective of game development. Ninety percent of the respondents had worked with cross-platform development with many different platforms, but the most popular platforms were Android and iOS. Unity was the most used game engine. The respondents mostly thought that cross-platform development is important considering the future of the industry, as it is a way for even small companies to reach a greater audience when the game is available on multiple platforms.

The practical part of the thesis was carried out during a seven week internship. The company wanted their Android and iOS mobile game to be converted to use the WebGL platform. The game was created using the Unity game engine which has a native support for cross-platform development and the WebGL platform, so the game engine did not need to be changed. On the coding level, the project had to be altered so that the game can be played and tested using a web browser. A document was written about this testing and it had information about what requirements and limitations WebGL development has, what functionalities do not work like on the mobile platforms and what should be taken in to account when developing to the WebGL platform. In the end, the game was playable on the browser and it was able to save the players' progress in the browser's cache. The original save function, in-game purchases and analytics did not work with the WebGL platform.

The final result is a comprehensive document about cross-platform development that can be used as a guide on how to get started with it and how certain features function on different platforms. It also provides an insight into how game developers today view the idea of cross-platform development.

KEYWORDS:

cross-platform, game development, game engine, digital games

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikan koulutus

2018 | 32

Riku Skyttä

ALUSTARIIPPUMATON PELINKEHITYS

Työn tarkoituksena oli tutkia alustariippumatonta pelinkehitystä, suorittaa internetkysely pelialalla työskentelevien parissa siitä, mitä mieltä he ovat alustariippumattomasta kehityksestä, sekä kirjoittaa omia kokemuksia ja ohjeita sen parissa työskentelystä.

Teoriaosuudessa tutkittiin, mitä alustariippumaton kehitys on yleisesti, vertailtiin erilaisia alustariippumattomaan kehitykseen soveltuvia pelimoottoreita, sekä käytiin läpi alustariippumattomaan kehitykseen liittyviä ongelmia ja spesifikaatioita. Tuloksena lista erilaisista alusta-riippumattomista pelimoottoreista ja niiden eroista, sekä mitä tulee ottaa huomioon tietyille alustoille kehittäessä, kuten lisälaitteiden yhteensopivuus ja kuvasuhde. Analyysi myös alusta kohtaisesta testaamisesta ja alustojen minimi testiympäristöistä.

Kysely suoritettiin Google Forms -palvelun kautta, ja se jaettiin IGDA Finland -järjestön Facebook -sivulla. Kyselyyn vastasi yhteensä kymmenen henkilöä, joista suurin osa oli ohjelmoijia. Valitettavasti yksikään graafikko ei vastannut, joten tuloksia graafikon näkökulmasta ei saatu. Yhdeksänkymmentä prosenttia kyselyyn vastanneista oli itse työskennellyt alustariippumattoman kehityksen parissa useilla eri alustoilla. Suosituimmat alustat olivat Android ja iOS. Unity oli ylivoimaisesti suosituin pelimoottori. Kyselyyn vastanneet olivat pääosin sitä mieltä, että alustariippumaton kehitys on tärkeä pelialan tulevaisuuden kannalta, sillä sen avulla pientenkin studioiden pelit voivat saavuttaa ison yleisön, kun peli on saatavilla usealle eri alustalle.

Työskentelyn tukena toimi seitsemän viikkoa kestävä työharjoittelu. Työharjoittelun tavoitteena oli kääntää yrityksen kehittämä mobiilipeli Android ja iOS -alustoilta tukemaan WebGL -alustaa. Peli on kehitetty käyttäen Unity pelimoottoria, jossa on sisäänrakennettu tuki alustariippumattomalle kehitykselle ja WebGL alustalle, joten pelimoottoria ei tarvinnut vaihtaa. Kooditasolla projektia tuli muokata siten, että peli saadaan kääntymään niin, että sitä voidaan pelata ja testata internet selaimella. Tästä testauksesta kirjoitettiin raportti, joka piti sisällään ohjeita siitä, mitä vaatimuksia ja rajoitteita WebGL -kehityksessä on, mitkä kaikki ominaisuudet eivät toimi samoin, kuin mobiilialustoilla ja mitä tulee ottaa huomioon, kun kehitys kohdistuu WebGL -alustalle. Peli saatiin toimimaan niin, että sitä pystyi pelaamaan verkkoselaimella ja peli tallentui selaimen välimuistiin. Alkuperäinen tallennusfunktio, pelin sisäiset ostot ja analytiikka eivät toimineet WebGL -alustalla.

Lopullinen tulos on kattava dokumentti alustariippumattomasta kehityksestä, jota voidaan käyttää oppaana kehitys projektin aloittamiseen ja löytämään neuvoja siitä, miten jotkin asiat toimivat tietyillä aluistoilla. Siitä näkee myös pelikehittäjien mielipiteen alustariippumattomasta kehityksestä nykyään.

ASIASANAT:

alustariippumattomuus, pelinkehitys, pelimoottori, digitaaliset pelit

CONTENTS

LIST OF ABBREVIATIONS (OR) SYMBOLS

2D	Two-dimensional
2.5D	Two and a half dimensional
3D	Three-dimensional
API	Application programming interface
C#	C Sharp (Programming language)
C++	Programming language
CPU	Central processing unit
GPU	Graphics processing unit
HTML	Hypertext markup language
IDE	Integrated development environment
IGDA	International Game Developers Association
JS	JavaScript
MIT	Massachusetts Institute of Technology
NDA	Non-disclosure agreement
RAM	Random-access memory
SSE2	Streaming Single Instruction, Multiple Data Extensions 2
SDK	Software development kit
Triple-A	Video game title produced and published by a major publisher in the industry.
VR	Virtual Reality
WebGL	Web Graphics Library

1 INTRODUCTION

The purpose of this thesis was to research cross-platform game development in general and to find solutions on how to approach a project when doing cross-platform development. Cross-platform game development is growing and gaining popularity with titles like *Fortnite* from *Epic Games* gaining 40 million players across 6 different platforms that almost all can play together, with the exception being that PlayStation 4 and Xbox One players can not play with each other (Tucker, 2018). Cross-platform play is more common today with platforms because mobilephones have enough power to run popular triple-A titles and the ability to go online almost anywhere in the world. So it is crucial for developers to try and gain a wide audience among users on many platforms, as the games are easier to work on different platforms than in the past. Yet, there are not many works that tackle the subject of how to start a cross-platform project and what to take in to account before doing so. This thesis aims to act as a starting point for new developers who want to start doing cross-platform development. It could help with choosing the right platform and a game engine for it, what sort of limitations and problems a developer might encounter when doing work for the chosen platform and how to test a project on that platform. Also, an important consideration for modern developers is to find the minimum testing environment for most of the popular platforms today and naturally the platform they intend to develop their game to. This thesis, however, does not cover virtual reality development as the virtual reality market is still very small compared to traditional games. In addition,virtual reality development on its own brings so many new ideas and problems to the table that covering them is not within the scope of this thesis. When developing for VR you need to take in to account the limitations and specifications of the platform as well as the VR headset. In addition a developer must understand the intensity and effects of VR gameplay to the player, so that the gaming experience does not cause unwanted nausea or other negative effects.

This thesis is structured as follows:

Chapter 1 introduces the reader to the subject and gives an overview of the thesis structure

Chapter 2 focuses mostly on the theory of cross-platform game development, acquiring information from multiple sources and covering the most common game engines that

support cross-platform development. In addition this chapter covers the minimum testing environments for the most common gaming platforms.

In Chapter 3 the online questionnaire and its answers are analyzed and summarized. With some questions, the answers are also compared to results that were found in the previous chapter of this thesis.

Chapter 4 is about the author's internship at a game company, where the task was to port a mobile game made with Unity to use the WebGL platform. As porting is also important when it comes to cross-platform development, this was included in the thesis as well. It mostly covers issues that the author encountered when the game, that was already developed so far with only mobile in mind, was ported to a completely different platform. In addition it covers the problems the author had encountered with the WebGL platform in general.

2 WHAT IS CROSS-PLATFORM DEVELOPMENT

Cross-platform development means that a computer software is developed and implemented for multiple computing platforms. It can be divided into two types, one that requires individual development and building for each supported platform separately, and the other type can be directly used on any platform without specific preparation. (Wikipedia) For example, when building Unity applications for different mobile devices, the same project usually works without issues, when a developer can just choose the correct build target to match the target platform.

The most commonly known cross-platform application is a web browser. Browsers are built to work on every PC workstation, mobile phone, game console or even a television. They usually render pages to look the same regardless of platform, with the exception of specific desktop and mobile page layouts. (PCMag, 2018)

One of the features that cross-platform development enables is cross-platform play. This means that the game has an online gaming component that allows gamers on different platforms to play the same game and interact with eachother simultaneously. Cross-platform play is technically possible with today's computer and game console hardware being compatible with each other, but it is generally impeded by two main factors. One factor is the difference in control schemes between PC and consoles. The keyboard-and-mouse controls usually give PC players an unfair advantage over the console players, but this of course depends also on the type of game being played. The second factor relates to the online services provided by the console manufacturers being so closed. They are generally meant to provide a safe and consistent environment for online gaming and cross-platform play depends on the provider to enable it for games. At present, with the game consoles of the current generation, cross-platform play can usually be found between Microsoft's Xbox One and Windows or other PC or separately between Sony's PlayStation 4 and PC. Microsoft has suggested further cross-platform play with other platforms, including Sony's PlayStation, the Nintendo Switch and mobile devices. (Wikipedia)

2.1 Development for multiple platforms

When planning to start development for multiple platforms, it is advised to choose a game engine that is compatible with the chosen platforms and can compile to them. This should usually be decided early on in the development process as porting games may prove to be difficult and introduce new requirements surprisingly, that may halt the development completely or slow it down immensely. Few recent examples of porting a game, in this case for a Linux operating system are *Gauntlet* and *Divinity: Original Sin*. They encountered numerous problems for various reasons, even though both studios behind these games are very familiar with Windows operating systems, but cross-platform was very new to them.

In addition there are few key points to take in to account when developing for multiple platforms simultaneously:

- Aspect ratio – This is rarely standard and can vary from 4:3 to 21:9
- Screen size – smartphones, tablets, PC's and televisions, the size may vary from 4 inches to 50+ inches.

Usually game engines offer a way to emulate different size screens so these issues can be solved fairly easy.

- Audio and video codecs – The supported codecs differ depending on the platform
- Platform-specific social features – These interfaces and functionalities are usually very different depending on the platform

These issues affect all platforms generally, but there are of course issues that are platform-specific and require certain attention when developing to them. (Justice, 2015)

2.1.1 Desktop

Because of Valve's SteamPlay, it has become more common to release games for the three most used desktop platforms, rather than just Windows. The input methods are almost identical on all platforms, but developers must pay attention to compatibility of third party plugins and frameworks. Apart from those, the key differences are:

- Filesystems differ by platform, also the path syntax is unique for Windows.
    - Windows – case-insensitive, home directory: C:\Users\example\
    - OS X – case-insensitive, home directory: /Users/example/
    - Ubuntu/SteamOS – case-sensitive, home directory: /home/example/
- GPU Drivers – Intel, AMD and nVidia drivers differ by platform. The compatibility and functionality of these should be tested early on in development because if they are tested too late, this might lead in to problems such as the game might only work with nVidia cards on Linux.
- Correct game integration into the platforms window managers (minimize, fullscreen.)

(Justice, 2015)

2.1.2 Web

Currently web games are gaining more popularity and they are much more complex than in the past. Web games are high quality and visually almost as stunning as games on other platforms. Web is also a great platform for distributing games. In the past most web games were done using Flash, but due to performance increase in JavaScript and new APIs games can be made more demanding and ran with HTML5-based browsers. (Mozilla, 2018)

Web of course also has limitations, for instance the only working programming language is Javascript, so game code usually must be written in JS or converted using a compiler like Emscripten, which is at least used in Unity when building for WebGL. Emscripten takes the native C# language of Unity, converts it to C++ and then further converts it to asm.js, which is JavaScript. (Mozilla, 2018) In addition threading does not work when building for Web, because JavaScript does not support threading. (Unity, 2018)

These among other problems are gone through more thoroughly in chapter four, where a Unity project that has already been developed for mobile devices is ported to use the Unity WebGL platform.

2.1.3 Mobile

For mobile game development, there are different input methods that must be brought to match the ones on desktop. There are also numerous different screen types that must be taken in to account. Some key points to take in to account are:

- Android and iOS filesystems are case-sensitive.
- Access rights for the filesystems are very different depending on the platform.

(Justice, 2015)

Most of the other major problems in development have something to do with Windows phones and their system, but Microsoft has stated that they are discontinuing the platform. (Wikipedia)

2.1.4 Consoles

Most if not everything from console devkits is under NDA, so not much information regarding specific consoles is known. Current generation consoles do have a very similar hardware compared to desktops and the app-ecosystem is similar to mobile devices. Consoles as a platform have these differences:

- They support different shader languages
    - Xbox One: HLSL (Direct X 11)
    - PlayStation 4: Playstation Shader Language (GNM, GNMX)

If the engine in use is Unity, GLSL shaders cannot be cross-compiled.

- Games usually run in a sandbox, so it limits the filesystem acess rights.
- Some platforms have very unique peripherals, like the Wii Remote, for example.

(Justice, 2015)

2.2 Selecting a cross-platform game engine

Finding the right engine for a cross-platform game project can be the key to a successful development process and deployment. With modern day standards releasing a game for multiple platforms at the same time or at least in a short time window has become typical. With this in mind, a cross-platform game engine offers many advantages. The engines that will be compared are Unity, Corona, Cocos2D JS, Appcelerator Titanium. (Weiss, 2014)

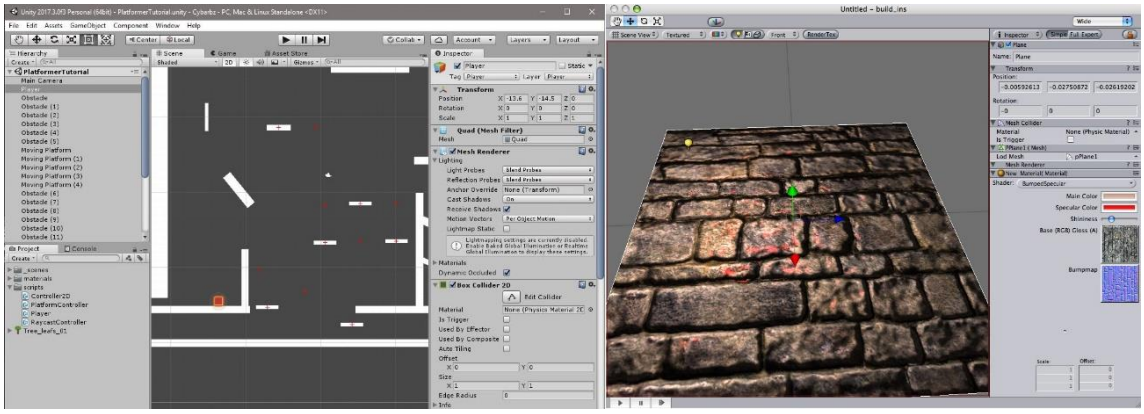
In addition Unreal Engine is one of the biggest cross-platform engines out there at the moment. However the platforms it supports and how games developed with it perform are very similar to Unity. (Slant, 2018) Unity is more popular among developers and hobbyists, mostly because of it's lower learning curve and C# support. (TNW Deals, 2016) Because of the many similarities to Unity, Unreal was left out of this comparison. However developers, especially those that have experience with C++ rather than C# should still have a look at Unreal Engine.

2.2.1 Unity

Unity is a closed-source, cross-platform game engine. Objects are manipulated in a 3D space and various components and scripts are attached to them. Even if the game is in 2D, the development still takes place in a 3D environment. Scripts are recommended to be written in C#, but it has the support for Boo and Unityscript as well.

Launching Unity for the first time can feel overwhelming as there is much to learn. It can take roughly 8-12 hours to become familiar enough with the user interface to start developing a game, so there is a noticeable learning curve.

Unity was released in 2005 and the interface still looks very similar as seen in picture 1. Some of the repetitive tasks in Unity development feel like busy work. Adding audio sources, updating prefabs and importing assets are examples of features that feel old and outdated. (Weiss, 2014)



Picture 1. A comparison between a modern Unity version (left) and the original Unity 1 (right) both still look very similar in terms of user interface and functionality.

After creating a game with Unity the deployment is very simple. With a few clicks, the game can be exported to mobile, desktop or web. The web version used to require a separate Unity player app to be installed, but Unity has moved on to use WebGL which does not require any additional players to be installed. With correct licenses, the game can also be deployed to consoles. Unity gives a free version that does not have all the features of the paid version, like video playback, but it allows a game to be deployed to most platforms, with the Unity logo displayed on launch. (Weiss, 2014) At the time of writing the Unity Plus version is 35 $ / month and Unity Pro is 125 $ / month. In addition Unity offers an enterprise version, which is a custom, tailored solution for organizations that have 21 or more employees. (Unity, 2018)

2.2.2 Corona

Corona is a closed-source 2D game simulator and cloud-build application. The game code is written in Lua scripts and played back in the Corona simulator. Corona has a wide variety of skins, resolutions, and ratios. When deploying a game, it builds it in the cloud and delivers a iOS or Android game client.

The development language is easy. Adding a physics body, for example, only takes one line of code. The learning curve for becoming familiar with the platform is approximately 2-4 hours. The simulator is responsive, quick, and uses the development computers resources sparingly. With the simulator and a code editor open, the code can just be saved with the editor and the simulator instantly loads the changes that have been made.

The greatest shortcoming for Corona is the limited deployment options. It only supports mobile platforms like iOS, Android, Kindle and Nook. Cloud-building can also be a bit tricky. When testing the game on a device it can take a few minutes before the new build can be loaded on to the device from the cloud, so especially in the last testing phases of development this can be tedious. (Weiss, 2014)

In the past, Corona has been free for companies with a revenue limit of 100 000 $, and paid from there on. (Weiss, 2014) This has changed and now the core of Corona is completely free for everyone, with a marketplace similar to Unity's asset store where users can distribute and sell their assets. (Corona, 2018)

2.2.3 Cocos2D JS

Cocos2D JS is a cross-platform, open-source, free game development SDK. It is essentially a combination of two popular open-source projects: Cocos2D X for mobile and desktop and Cocos 2D HTML5 for web. It currently supports 2D and 2.5D, but there are plans to add support for 3D development.

The coding is done entirely in JavaScript on native platforms like mobile and desktop. The JavaScript is the bound to native C++ objects, granting maximum speed without having to write any native code. Web platforms run pure JavaScript and render using Canvas or WebGL, so there is no need to install separate player applications.

The simplest way to start developing with Cocos2D JS is using the HTML5 platform. With a browser window open and using a code editor to save the JavaScript. Refreshing the browser window then displays the game. When the game needs to be deployed on a native platform an application like Xcode, Visual Studio or Eclipse is needed.

Cocos2D JS games can currently be deployed to iOS, Android, Blackberry, Windows Phone, Mac, Linux and HTML. This is a very wide range of options to choose from. Cocos2D JS is also 100% open-source and liberally MIT licensed, which means it is a free software license that is allowed to be used and modified freely, even in closed source projects. The license text however needs to be visible in the source code. (Wikipedia) Developers can learn from reading the code and in addition editing and modifying the code to match needed requirements is simple. (Weiss, 2014)

2.2.4 Appcelerator Titanium

Titanium is a cross-platform, open-source app development kit and Eclipse-based IDE. The code is written in JavaScript and then run natively, not only in WebView. Titanium Studio makes it possible to develop, test and deploy to mobile and web platforms.

If developing a 2D game, there is a Platino Game Engine, which is open-source, but not a free SDK, that can be added in to the Titanium stack. The documentation for Platino is very limited and missing some key parts, so learning and using it might prove to be troublesome. The physics engine is also very archaic. All physics bodies and sprites must be synchronized manually using a non-JavaScript, C-like API, which is very cumbersome. Though build times on Titanium are very short as it uses a prebuilt SDK, the developed game can be ran on a simulator or a device quickly. Titanium apps can be deployed to iOS, Android, Blackberry and HTML5 for free. The Platino Engine as well as other tools that might be needed are available for a yearly subscription fee. (Weiss, 2014)

2.2.5 Conclusion of game engine comparison

Based on the options that were compared here, it is recommended that if the developed game needs 3D support Unity is the best choice. If only 2D support is needed, the games are simple enough and the target platforms are mobile then Corona. Cocos2D JS is a good choice when the game needs to be deployd to many platforms. Because it is open-source, it might attract certain developers more.(Weiss, 2014)

2.3 Cross-Platform Playtesting

When developing for multiple platforms, naturally the game should also be tested on each desired platform. This could get very expensive, especially as an indie developer, buying all the required systems might not be possible in the beginning. If not developing a native cross-platform multiplayer game, the only thing that should be considered first is getting the game to launch on each platform properly. If using Unity, Java or other cross-platform environments, it can be assumed that the cross-compilation will keep the game logic intact and consistent across all the chosen platforms, unless the game code

has something exotic and out of the ordinary in it. Of course before releasing the game on any platform it is necessary to test it on each targeted platform. (Justice, 2015)

2.3.1 Desktop minimal testing environment

Desktop platforms are all somewhat similar. The same machine that is used for development can usually be used for testing the game. On a PC it is possible to install Windows and Linux in a dual boot configuration, this enables testing of the game on a case-sensitive and case-insensitive system. It should also be made sure that the game runs on Windows and Linux Window Managers correctly. For example minimizing and fullscreen should work without issues.

The desktop platforms offer similar periphery, supporting a keyboard and mouse combination as well as controller input. A Xbox 360 controller is recommended for testing, due to having plug and play support on Windows and Linux systems. It also works on OS X, but requires a separate driver to be installed. (Justice, 2015) It is difficult to specify the minimum system requirements for a PC game, as it depends so much on the type of game being developed and different PC platforms can vary in power greatly. Game engines have minimum system requirement specifications that can be used as a reference to find out the minimum requirements of the game. For example Unity requires a modern supported Operating System, a GPU with DirectX 10 support and a CPU with SSE2 support to run a game. (Unity, 2018) To see how well a game actually runs on different builds the only way is to just try the game on older machines and see how well it runs.

2.3.2 Mobile minimal testing environment

Mobile devices all share the same touch-based periphery and sensors, but their operating systems differ greatly. Filesystem access differs between iOS, Android and Windows Phone, so all of the file operations should be tested on each desired platform before publishing. In order to get started with mobile testing it is recommended to get a cheaper tablet that runs with a qualcomm quad-core cpu and 1GB of RAM. (Justice, 2015) This is to ensure that the game will run smoothly on more high end devices as well, if it can do so on weaker hardware. The operating system version usually does not

matter, if it meets the minimum required version. Unity games for example require an Android OS version of 4.1 or later to run a game. (Unity, 2018)

2.3.3 Console minimal testing environment

When developing for consoles, it is usually adequate to have a computer and a controller. (Justice, 2015) Of course if the platform uses non-standard peripherals like the Nintendo Switch, it might cause problems if the game can't be tested on the actual platform.

A common topic is the price of console devkits. Sony and Nintendo usually do not reveal their pricing, but the Xbox One Devkit is available for free, if Microsoft approves the games idea. Console manufacturers also expect developers to sign a contract along with a NDA. These documents should be read carefully, they might include information on how long the devkit is available for the developer and other important information about the game's release and publishing. (Justice, 2015)

3 SURVEY RESULTS

An anonymous Google Forms survey about cross-platform game development was made with a total of 15 questions. The questionnaire was shared on the IGDA Finland Facebook page and got a total of ten answers from people with different backgrounds and experience levels. In this chapter the questions and their respective answers will be gone through and analysed.

The first two questions seen in figure 1 and 2 were about the respondents work position and experience level in the game industry. Most of the respondents were programmers, there were also a couple of managers and a designer. Unfortunately there were no graphical artists that answered so there are no answers from the graphic point of view. thirty percent of the respondents had not worked in the gaming industry at all and the rest had at least one year of experience each, with some having over five and one respondent even having over ten years of experience.

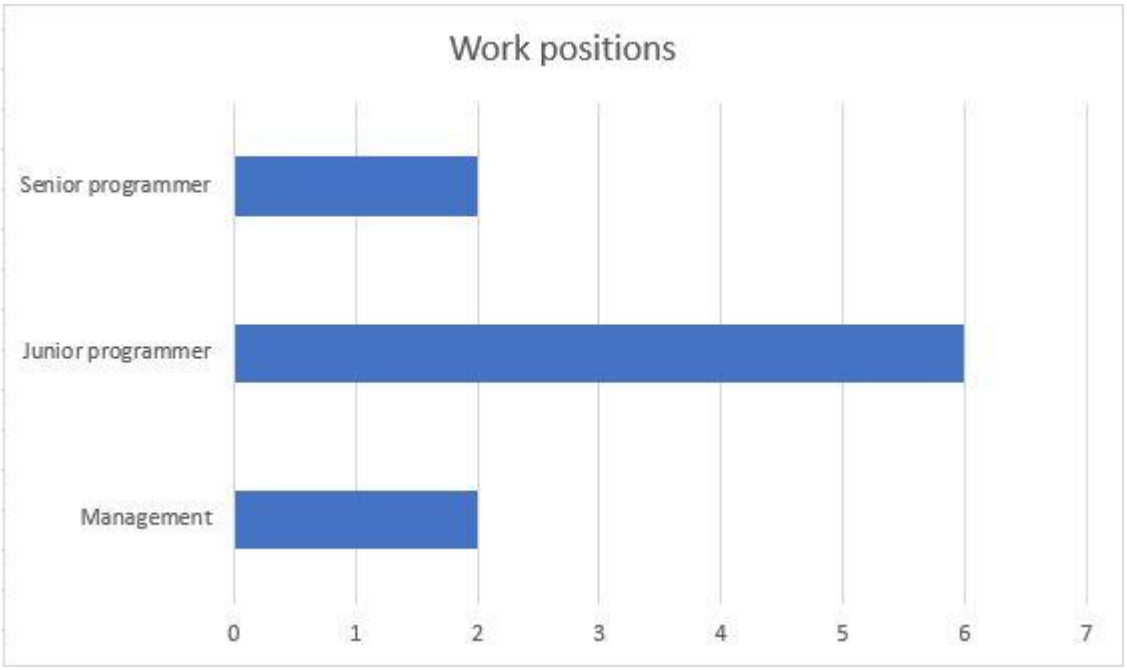


Figure 1. The work positions of the respondents.

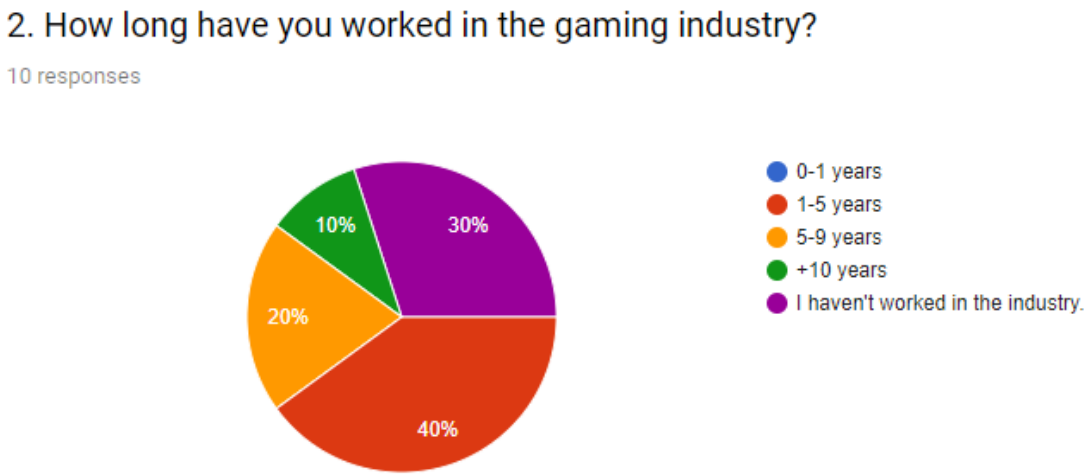


Figure 2. The respondents working experience in the gaming industry.

The third question seen in figure 3 was about experience with cross-platform development specifically. Only one respondent had not worked with it before. So the questionnaire in that sense was a success, because it reached the right target audience, which was developers who have experience with cross-platform projects.

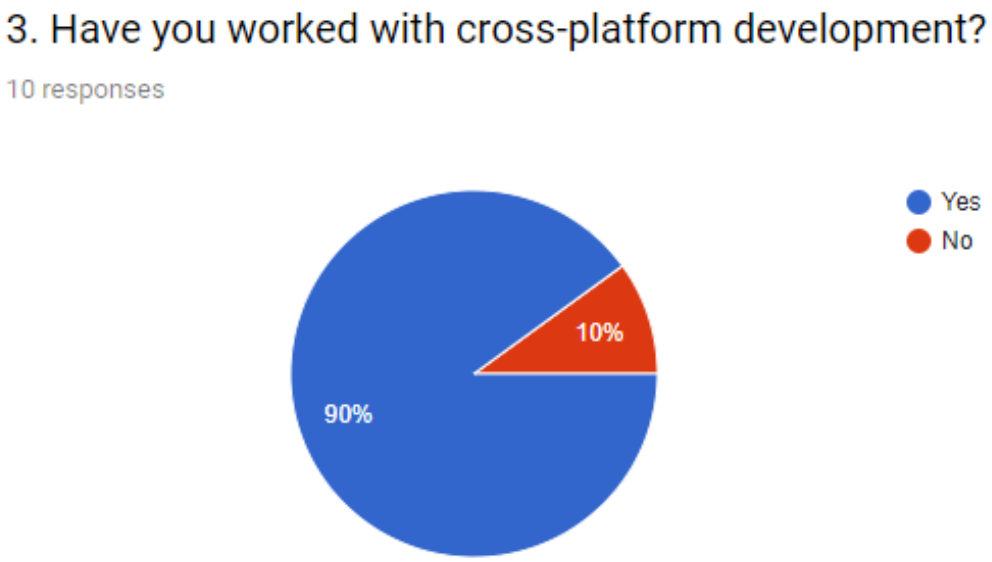


Figure 3. The respondents experience with cross-platform development.

The next questions in figures 4 and 5 were about platforms the respondents have worked with, when developing cross-platform solutions and game engines that were used. Android and iOS were the most popular platforms and right after that Windows and other PC operating systems. One respondent noted that it could have been better to split the most common PC operating systems, rather than clumping Mac, Linux and other in to one, which would have been better, because Mac and Linux are still different enough. Other than that the platforms were distinct enough for the purposes of this survey. From game engines Unity was the most popular and after that a plethora of other engines, mostly mobile oriented. Some had used an in-house engine.

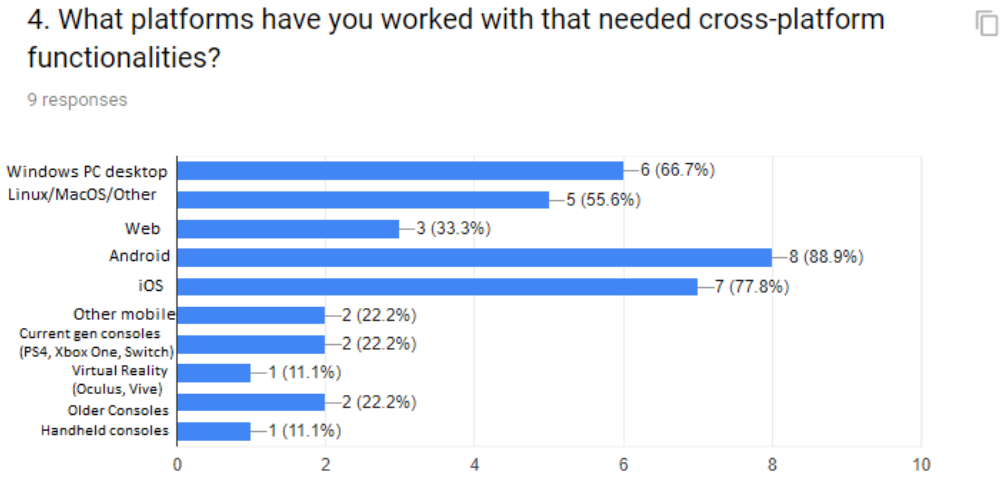


Figure 4. The distribution of different platforms the respondents have done cross-platform work with.

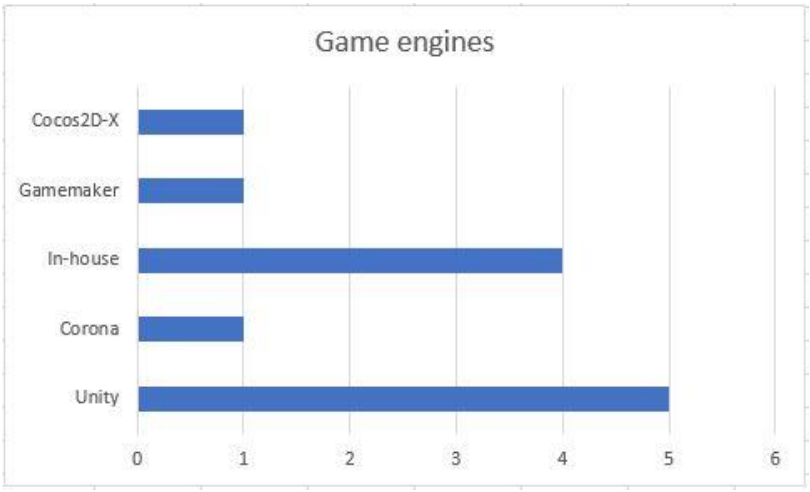


Figure 5. The distribution of different game engines that the respondents had used for development.

Next up the respondents were asked to answer in their own words what was most difficult to do when developing for cross-platform. Not everyone answered this, but the answers given in figure 6 were mostly on point with how cross-platform development was analysed in the last chapter. Many considered testing is tedious and demands lots of resources. Some platforms had almost no users so developing for them felt pointless and then the general platform differences that had to be taken in to account.

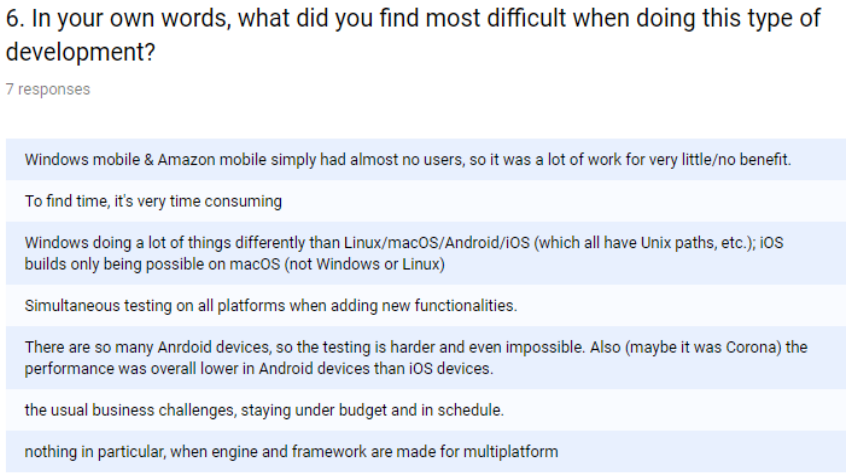


Figure 6. What the respondents considered difficult when developing for cross-platform.

The next question in figure 7 asked the respondents did they find something easier than they initially expected. The answers were all very different. The modern game development tools were praised for being much better than in the past and some platforms have easier ways develop and publish games than others.

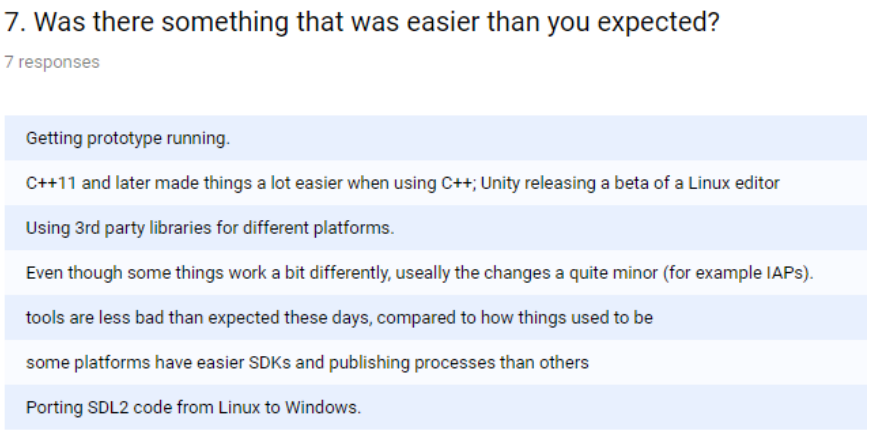


Figure 7. What the respondents thought was easier than expected.

Next questions in figures 8 and 9 were about how relevant the respondents think cross-platform development is in the current gaming industry and what sort of benefits do they see in it. The general consensus seemed to be that cross-platform development is very relevant and important to reach a larger audience. One respondent even noted that having the game on multiple platforms is crucial for indie developers to get enough sales to keep up a reliable profit.

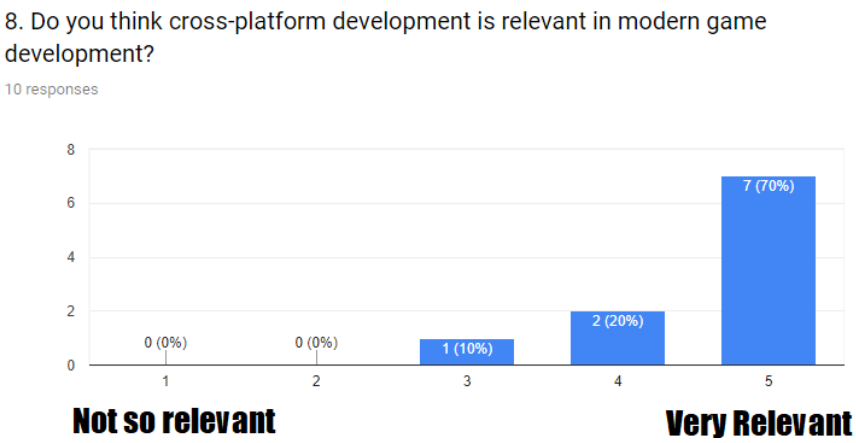


Figure 8. How relevant the respondents think cross-platform development is.

From a programming point of view, one respondent also noted that when doing cross-platform development the coding requires more generic code, which is usually easier to maintain than platform specific code.

Figure 9 The benefits the respondents think cross-platform development has.

After answering what benefits the respondents thought cross-platform development would have, they were asked in figure 10 to answer what sort of problems it might bring. Most notably having multiple platforms brings more work for developers and testers. Testing will take time as the game must be tested on all the platforms. One respondent also noted that there might be a big skill gap involved in games where players from different platforms play together, for example in shooting games a mouse is more accurate than a controller. Also noted was that exclusivity to a certain platform could work as a marketing tool for publishers.

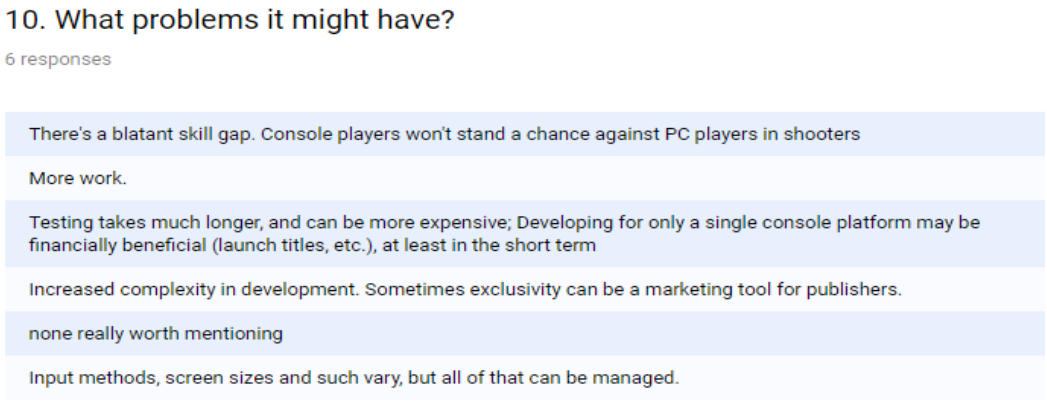


Figure 10. Problems of cross-platform development.

When asked about their experience on playing cross-platform games in figure 11, sixty percent of the respondents had played something that had a cross-platform element, unfortunately when asked what games and on what platform they played on, most did not answer.

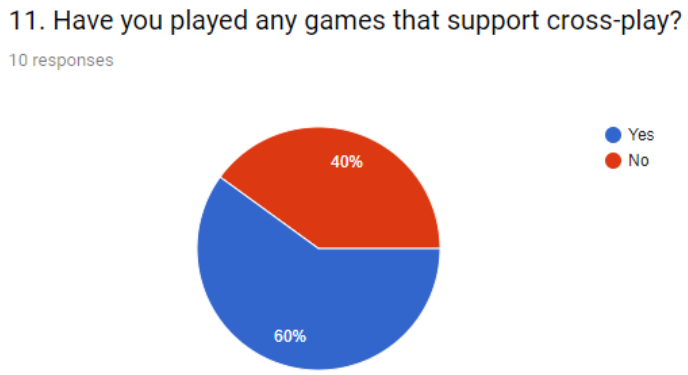


Figure 11 The respondents experience with playing cross-platform games.

Two games that appeared in the answers in figure 12 were *Minecraft* and *Hearthstone*. *Minecraft* is available on many consoles and PC and *Hearthstone* is available on PC, Android and iOS. Both of these games have the possibility to play together with other people, regardless of their chosen platform.

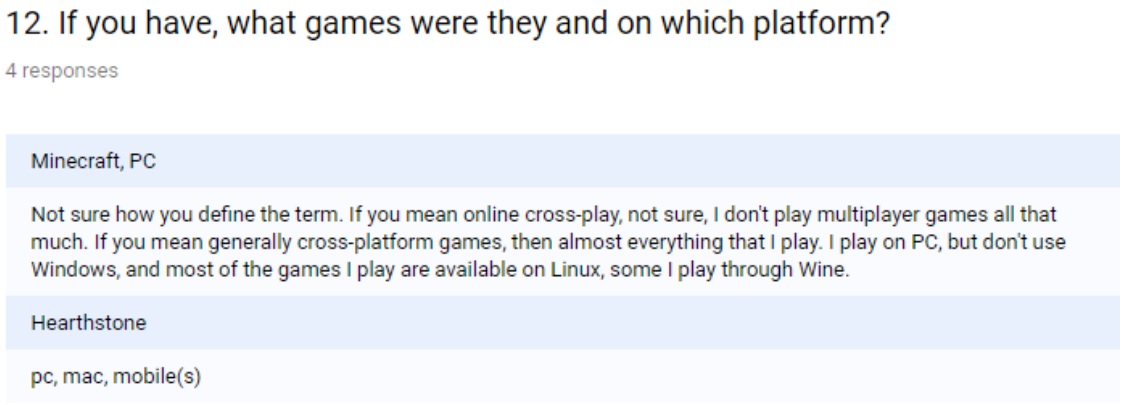


Figure 12 What games the respondents played that had cross-platform elements.

The three questions that were asked next in figures 13, 14 and 15 were about the cross-platform features of the games played. Firstly did the respondents notice any problems that might have been caused by the cross-platform element. Biggest problems noted were input related with some graphical issues too. Also as noted by one respondent, some issues felt like they were created when porting the game, rather than developing it for multiple platforms from the beginning.

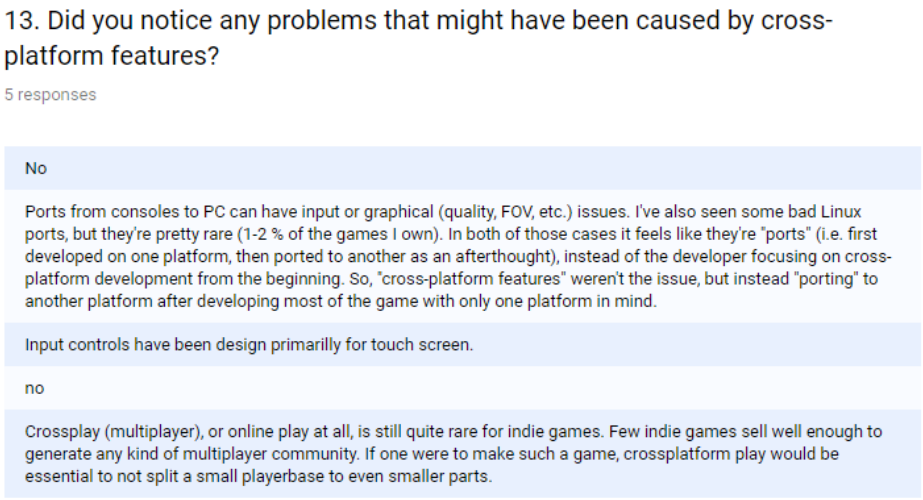


Figure 13. What problems the respondents had when playing cross-platform games.

When asked did the respondents think the games they played benefitted from the cross-platform features the general consensus was very positive.

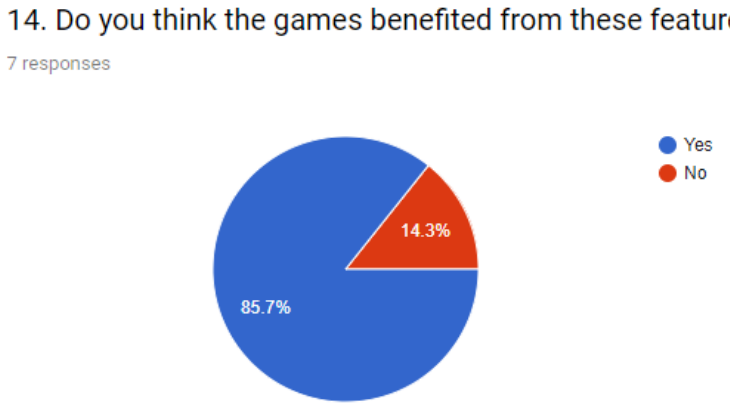


Figure 14. How many respondents think games benefit from cross-platform features

Not many answered the question about what features they thought worked well. This was some what expected, because usually the cross-platform element is not so visible in games, unless they are multiplayer games where players from different platforms play together. There still were some key points, like the ability to share save files through different platforms. For example the remaster of Final Fantasy X | X-2 has a cross-platform save functionality across PlayStation 3, PlayStation 4 and PlayStation Vita.

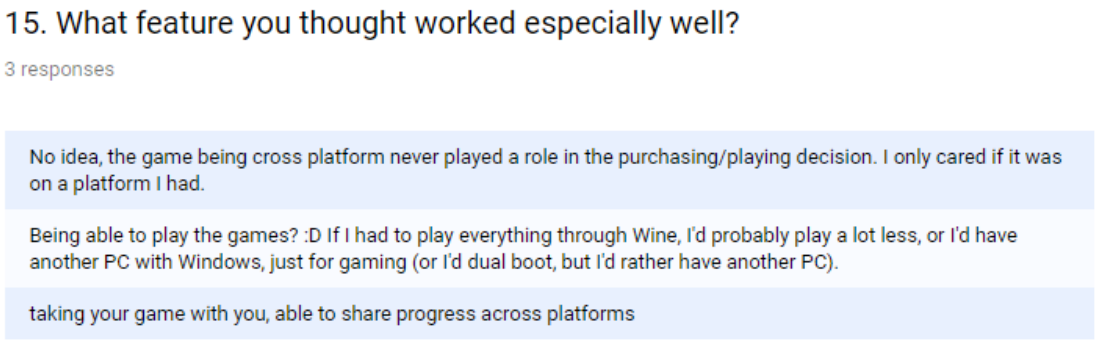


Figure 15. What features the respondents thought worked well.

In conclusion the survey was a success. Even though only ten people in total answered, the answers were broad and really reflected the thoughts about cross-platform development in the gaming industry.

4 PORTING A UNITY MOBILE GAME TO WEBGL

The work was done during a seven week internship at a Turku-based game company that specializes in mobile games. The company wanted one of their mobile games to be ported to the WebGL platform. The port did not need to incorporate all the features of the mobile version, rather just work as a starting point for their developers to continue the work. The goal of this task was to research and find out issues that might come from porting the game to the WebGL platform and list them out with possible workarounds.

4.1 Introduction of the case

The game is a mobile game developed primarily to Android and iOS. It was made using Unity, so there was no need to switch game engines, as Unity natively supports building to the WebGL platform. This sped up the process greatly as the games features only needed to be made compatible with WebGL, rather than creating them from scratch on another game engine. The game did not need to fully work on WebGL. The main point of the work was to see how easy it is to port the game for the WebGL platform and find out all possible problems and compatibility issues that come up. This would then serve as a basis for continued development.

4.2 Issues and workflow with WebGL

The Unity WebGL platform proved to be quite unstable and building for it in the beginning was near impossible. Some features needed to be completely removed, like the Google Firebase analytics. With it enabled the editor would not compile the game at all, rather just give a generic Firebase error. Also when debugging a WebGL project, most of the errors do not come up in the editor, so the build needs to be compiled, opened in a browser and then debugged with the browsers development console. This is very time consuming as it halts all work because the engine compiles the code and cannot be interacted with at that time. One way to do this is to add *Debug.Log* calls to the code in places that might have issues in execution.

The project can also be built as a development build, that keeps the asm.js script in a format that is readable. The downside of the development builds is that they are very

large, because the content is not minified like in the regular build. (Unity, 2018) The readable asm.js helped a bit, but usually the problems were still easier to find in the original c# scripts. One of the most effective ways to debug a WebGL build was using the browsers development console window and having a development build with exeptions enabled. (Gonzalez, 2016)

There is also an option to build the project using a pre-built engine, when using a development build. This speeds up the process of rebuilding a project approximately 30 – 40 %, this is because when using this option Unity rebuilds only the managed code and then links it dynamically with the pre-build Unity engine. This method is only suitable for development, because it always creates the project with unstripped engine code. The dynamic linking overhead also degrades performance, so the game runs slower than it would as a regular build (Unity, 2018)

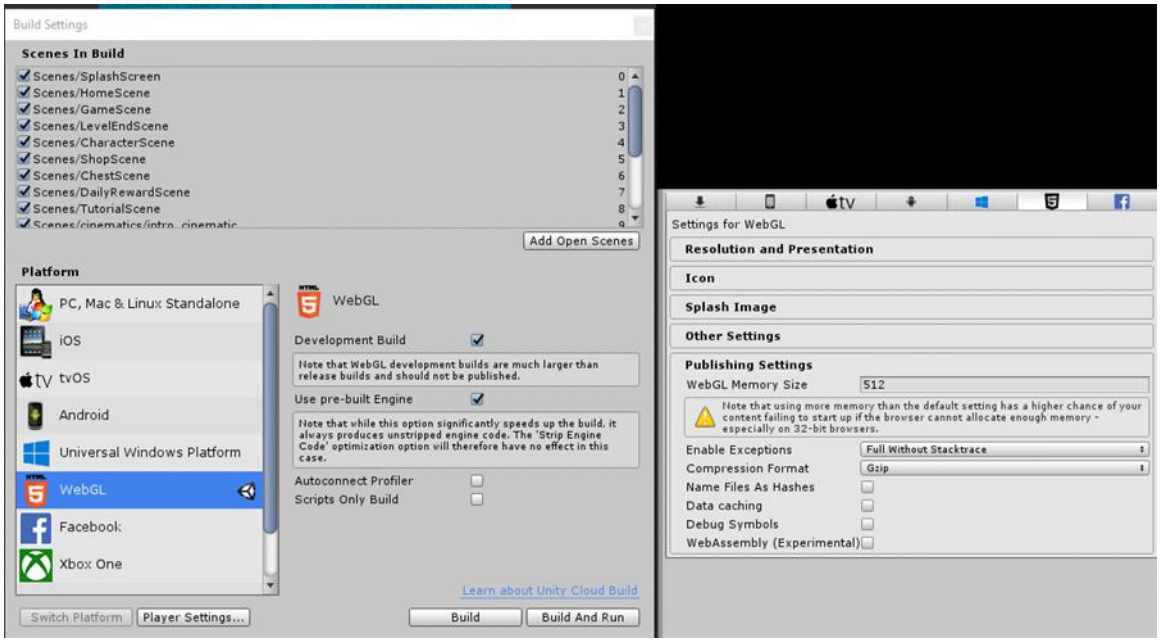
WebGL does not support Unitys threading features at all, so everything that uses that must be replaced. Easiest way of doing this is to search the whole solution for code that uses threading. Then adding a pre-processor directive for it, so that the code is not executed if the platform in use is WebGL.

Program 1 Pre-processor directive that excludes the code if WebGL is used.

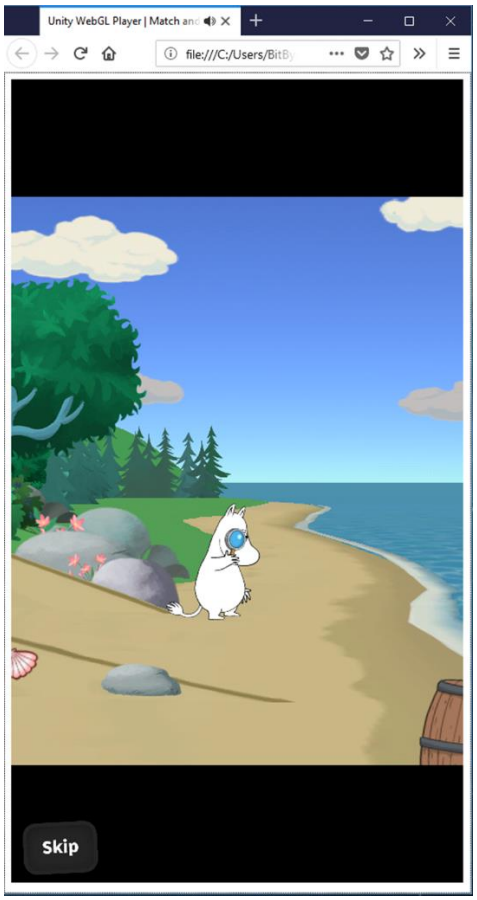
```
#if !UNITY_WEBGL

using System.Threading;

#endif
```

Then after excluding threading from the solution it needs to be checked for errors caused by the directive and fixed by finding a solution that does not use threading features. (Gonzalez, 2016)

In the beginning the development builds would not launch in the browser, but rather after loading for over two minutes they gave a not enough allocated memory -error. No matter how much memory was allocated to the build, the error stayed the same. This was fixed by disabling the strip engine code option from the player settings before building the project. The settings in picture 2 are settings that finally allowed the development build to run without major issues. In addition to these, the strip engine code was also kept disabled. In picture 3 the games intro cinematic is working and running on Firefox and the game is fully playable.



Picture 2. The settings for a working development build.



Picture 3. The games intro cinematic running on Firefox

In the beginning working was tedious and most of the time went in to researching how WebGL as a platform works and what functionalities do not work with it. After getting used to WebGL as a platform and the project itself the general workflow started to look something like this:

- Make a new build for the project
- Open the solution in the browser window
- Check developer console for possible errors
- Find where the errors point to in the actual project scripts
- Make a fix or workaround in that code
- Repeat

This was still very time consuming as the project was still so big that building could take somewhere from 10 minutes up to 25 minutes. For some reason the building also froze the workstation that the work was done on and the computer needed to be hard reset several times. This also ate a big portion of working time. And of course when Unity was doing the building, nothing else could be done with the project in the meantime, so most of the time went in to waiting for the project to build.

4.3 Conclusion of the porting process

In the end, the project was working and playable with an internet browser. The core gameplay functions worked without any issues and the game was as enjoyable as the mobile counterpart. Some functions did not work, because the WebGL as a platform did not support them in their current state. These included in-game purchases, user analytics, cloudsaving and advertisement related events. These issues came up early in the development process and it was decided to ignore them and try focusing on playability on the WebGL platform. When that is perfected the companys own developers would start to work with these found issues.

So in conclusion the assigned work was finished and met all goals set in the beginning. The game was playable, it did not suffer from performance or other issues when playing and the work done will act as a great point to continue with getting the game fully ported

to the WebGL platform. Table 1 shows the most complicated problems encountered during development and their solutions.

Problem	**Solution**
Threading not supported by WebGL.	Exclude System.Threading from the solution using a pre-processor directive and then fix all parts of the code dependent of it.
Project not compiling with Google Firebase Analytics.	No WebGL support, or atleast the project in its current state wasn't able to build with Firebase enabled. It was removed from the project completely and the company will find an alternative analytics system to replace it.
Long compiling time.	The compiling time for the full project is long, this can be shortened significantly by using a development build with pre-built engine.
Memory allocation error when running the development build on the browser.	Disable the strip engine code from the build settings and set the enable exeptions option to full without stacktrace.

Table 1. The most complicated problems encountered in the porting process.

Some of the issues faced here could have been avoided, if the game would have been developed with other platforms than mobile in mind from the beginning. These problems somewhat slowed down the porting process, but they were not at all insurmountable.

5 CONCLUSION

The purpose of this thesis was to research methods of cross-platform game development, find out what the industry professionals have to say about the concept of cross-platform development and have they actually used it when developing their games. Supporting this research are the author's personal experiences on working with porting a mobile game to WebGL.

The results of this thesis can be used as a starting point for new developers that want to start developing games to multiple platforms. It handles issues like choosing a game engine and setting up a minimum testing environment, which are crucial for new developers. However, the thesis does not include VR development, which still has a relatively small market in the game industry, but is growing and gaining popularity rapidly.

Even though many game engines were compared in this thesis, the ultimate choice is in the hands of the developer. The developer should choose an engine they feel the most comfortable with, which also supports the key features of the game and is compatible with all desirable platforms.

The results of the questionnaire indicate that cross-platform development is looked upon positively and encouraged for indie developers to reach a greater target audience.

The practical part shows how many issues porting a game rather than developing it for multiple platforms from the ground up can bring. It also serves as a guide to developers who want to port their Unity-based games and application to use the WebGL platform. WebGL is a suitable platform for game development today, but it's limitations should be considered before starting a project. When trying to port an existing game to support WebGL developers are bound to run in to errors and issues, if they haven't focused development to support WebGL in the beginning.

REFERENCES

Corona. 22.3.2018. Referenced 22.3.2018 https://coronalabs.com/

Gonzalez, L. 2016. Unity 5 and WebGL porting guide. Referenced 15.5.2018. https://www.gamasutra.com/blogs/LeandroGonzalez/20160719/277368/Unity_5_and_WebGL_porting_guide.php

Justice, B. 2015. How to Develop for Cross-Platform. Referenced 27.2.2018. https://www.gamasutra.com/blogs/BenjaminJustice/20151029/257691/How_to_Develop_for_CrossPlatform.php

Mozilla 4.6.2018. Referenced 4.6.2018 Introduction to game development for the Web. https://developer.mozilla.org/en-US/docs/Games/Introduction

PCMag, 4.6.2018. Definition of: cross platform Referenced 4.6.2018. https://www.pcmag.com/encyclopedia/term/40495/cross-platform#fbid=aHfb3ldkqPq

Slant, 4.6.2018. Referenced 4.6.2018 https://www.slant.co/versus/1047/5128/~unity_vs_unreal-engine-4

TNW Deals, 24.3.2016 This engine is dominating the gaming industry right now. Referenced 4.6.2018. https://thenextweb.com/gaming/2016/03/24/engine-dominating-gaming-industry-right-now/

Tucker, J. 25.4.2018. Epic's Sweeney touts a cross-platform play future for the games industry. Referenced 16.5.2018. https://www.mcvuk.com/development/epics-sweeney-touts-a-cross-platform-play-future-for-the-games-industry

Unity. 22.3.2018. Referenced 22.3.2018 https://store.unity.com/

Unity. 15.5.2018. Referenced 15.5.2018 https://docs.unity3d.com/Manual/webgl-building.html

Unity 4.6.2018. Referenced 4.6.2018 https://docs.unity3d.com/Manual/webgl-gettingstarted.html

Unity. 4.6.2018. Referenced 4.6.2018 https://unity3d.com/unity/system-requirements

Weiss, N. 14.5.2014. Selecting a Cross-platform Game Engine. Referenced 18.4.2018. http://www.binpress.com/blog/2014/05/14/selecting-cross-platform-game-engine/

Wikipedia (1). Cross-platform. Referenced 27.2.2018. https://en.wikipedia.org/wiki/Cross-platform

Wikipedia (2). Cross-platform play. Referenced 27.2.2018. https://en.wikipedia.org/wiki/Cross-platform_play

Wikipedia (3). MIT License. Referenced 4.6.2018. https://en.wikipedia.org/wiki/MIT_License

Wikipedia (4). Windows Phone. Referenced 8.3.2018. https://en.wikipedia.org/wiki/Windows_Phone