Faculty of Science and Technology

BSc (Hons) Games Technology

May 2017

Evaluating the use of the Unity engine for developing 2D mobile games in consideration of start-up/student developers

By

Jack Brett

DISSERTATION DECLARATION

This Dissertation/Project Report is submitted in partial fulfilment of the requirements for an honours degree at Bournemouth University. I declare that this Dissertation/ Project Report is my own work and that it does not contravene any academic offence as specified in the University's regulations.

Retention

I agree that, should the University wish to retain it for reference purposes, a copy of my Dissertation/Project Report may be held by Bournemouth University normally for a period of 3 academic years. I understand that my Dissertation/Project Report may be destroyed once the retention period has expired. I am also aware that the University does not guarantee to retain this Dissertation/Project Report for any length of time (if at all) and that I have been advised to retain a copy for my future reference.

Confidentiality

I confirm that this Dissertation/Project Report does not contain information of a commercial or confidential nature or include personal information other than that which would normally be in the public domain unless the relevant permissions have been obtained. In particular any information which identifies a particular individual's religious or political beliefs, information relating to their health, ethnicity, criminal history or personal life has been anonymised unless permission for its publication has been granted from the person to whom it relates.

Copyright

The copyright for this dissertation remains with me.

Requests for Information

I agree that this Dissertation/Project Report may be made available as the result of a request for information under the Freedom of Information Act.

Signed:

Name: Jack Brett

Date:

Programme: BSc GT

Acknowledgements

Thank you to Alain Simons, my primary supervisor. His continuing support and guidance allowed me to create something I thought was not possible. His honest opinion and desire to improve my work even when he was not scheduled to work helped me not to give up and to always strive for more.

Thank you to my secondary supervisor, Jose Fonesca. His up-beat attitude and constant questioning regarding this subject kept me on my toes and made me ask the harder questions regarding this project. He may have helped more than he realised.

Special thank you to my family, especially my two brothers who supported me through this whole process and picked me up when I wanted to give up.

A final thank you to those who participated in the interviews, the varied answers helped to form an even stronger conclusion.

Abstract

The Unity 3D engine is used by a large majority of developers to create games. It owns a forty five percent market share and is considered one of the biggest development tools today; this is due to its simple and fast development process which allows for rapid production of game prototypes. However, with over a hundred different options available to develop games, one must ask whether using an engine such as Unity to generate simple 2D mobile games is necessary. This paper aims to discover whether the use of the Unity engine is appropriate for beginner developers who are looking to create 2D mobile games whilst also providing insight into how influential Unity is within education and whether learning more programming orientated applications is beneficial in regards to universal application and longevity.

We will define the criteria for selecting a development methodology and create a 2D mobile game within the Unity engine and replicate this game using Corona SDK. The development process for both implementations will be reviewed and compared then the game will be tested using a benchmark application on various devices to help demonstrate which method was the most optimised and therefore appropriate for mobile development.

Table of Contents

Table of Figures

1.0 Introduction

1.1 Project Justification

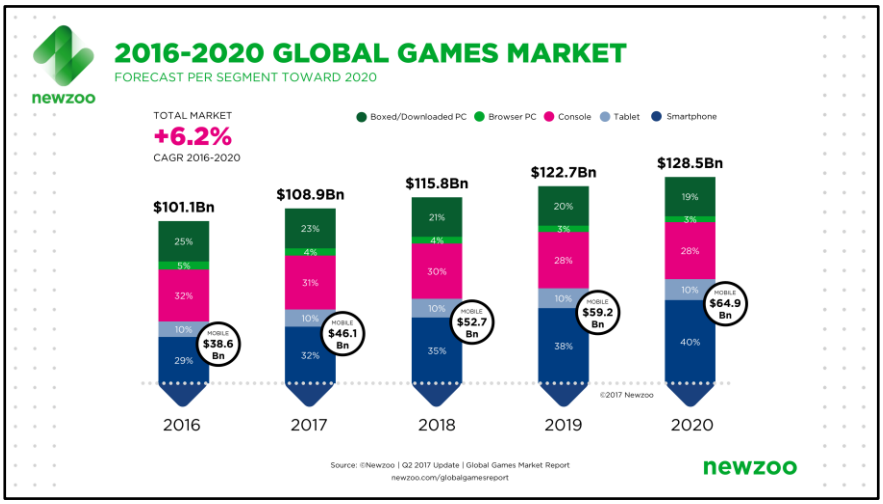


Figure 1. Market share of different platforms

Mobile games have become incredibly popular, consuming a vast market share within the gaming industry and by 2018 mobile games will account for forty five percent of the gaming market revenue. (Newzoo 2016 [1])

Mobile Game Genres Overview

Position & Subject	Monthly Unique Users [Net] [M]	Net Reach [%]	Stickiness [%]	Time Spent per Month per User [Hours]	Active Days per Month [#]	Sessions per Month per User[#]
1. Brain Puzzle	37.8	20	40	06:11	12	92
2. Matching Puzzle	29.9	16	35	07:38	11	56
3. Alternate Reality (AR)	28.7	15	34	04:32	10	73
4. Action/Strategy	13.0	7	40	06:09	12	63
5. Casino - Slots	9.8	5	41	06:12	13	76
6. Building Simulation	8.4	5	35	05:56	7	62
7. RPG Card	3.0	2	22	04:33	11	33
8. Casino - Poker	2.6	1	48	18:51	4	100
9. Race	1.0	1	14	01:12	15	10

Source: Verto App Watch™, U.S. adults ages 18+, Auguest 2016

verto analytics

Figure 2. Overview of popular mobile game genres (Verto Analytics, 2016)

The most popular genres of mobile game are those which lack complexity; brain puzzle games such as cross words, trivial based question games and matching puzzle games. This popularity concerns how many unique users play per month, time spent played per month and 'stickiness', which is defined as how long a player will continuing playing after they have reached a certain skill level and will normally play for the social aspect (competition or team play) rather than actually having true fun (Pierce 2010 [2]). A prime example and pinnacle of these games is the Candy Crush Saga series with over 500 million downloads it is the leader of the mobile marketplace and yet the game itself is considered relatively easy to make.

The reasoning behind these games popularity can be compressed into a few main ideas. It is simple, users can play it anywhere at any time without having to wait, it is based on scoring and achieving the highest score – most people prefer to compete with themselves than others, and finally the gameplay is satisfying and oddly addictive with an overall aesthetically pleasing design.

Figure 3. Candy Crush Saga

According to the UKIE's games industry map, there are nearly a thousand games companies in London. Out of these, roughly seven hundred are working on the mobile platform and over two hundred were formed in the last five years. Not all of these

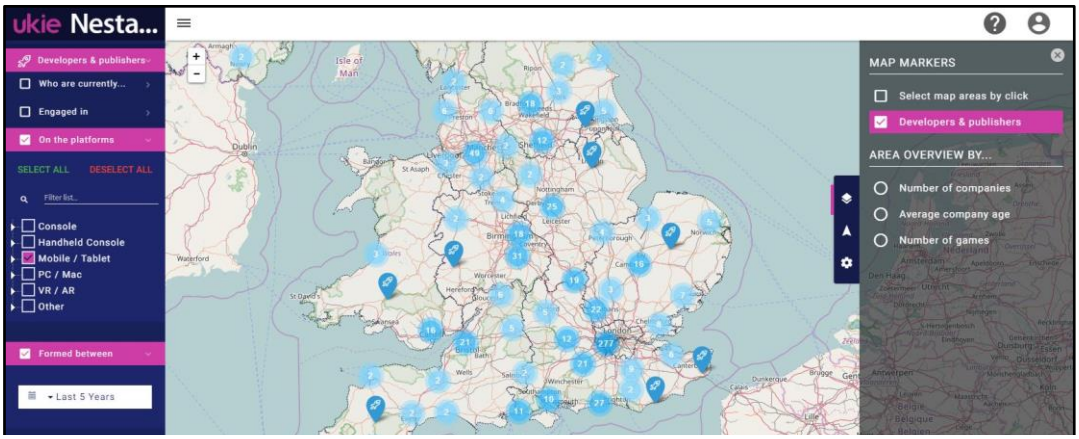


Figure 4 Ukie's Games Map which shows how many companies developing mobile games were formed in the last five years (Ukie Nesta, 2016)

companies will be run by younger developers but it is safe to assume that at least a small portion of them are. With a large influx of students studying game related courses, it is inevitable that a significant majority will be taught how to use Unity to develop games over other methods due to its popularity in the industry coupled with its ease of use.

Considering that nearly two billion mobile devices are running a Unity-made game, and that some of these new companies will be run by students who previously studied a game related course, it is probable that quite a large majority of these companies will be using Unity to develop mobile games.

Figure 5. Unity statistics regarding users, downloads and mobile device usage (Unity Technologies, 2016)

1.2 Alternative Solutions

It has been established that mobile games are becoming increasingly popular and that the most popular genre is puzzle or match type games. The main issue that arises is that new developers who emerge into the market, primarily student run or amateur level developers will assume that the easiest and quickest tool for them to use is Unity or a similar game engine. The reasoning behind choosing Unity is apparent from a student perspective; it has been taught throughout education as an easy to

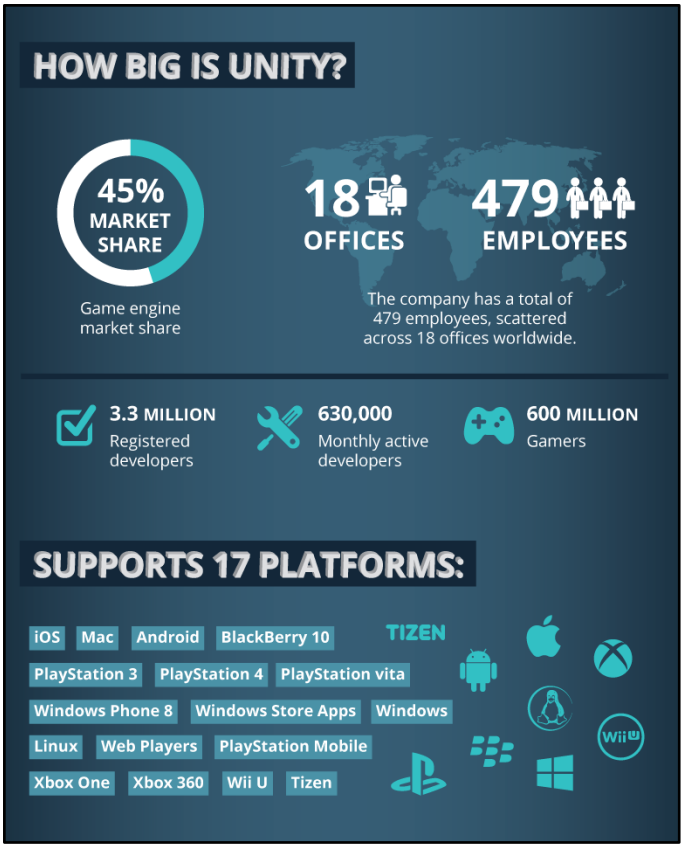


Figure 6. Infographic regarding Unity statistics

use tool, it is marketed as a tool with multiple platform support but where it boasts quantity and speed, it may possibly lack quality and depth. There are many alternatives to using Unity such as writing a game from scratch with a native language, using a graphical API but for the purposes of keeping the implementation relatively simple using an SDK (software development kit) may be the most optimal methodology.

Alternative solutions to Unity offer more than just being lightweight and a higher quality result – they offer a richer learning experience throughout development if logic and programming knowledge is applied. The use of Unity may not demonstrate the fundamentals of game development, such as basic mathematical conventions (e.g. dot product) because someone has already created a function for this convention. Additionally, using primarily C# for scripting purposes does not have a universal application. Whereas using C++ with an SDK or graphical API will not just have a universal application to programming but also build understanding of how a game is created and these basic rules can and will be applied in future game development. In summary, although the learning curve for a non-engine implementation may be steeper, it can be seen as more beneficial for future development and prospects.

Figure 7. Unity Logo

Figure 8. Corona SDK Logo

1.2 Aims and Objectives

The core objective of this paper is to determine whether or not Unity is the correct choice for student or beginner developers creating mobile games. In order to achieve this objective a mobile game will be created using both Unity and Corona SDK. The development process will be documented and compared against specific criteria. Then both games will be tested across a multitude of devices and operating system using some normal and extreme values in order to gauge performance of both

implementations. A conclusion will be drawn upon regarding this subject and future prospects for beginner developers who wish to create mobile games will be reviewed.

2.0 Background and Literature Review

2.1 History of Unity 3D and Corona SDK

In order to undertake this comparative study, one must understand what Unity is exactly and how Corona SDK differs – in their most basic forms they are tools used to create applications and video games.

Unity is a cross-platform game engine developed by Unity Technologies. Unity was created was by colleagues David Helgason, Joachim Ante and Nicholas Francis in Denmark (Haas 2014 [3]). The final product was launched on June 6, 2005 with the aim to create an affordable game engine with professional tools for amateur developers. The original version of Unity was only released for Mac OS X and could only deploy projects onto a few specific platforms. Today, the current version of Unity (5.3.4 as of this writing) is supported on both Windows and Mac OS X, and offers twenty five target platforms with virtual reality build support as well. (Haas 2014 [3])

Figure 9. Default Layout of Unity Engine

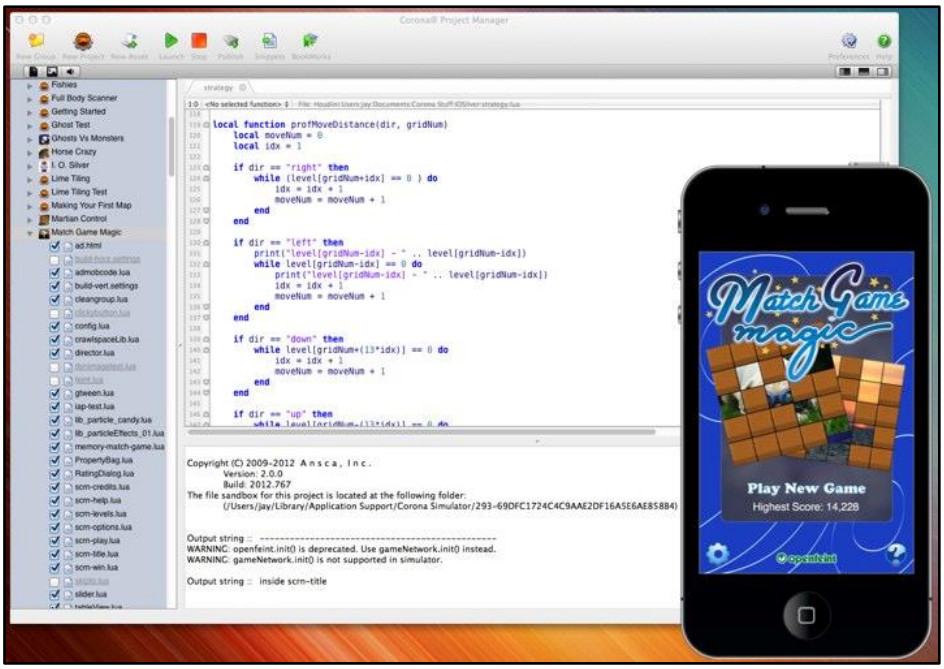


Figure 10. Example of Lua and Corona Simulator

Corona SDK is a software development kit developed by Corona Labs Inc. It uses integrated Lua, layered on top of C++ which is used to build graphical applications such as mobile games. The company, Corona Labs was founded in 2008 by Carlos Icaza and Walter Luh. In 2009, Corona SDK was released, initially supporting iOS. In April 2010, they expanded support for Android devices and in 2011 Corona Labs began development for Amazon Kindle and other E-Readers. In 2012, Corona Labs introduced Corona Enterprise which allows developers to integrate any Objective C and Java libraries, then later expanded offering Corona SDK 'starter', a free mobile development solution. As of 2011, over three hundred thousand developers have used Corona SDK to create games and there is over twenty third-party tools and services which can be used in conjunction with the SDK. (CoronaLabs 2016 [4])

The key difference between Unity and Corona is that Unity 3D is defined as a game-engine whereas Corona is a software development kit (SDK). A game engine is a software framework; which is a collection of libraries and services required to make a vast range of applications, "so one "library" provides support for pretty much everything you need to do. Often a framework supplies a base on which you build your own code, rather than you building an application that consumes library code." (Stackoverflow 2012 [5]) An SDK is a library that help develop code which uses a particular system (e.g. Windows SDK), drawing 3D graphics (Corona uses Lua which

is layered on top of C++/OpenGL SDK) etc. An SDK is a set of tools to help you create a final product where a game engine is a base to build from and as such the two development implementations vary and have specific positive and negatives regarding game development (Stackoverflow 2014 [6]).

It is important to note that game development software is mostly specific to certain platforms so when comparing engines with one another the components which focus on creating 2D mobile games are compared and nothing else as this is a false representation. For example, real time particle effects, a 3D graphics algorithm, cannot be used in mobile games but the engine may use it for a higher level platform. This is important as Unity is considered a 3D engine so using it for mobile development one must consider the elements which concern 2D mobile development and no other elements.

Figure 11. 2Dvs3D

2.2 Software Comparison

Game engine selection methodology has been defined by (Petridis et al, 2010 [7]). In order to make a comparison, criteria must be defined; they defined this criteria as audio-visual fidelity, functional fidelity, composability, accessibility, networking and heterogeneity. Then they break these criteria down into smaller sections – not all criteria will be used in this research as their criteria applies to all game development rather than just mobile. A key criteria which falls under the 'accessibility' heading is the 'learning curve'. This is an important factor as amateur developers will choose an easier route to develop games as it can produce results in a quicker timeframe but occasionally choosing the route with a steeper learning curve can be more beneficial in regards to how much knowledge will be learned and how this can be applied on a larger scale. In addition, it will help increase longevity when developing games – creating something from scratch will give a deeper understanding and less repetitiveness.

A similar paper compared game engines which derived some of the criteria from Petridis' work (Akekarat 2014 [8]). Although the research focuses on building to multiple platforms, the conclusions drawn upon take into account how suitable each engine is depending on what platform is used. There is a focus on ease of learning too, this an important factor to consider with amateur developers who require guidance with complex issues. Also taken into account is what programming language is used for each software which has universal application within the gaming industry. The conclusion describes which engine was best in regards to specific criteria and genre; there was no one main winner but it appeared as though choosing software for development relied on two main things: experience of the developer and the type of game one wishes to create.

Another paper by (Haag and Kleinschmidt 2016 [9]) conducted a comparison between certain game engines for the creation of 'Serious Games for Health'. They narrowed specific game engines by not including those which did not offer cross-platform support for at least Android and iOS. Later, research was conducted which included developers with more experience to help rule out any outliers and three categories were created: powerful 3D game engines - in which Unity was chosen, 2D game engines – Marmalade engine was chosen for this and finally Turbulenz HTML5 Game

Engine was chosen for an engine without a graphical editor. Aspects of each engine were compared against one another with a rating of one to five and discussed in detail later. The criteria which apply to this paper are as follows: licencing, installation issues, available OS, amount of understanding needed, scripting editor, physics engine, collision detection and cross platform deployment /testing. The results showed that Marmalade was the best solution for creating serious health games and Unity was a close second, stating that, “The game can be deployed to different target platforms, but as the simulation of the scene on a mobile device can only be done after deployment, the Marmalade engine is the better choice here.” (Haag and Kleinschmidt 2016 [10])

Peer reviewed papers which directly compare Corona SDK and the Unity engine are scarce due to Unity becoming popular relatively recently and SDKs being used by developers who have more experience in the field. However, research in relevant gaming forums such as “Stack Overflow” was conducted to include experiences from users that have more expertise in this area. Consulting both the Unity and Corona forums the general consensus was that Unity was more powerful overall especially when creating 3D applications. On the other hand, Corona seemed the better choice for smaller, 2D games which required a small timeframe and learning curve [11]. In addition, coding within Corona and Unity is relatively easy to learn – either using C# with Unity or Lua with Corona, but compilation times vary greatly. Developers pointed out that making small changes within Unity was a long process as it involved waiting for compiling but in Corona the compilation time is reduced and as a result small changes can be made quickly – the only downside is that the compilation happens online [12]. Finally, an article outlined the pros and cons for both Unity and Corona. The pros for Corona were that it runs quickly, changes can be made rapidly but plugins/third party tools are limited and the cost is quite steep. Unity was praised on how powerful it can be but it can take some time to adjust to using it, “launching Unity for the first time, you may feel like the pilot of a 747 jet plane.” (Renatus 2014 [13])

2.3 Software Testing

Creating a fair testing environment for mobile games is quite different to other games, in this example testing refers to the performance of the mobile application rather than the user experience. Mobile application testing guidelines have been

created by (Gao et al 2014 [14]) who states that not only do mobile applications have to work anywhere and at any time, they also should work across platforms, different operating systems, display sizes and not drain battery life. They later split testing into separate goals; quality of service testing, reliability, and interoperability testing – these are the relatable testing methods which will help achieve a fair conclusion. Finally, they outline different approaches for testing a mobile application. Device-based testing requires multiple devices and time but for this scale seems the most appropriate for it can "…verify device-based functions, behaviours, and QoS parameters that other approaches cannot." (Gao et al 2014 [15])

Another decision must be made regarding mobile game testing and that is the choice to manually or automatically test. Automated testing refers to implementing code within an application which will carry out certain actions in order to gauge performance of a given application. In (Amen et al 2017 [16]) research on mobile application testing he found that manual testing may be more time consuming but doesn't require the programming skill to initially set up the automation for automatic testing, moreover, for testing performance and playability using real people gives more accurate results. Using this data, it is apparent that automated testing may not be achievable due to time constraints and lack of programming knowledge. In addition, user testing is not required as the game is simple enough to test all features in a small time window.

Finally, a blog post conducted by (Manouchehri 2012 [17]) explains how to unit test within Unity and the challenges that arise from it. The post discusses how to test specific functions for bugs or optimisation within the code if one is using Unity to develop games, and this can be extremely beneficial in narrowing down exactly what is causing certain issues. An important piece of advice given is that in order to test your game first figure out what aspects of a piece of code can be tested and what variables are expected to change or not change frequently – consider aspects which may be too volatile. Finally, Manouchehri [18] mentions that "…all code can be subject to change…" in order to fairly test a game then simple and extreme value tests need to be run in consideration to specific pieces of code.

3.0 Design, Development and Testing

3.1 Previous Comparisons

Interviews with independent game developers were conducted by the author to help understand the reasoning behind why they chose certain software to develop their games. Questions began with the discussion of previous experience with game development and different tools used to develop these certain games then later interviewees provided the reasoning behind choosing these tools. Interviews progressed further into opinions regarding the ease of using an engine, whether 2D mobile indie games require an engine for development. Finally, the interview is concluded with questions relating to how to choose software for game development and whether or not a steeper learning curve can produce higher quality results. It is important to note that all interviewees were not developing mobile games but all were or had been independent developers who are creating games in small teams.

are currently creating a horror game using Unreal Engine 4. They are a team of seven with nearly three years of experience and have released a small game leading up to the release of their main game. The main reasons they gave behind choosing Unreal Engine 4 to develop games is that it allows freedom of control, multiple platform support and primarily because they had experience using it which was gained from University education. The idea to use an engine to create mobile games was held in a positive light, they claimed it was a great source of information and using an SDK could be easier but it may restrict what type of games one can make as they normally focus on a single type of game. Finally, they believed that a steeper learning curve can be dangerous for amateur developers as it has the risk of overwhelming the developer – starting with a drag and drop/scripting ideology uses the same logic as a pure code solution just in a different way.

team is made up of three people but there was one main developer for the creation of a variation of games ranging from mobile to browser based. Starting after the completion of university they hold three years of experience with roughly fifteen games released mostly designed for web but also iOS and Android. Their main tool for development was UDK but had switched recently to Construct 2 which uses C# and mostly scripting rather than programming. They had experience with using Unity but branched into Construct 2 when they began to develop more web based games. Discussions regarding using an engine to develop mobile games were positive, “Why reinvent the wheel?” was the general consensus. The downsides with using Unity to develop games were apparent – high pricing, lack of control and limited optimisation techniques but overall they were advocates for using Unity. The key advice gained from this interview was that one needs to consider what type of game is being created and what skills are possessed before deciding on the best methodology to create said game.

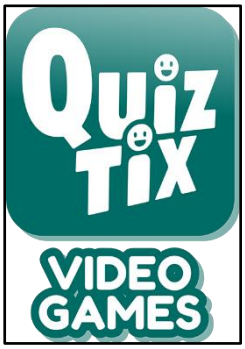


Figure 12.
QuizTix Logo

The last independent developer interview was held with a more experienced developer called Ian Masters. He is the head of three different teams, has been creating games since 1998 and has made roughly forty games most of which are mobile based. His main game which has been in top ten on iOS marketplace, ‘QuizTix’ is held in high regard within the mobile games industry as well as many others such as ‘Finger Hoola’ etc. The main tool he used for development of mobile games was Corona SDK and had been using it for a number of years prior to this interview. The main reason given for this development was that the games he was developing were for mobile and were

also 2D so Corona SDK was the most optimal tool for this type of development. He hadn't had much experience with alternative development routes which is due to the fact that the majority of games that had been made or will be made are designed for mobile so there is no need for change. On the topic of using engines, the interviewee felt Unity works for developers who want to create a game in a small timeframe and easy manner. The only issues that arose was limitation to one toolset and one language may restrict future development and prospects. The drawn upon conclusion was as previously stated; what type of game do you wish to create, what experience do have in this field and what are your future plans? These are the questions that one must ask when deciding on what tool is best for development of 2D mobile games.

Figure 13. Amuzo Logo

A final interview was held with a former game research engineer working at an independent games company, 'Amuzo'. While not a developer, the interviewee has many years of experience within the field and a result held in depth knowledge regarding this subject whilst offering an alternative view. During his experience, the most commonly used software to develop small mobile games was Flash with C++ or C. He felt that using an engine to develop mobile games, especially indie ones, was "overkill", as it gives the developers a false view on game development – using an engine was adding more complexity to an already simple task. He held the opinion that Unity offers a common base to work from and as a result offers a universal application but it also limits understanding of game development and maintaining the game afterwards can be difficult due to deprecation. Furthermore, using Unity or similar engine allows one to jump far into game development but as a result there is a lack of understanding of how certain aspects work, for example, Shaders and languages surrounding them. Finally, he recommended that developers should not be thrown off by other methodologies, stating that by trying other forms of development you may find it is easier and more suitable, but it requires trying them first.

The overall consensus left Unity in a positive light, most developers claimed that they had used it in the past and the experience was quick and relatively easy. However, those who had more experience in this field and had developed more games had slightly differing opinions, stating that it is a useful and powerful tool but lacks freedom

of control, universal application and the fact that one can build to a large array of platforms meant that the overall quality of the build was poor i.e. quantity over quality wasn't a balanced exchange. An extra anecdote; those who could not find many issues with using Unity (or similar engine) were also those who had little experience in other methodologies of development and those of whom did have experience in other software claimed that the learning curve may be steeper for programming orientated implementations but also has a larger scope of application. When asked which programming language to study first when beginning game development, most answered with C++ as it is used more frequently within the industry however Unity uses only C# which again, may restrict amateur developers to one development route. Finally, the choice of which tools to use when developing mobile games was defined by two principles: what type of game is being created and what experience is possessed at the time – an idea reinforced by prior literature.

3.2 Software Comparison

This solution aims to eliminate the issue that beginner developers face when deciding on which software to use when developing mobile games. In this paper a comparison and analysis is carried out on two different implementations of a 2D mobile game, using Unity as the engine and Corona SDK. Criteria was formed based on research found during the review of literature and in order to compare the two development processes criteria is split into individual categories for fair distribution of comparison:

- Audio visual fidelity, how images are drawn to the screen, animated and re-distributed throughout the game which mostly is comprised of 2D sprite animation.
- Functional fidelity, regarding scripting language, efficiency of language, collision detection and basic physics – the basic mechanics which make up the game.
- Composability, asset importing and exporting limitations as well as available third party tools/libraries.
- Accessibility, this regards mostly the learning curve for each development process and documentation readily available to help support development or any issues that may arise.
- Heterogeneity which is the overall quality of multiplatform support.

The game which will be created must feature the basics of a 2D mobile puzzle type game so it is applicable to the general consensus. Therefore, the game must contain specific elements:

- UI system; a menu in which users can navigate through the game
- Layered background elements which feature parallax scrolling
- Basic enemy AI and a scoring system
- Collision with enemies and a life system
- Touch controls and basic player movement linked to touch controls

Figure 14. An example of the game which was created

3.3 Unity Development Process

3.3.1 Composability

Importing assets into Unity was a relatively easy task; dragging the required assets from a source folder into the Unity editor was efficient and any assets which required extra steps (such as splitting sprites into segments for animation) were done by the program. Within the editor, the asset explorer allows creation of folders and sub-folders meaning navigation regarding assets has been simplified and thus an easy task.

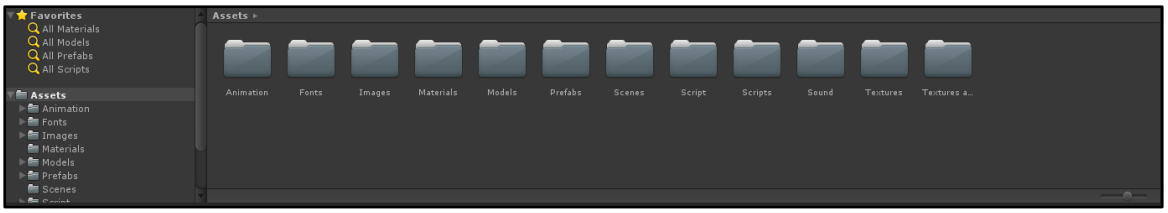


Figure 15. Unity File Explorer

An issue with this easy importing technique is that certain actions which Unity will undertake when importing are not apparent at first which can lead to issues arising at a later stage in development. Although this was not an issue for this development due to experience using Unity, one can see how this may affect first time users.

3.3.2 Audio Visual Fidelity

Once assets had been imported, development began and the initial stage was to create background elements. Dragging and dropping an image into the scene then scaling it correctly to fit the camera bounds a background was created, adding other foreground elements was the same process just a different position within the camera bounds. Creating the parallax effect can be implemented in many different ways; by moving an image then replacing it with a duplicate when it reaches the end of the screen and moving the duplicate image then loop the effect is a simple technique.

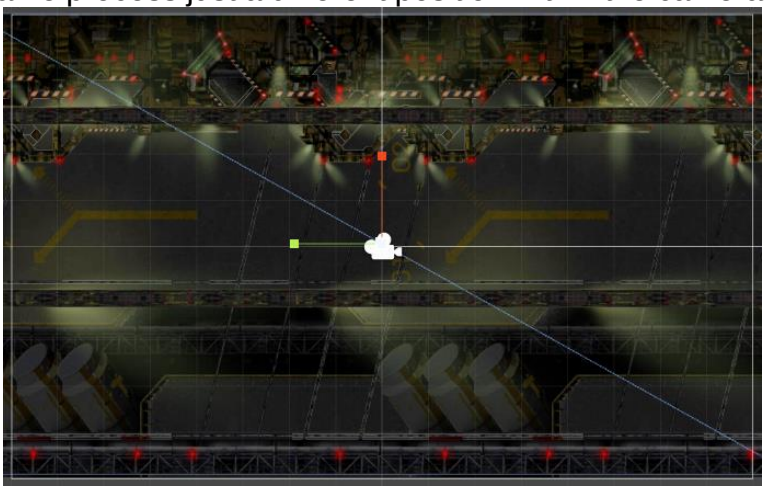


Figure 16. Background Elements in the Scene

However, for this demonstration texture offsetting was used to implement the effect. This simply means apply the image as a texture to an object and offset the image by

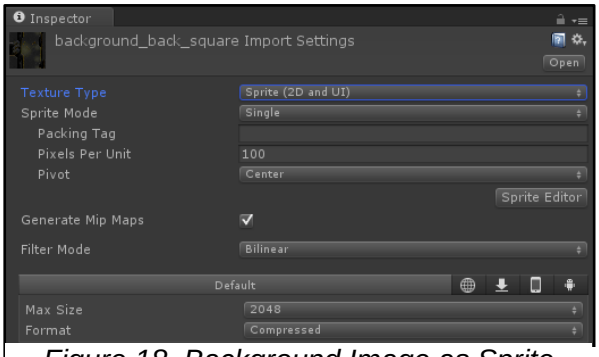


Figure 18. Background Image as Sprite

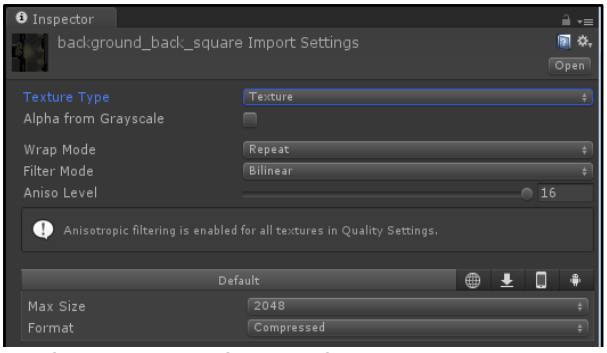


Figure 17. Background Image as Texture

a certain rate and loop this effect. To carry out this effect in Unity it first requires transforming the images that were imported into textures from sprites – which can be changed with relative ease.

Next, a 3D quad (a flat plane) was placed into the scene and scaled to fit the camera, the texture was applied to the quad and a small script which offsets a texture from right to left at a specified rate. It is important to note that this process requires complex mathematics but due to rich documentation and an already existing function provided by Unity the task was simplified but not understood to a full degree.

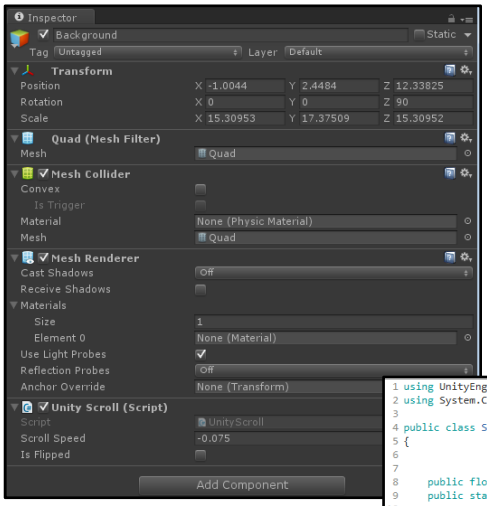


Figure 20. The 3D Quad with no Texture

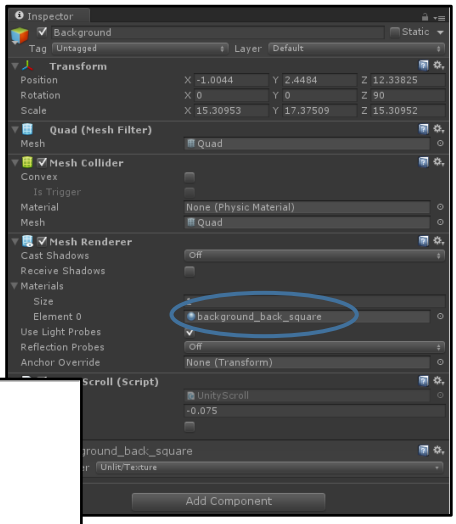


Figure 19. 3D Quad with Texture Applied

```
using UnityEngine;
using System.Collections;

public class Scroll : MonoBehaviour
{


    public float m_speed;
    public static Scroll current;

    float pos = 0;
    Renderer m_renderer;

    // Use this for initialization
    void Start()
    {
        m_renderer = GetComponent<Renderer>();
        current = this;
    }

    void Update()
    {
        Go ();
    }

    // Update is called once per frame
    public void Go()
    {
        pos += m_speed;
        if (pos > 1.0f)
        {
            pos -= 1.0f;
        }
        m_renderer.material.mainTextureOffset = new Vector2(pos, 0);

    }
```

Figure 21. The Parallax Scrolling Script

The final visual aspect that needed be carried out was the player and animating the ship sprite image. A submarine sprite sheet was provided by a previous mobile game development and the spaceship sheet was found online. Using the sprite animator provided by Unity, the sprite sheet was automatically split into the different phases of the animation and when it was dragged into the scene the animation was created and set on a loop. Once again, this was very simple and easy to carry out but there was obvious ignorance to what was happening behind the scenes – what actually

happened when the sheet was split and the animation was set up was dismissed because the action had already taken place.

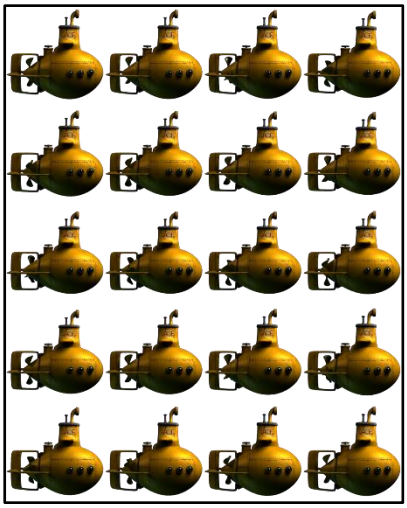


Figure 24. Original Sprite Sheet

Figure 23. Sprite Sheet split in Unity

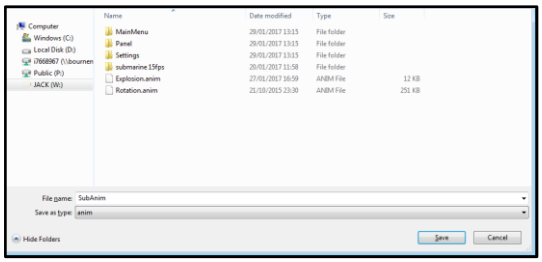


Figure 22. Saving the Animation after placing it in Unity

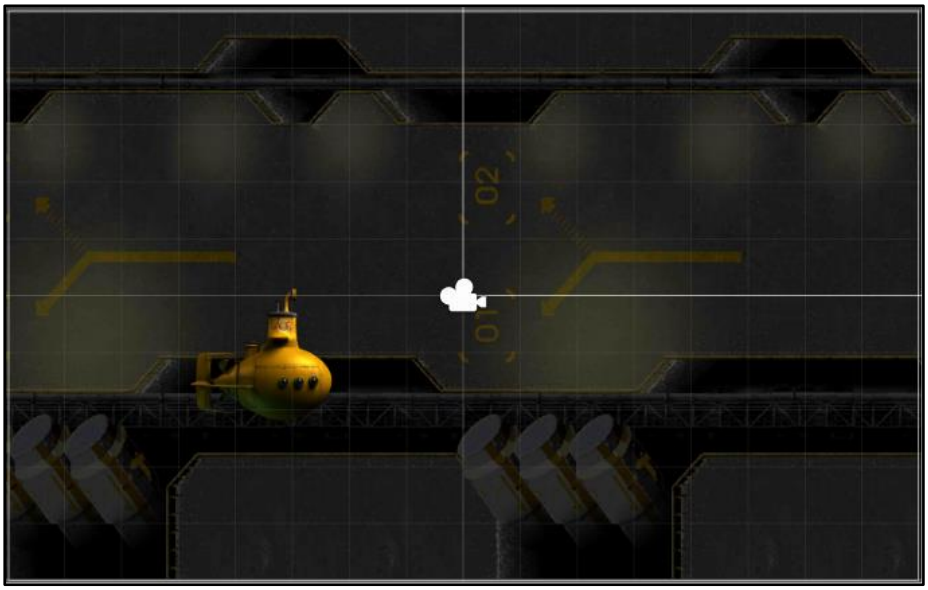


Figure 25. The Submarine/Ship is animated and placed in the scene

3.3.3 Functional Fidelity

Scripting within the Unity editor requires some knowledge of C# and a text editor, Unity provides Monodevelop as a scripting editor but many developers choose Visual Studio as it is somewhat easier to use and slightly faster. To fairly test the Unity engine on

```
using UnityEngine;
using UnityEngine.EventSystems;

public class DragAndDrop : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler
{
    public static GameObject DraggedInstance;
    public int lives;

    Vector3 _startPosition;
    Vector3 _offsetToMouse;
    float _zDistanceToCamera;



    public void OnBeginDrag (PointerEventData eventData)
    {
        DraggedInstance = gameObject;
        _startPosition = transform.position;
        _zDistanceToCamera = Mathf.Abs (_startPositicn.z - Camera.main.transform.position.z);

        _offsetToMouse = _startPosition - Camera.mair.ScreerToWorldPoint (
            new Vector3 (Input.mousePosition.x, Input.mouseFosition.y, _zDistanceToCamera)
        );
    }

    public void OnDrag (PointerEventData eventData)
    {
        if(Input.touchCount > 1)
            return;

        transform.position = Camera.main.ScreenToWorldPoint (
            new Vector3 (Input.mousePosition.x, Input.mouseFosition.y, _zDistanceToCamera)
        ) + _offsetToMouse;
    }

    public void OnEndDrag (PointerEventData eventData)
    {
        DraggedInstance = null;
        _offsetToMouse = Vector3.zero;
    }

    public void Update()
    {
        if (lives <= 0)
            Destroy (gameObject);
    }

}
```

Figure 26. The Touch Movement script

the whole, Monodevelop was used for scripting. In order for the player to be able to move the ship by touch and drag a script which detects where the screen has been touched and let go must be implemented. Firstly, a function provided by Unity, 'Input.touchcount' will recognise if a touch has happened, the script will track where the touch is and create a new position based on where the touch ends (and after converting the position from screen space to world space), this script is attached to the ship so only it can detect touches and thus movement by touch is created.

Creating enemies within the game relied on a 'spawner', where enemies are created or 'spawned'. The first step is to create an enemy which moves right to left, carried out by constantly changing the enemies' position on the X axis. To spawn the enemies at a given rate required a script which would create enemies between two random points in order to create randomness within the game, then a Unity provided function which will repeat an action between a given start time and rate allows the enemies to constantly spawn in. Later, a sine wave and rotation was applied to enemies' movement to make the game feel more like an actual game. Collision was a matter of attaching a collider (something which can detect a hit) to the player and the enemies, then a script would tell the game what to do if an enemy collided with the player – in this case, loose a life.

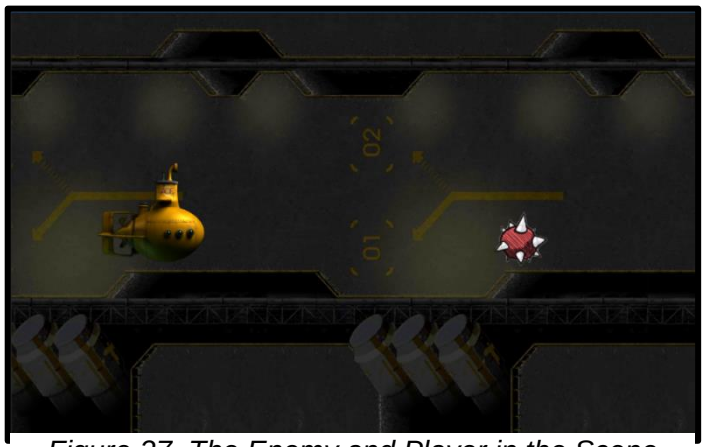


Figure 27. The Enemy and Player in the Scene

```
using UnityEngine;
using System.Collections;

public class Enemy : MonoBehaviour {

    public float m_speed = 2f;

    Rigidbody2D m_rigidbody;

    // Use this for initialization
    void Start () {

        m_rigidbody = GetComponent<Rigidbody2D>();
        m_rigidbody.velocity = new Vector3(-1f, 0f, 0f) * m_speed;

    }

    public void Update()
    {
        RotateLeft ();
    }

    void OnBecameInvisible() {
        Destroy(gameObject);
    }

     void OnCollisionEnter2D (Collision2D m_collider)
    {
        GameObject playerLives = GameObject.Find ("Player");
        DragAndDrop Lives = playerLives.GetComponent<DragAndDrop> ();

        if (m_collider.gameObject.CompareTag ("Player"))
        {
            Lives.lives--;
            Destroy (gameObject);
        }
    }

    void RotateLeft () {
        transform.Rotate (Vector3.forward * -90 * Time.deltaTime);
    }

}
```

Figure 29. The Enemy Script

```
    using UnityEngine;
using System.Collections;

public class Spawner: MonoBehaviour {

    public GameObject Bomb;

    Renderer m_renderer;
    float SpawnTime = 3.0f;


    // Use this for initialization
    void Start()
    {
        StartCoroutine(IncreaseSpawnSpeed());
    }

    IEnumerator IncreaseSpawnSpeed() {
        m_renderer = GetComponent<Renderer>();

        InvokeRepeating("addEnemy", SpawnTime, SpawnTime);

        yield return new WaitForSeconds(25f);
        InvokeRepeating("addEnemy", SpawnTime/2, SpawnTime/2);
    }
    public void addEnemy()
    {
        var y1 = transform.position.y - m_renderer.bounds.size.y / 2;
        var y2 = transform.position.y + m_renderer.bounds.size.y / 2;

        var spawnPoint = new Vector2(transform.position.x, (Random.Range(y1, y2)));

        Instantiate(Bomb, spawnPoint, Quaternion.identity);

    }
}
```

Figure 28. The 'Spawner' Script

3.3.4 Accessibility

Many of the techniques that were carried out throughout this implementation were derived from online documentation. The parallax scrolling techniques were found on the Unity tutorials library and the one seen within the game were carried out by using knowledge gained from tutorials. In addition, the spawning script was found online in a Unity forum – it is a common convention used in 2D games so there was a rich amount of data and examples found online to help support this convention. In addition, issues that arose such as textures not offsetting correctly, certain functions which required specific spelling were all remedied using online solutions. It is apparent that the documentation and support for Unity development is incredibly rich however, this can lead to ignorance when developing game – there are many aspects which were not understood during development such as sine waves, sprite animation and how Unity communicates with a device to recognise touch input.

3.3.5 Heterogeneity

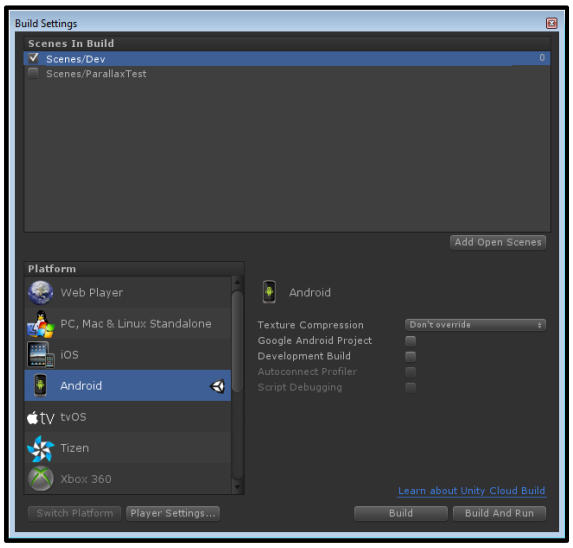


Figure 30. Build Settings within Unity

Unity offers a vast array of platforms to build to but for this scenario the compile and build times in regards to Android development were reviewed. Due to the sheer size of the engine and the speed of compilation within Monodevelop, there was a lot left to be desired when making a small mobile game which requires a lot of small details to be changed. Finally, building the application to Android was a seemingly long process due to the nature of how it is built (using a custom virtual machine) and the total file size seemed unnecessary large. This meant that if a small changed needed to be made, it meant waiting at least five minutes for an APK to be created and in the professional industry this is too long for mobile development.

To summarise, the Unity implementation was successful to almost all degrees. The development process was streamline and complex issues that arose were answered quickly and efficiently. Animating sprites were carried out mostly by the engine and mobile integration was handled with almost no problems. In addition collision handling and physics were easy concepts to grasp and once again, issues that arose were answered through online documentation. The downside to this process is that the learning curve was too shallow and a result a deeper understanding of how the game works and the fundamentals of game development conventions were overlooked. Finally, slow compilation and build time made testing small changes an inconvenience. To generate a quality prototype in a small timeframe, this is one of the most efficient ways.

3.4 Corona Development Process

3.4.1 Composability

Figure 32. Visual Studio Code Logo

Learning Corona SDK and Lua (a scripting language) from scratch may appear to be a somewhat daunting task but once the basics have been laid out the rest seems to flow. The initial phase of the Corona SDK development process was considerably different from the Unity process. Due to the fact that Corona SDK supplies no graphical interface, all changes must be made with scripting in Lua which required learning a new language and adjusting to working without any visual feedback. Visual Studio Code was used as the editor for writing the Lua scripts and the simulator provided by Corona meant that the game could be tested on various different devices when changes were made.

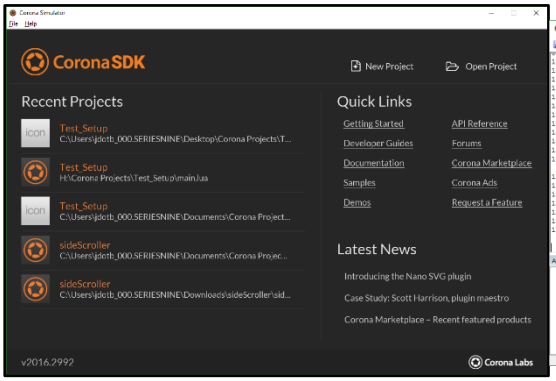


Figure 33. Corona Simulator; Project Loading

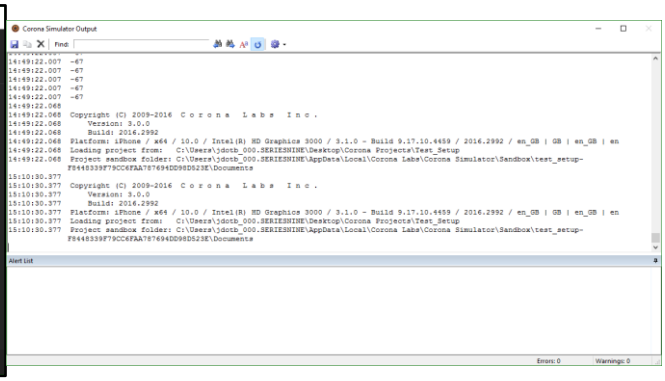


Figure 31. Corona SDK Output Console

```
12  local composer = require ("composer")
13  local scene = composer.newScene()
14
15  local widget = require("widget")
16
17  --graphics
18
19  local background = display.newImage("background.png", true)
```

Figure 34. Lua Code to load an Image within Corona SDK

The initial set up of loading images is the same process undertaken within the Unity development but carried out within scripting rather than dragging and dropping. A variable is declared, e.g. ‘Background’, and an image is loaded using a function provided by the SDK into that variable, when viewed in the simulator the image appears in the top right and has been scaled to fit the screen.

Figure 35. Initial Image Loaded in Corona SDK Simulator

This is how the majority of declarations are made within Corona SDK; a variable name is given, an image is loaded and the properties of that variable can be changed, moved etc. later in the script. This allows full control over variables within the game as they can be accessed anywhere within the script in a multitude of different of ways such as creating an instance of that variable which can be changed independently of the original variable which is incredibly powerful aspect of Corona SDK.

3.4.2 Audio Visual Fidelity

Once the background variables had been set up, it was a matter of scaling them correctly to fit the screen. Scaling is an aspect which is overlooked in game development as a lot of engines will carry this operation out independently of the developer. Corona SDK has multiple options for scaling images and the graphical user interface aspects of any given game. The ‘letterbox’ option will scale the content to fit the screen while preserving the same aspect ratio which may give black bars on the edge of the screen if done poorly. The ‘zoomEven’ option will scale the content area

to fill the screen while preserving aspect ratio but will stretch the images to fit the screen – this is useful for fitting content on all devices. For this implementation, the 'letterbox' scaling technique was used as it helped scale for larger devices and the simplicity of it was appropriate for the type of game being created. Once the scaling had been set, background and foreground elements were declared and scaled to fit the smallest screen size and the images could be up scaled to fit various screen sizes.

```
application =
{
    content =
    {
        width = 320,
        height = 480,
        scale = "letterbox",
        xAlign = "center",
        yAlign = "center",
        fps = 60,


        imageSuffix =
        {
                ["@2x"] = 1.5,
                ["@4x"] = 3.0,
        },

    },
}
```

Figure 37. Config Script where Scaling Options are set

```
--graphics

local background = display.newImage("background.png", true)
background.anchorX = 0
background.anchorY = 1
background.x = 0
background.y = 400
background.speed = 1

local backgroundCopy = display.newImage("background.png", true)
backgroundCopy.anchorX = 0
backgroundCopy.anchorY = 1
backgroundCopy.x = 640
backgroundCopy.y = 400
backgroundCopy.speed = 1

local midground = display.newImage("midground.png", true)
midground.anchorX = 0
--midground.anchorY = 1
midground.x = 0
midground.y = 105
midground.speed = 3

local midgroundCopy = display.newImage("midground.png", true)
midgroundCopy.anchorX = 0
--midgroundCopy.anchorY = 1
midgroundCopy.x = 640
midgroundCopy.y = 105
midgroundCopy.speed = 3

local foreground = display.newImage("foreground.png", true)
foreground.anchorX = 0
foreground.anchorY = 1
foreground.x = 0
foreground.y = 460
foreground.speed = 4

local foregroundCopy = display.newImage("foreground.png", true)
foregroundCopy.anchorX = 0
foregroundCopy.anchorY = 1
foregroundCopy.x = 640
foregroundCopy.y = 460
foregroundCopy.speed = 4
```

Figure 36. All Background Elements Set up

Figure 38. All Background Elements seen in the Simulator

Unlike Unity, Corona SDK only handles 2D aspects of game development which meant parallax scrolling could not be carried out by offsetting textures on a 3D quad. In any case, a simpler solution was used. By placing an image on the screen and a copy of it directly behind it, a function was created which moved the first image along the X-Axis and when it reached the end of the screen the copy would do the same and this process was looped through an 'if' statement – a process which relied on logic

over demonstration. This function was then applied to the other background elements but due to the fact there was no visual scaling, the images had to be re-sized in a photo editing program in order to comply with the function – or the whole function was copied with different values dependent on the image size.

```
local background = display.newImage("background.png", true)
background.anchorX = 0
background.anchorY = 1
background.x = 0
background.y = 400
background.speed = 1

local backgroundCopy = display.newImage("background.png", true)
backgroundCopy.anchorX = 0
backgroundCopy.anchorY = 1
backgroundCopy.x = 640
backgroundCopy.y = 400
backgroundCopy.speed = 1
```

Figure 40. The Background and its Copy

```
function scrollBackground (self, event)
    if self.x < -635 then
        self.x = 640
    else
        self.x = self.x - self.speed
    end
end
```

Figure 39. The function will allows Background Elements to 'Scroll'

Once again, the final visual aspect was the player and animating the ship/submarine. Creating sprite animation was rather straight forward due to the documentation on the Corona website. The idea behind sprite animation is to tell the engine the size of each individual image, for example, a square image of 128 pixels with four states would mean that each animated state is 32 pixels in size. Once the properties of the sprite sheet have been set, options for the animation must be set. This means telling Corona that the sprite sheet given has a set amount of frames, what frame the animation should start from and whether or not to loop it. Then later in the script, this sprite sheet can be called and be given a position and other properties where the animation will run when called using, '[animation]: Play'.

```
local sheetOptions =
{
    width = 256,
    height = 256,
    numFrames = 20
}
```

Figure 42. Options for Sprite Sheet

```
local sequences_sub = {
    {
        name = "normalAnim",
        start = 1,
        count = 20,
        time = 800,
        loopCount = 0,
        loopDirection = "forward"
    }
}
```

Figure 41. Options for Animation of Sprite Sheet

```
local subAnim = display.newSprite(ship_sheet, sequences_sub)
subAnim.x = 200
subAnim.y = 200
subAnim:scale(0.7, 0.7)
physics.addBody (subAnim, "dyanmic", {density = .1, bounce = .1, friction = .1, radius = 30})
subAnim.gravityScale = 0.0
subAnim:play()
subAnim.lives = 8
```

Figure 43. Attaching other Properties to the Animated Ship and Running the Animation

3.4.3 Functional Fidelity

The scripting aspect within Corona SDK uses Lua which is derived from C++ so having knowledge regarding C++ may be somewhat beneficial but scripting relies more so on logic and an understanding of game mechanics. For this scenario, Visual Studio Code was used as the text editor due to its compact size and fast loading times. Implementing touch mechanics in order to move the ship was carried out by using code mostly found online due to lack of experience and small timeframe. The basic understanding of the touch functionality is a run time event called 'touch'. Using the event system, the engine can recognise when the touch has started and when it has been moved and let go. Applying this code to the player means that the system will recognise when a touch has begun on the ship and where it has been moved to, then move the ship sprite according to the calculations. Finally, the event listener must be called afterwards in order to recognise when the touch has happened and act on the function provided.

```
121  function subAnim:touch( event )
122      if event.phase == "began" then
123
124          self.markX = self.x    -- store x location of object
125          self.markY = self.y    -- store y location of object
126
127      elseif event.phase == "moved" then
128
129          local x = (event.x - event.xStart) + self.markX
130          local y = (event.y - event.yStart) + self.markY
131
132          self.x, self.y = x, y    -- move object based on calculations above
133      end
134
135      return true
136  end
```

Figure 44. The Touch Function within the Player Script

```
238  subAnim:addEventListener( "touch", myObject )
```

Figure 45. The Event Listener for the Touch Function

The first step to creating enemies was similar to the Unity implementation. Enemies were declared, given images and scaled correctly. They were then moved from the right hand side of the screen to the left by changing their position on the X axis. Again, a sine wave was applied to the enemies to give them random movement and add complexity to the game.

```
162  function spawnMines(self, event)
163      if  self.x < -50 then
164          self.x = 700
165          self.y = math.random(0, 300)
166          self.speed = math.random(1,2)
167          self.initY = self.y
168          self.amp = math.random(20, 30)
169          self.angle = math.random(1,240)
170      else
171          self.x = self.x - self.speed
172          self.angle = self.angle + 0.1
173          self.y = self.amp * math.sin(self.angle) + self.initY
174      end
175  end
```

Figure 46. The Mine Spawning Function

Spawning enemies could not be carried out by using a function which can repeat certain actions as there is no such function. This action must be carried out using a slightly more complex programming convention. Creating a table (or array) then adding the enemy to this table meant that multiple instances of the same object could be generated. Using a public variable, one can state how many enemies can spawn and a runtime event will call the function depending on how many values were in the table. This is a core concept of programming and game development yet the first time it has been used correctly, as educational institutions only use Unity there has never been a reason to understand this logic.

```
76  for i=1, mineCount, 1
77  do
78      i = display.newImage ("SeaEnemy.png", true)
79      i.x = 700
80      i.y = math.random(0, 300)
81      i.speed = math.random(1,2)
82      i.initY = i.y
83      i.amp = math.random(20, 30)
84      i.angle = math.random(1,240)
85      i:scale(0.2, 0.2)
86      physics.addBody (i, "static", {density = 0.1, bounce = 0.1, friction = 0.1, radius = 15})
87      table.insert(mineList, i)
88      print(i)
89  end
```

Figure 47. The Mine properties are first set then entered into the table

```
242  for i=1, table.getn(mineList), 1
243  do
244  mineList[i].enterFrame = spawnMines
245  Runtime:addEventListener ("enterFrame", mineList[i])
246  mineList[i].collision = onLocalCollision
247  Runtime:addEventListener ("collision", mineList[i])
248  end
```

Figure 48. A 'for' loop is created which will spawn enemies based on how many are in the table and loop this action

Colliders were added later in development, this was a simple process of applying a physical body to the enemies' properties with a few options such as size of collider etc. Then a function was created in which an event listener will recognise collision and if an enemy collided

```
177  local function onLocalCollision(self, event)
178      if event.phase == "began" then
179          print(subAnim.lives)
180          for j=1, table.getn(mineList), 1
181          do
182          if event.other == mineList[j] then
183              print(mineList[1])
184              subAnim.lives = subAnim.lives - 1
185              transition.to(event.other, {x = -70, y = 0, time = 0})
186              end
187          end
188      end
189  end
```

Figure 49. Collision function which is run on a 'for' loop based on how many entries are in the enemy table

with the player then a certain action will happen such as loose a life and delete itself. As there was no visual aid to help show where the collider was on the enemies and player, small changes were made in order to make collision as accurate as possible. Finally, the collision event listener was called which is provided by Corona and the function stated above was called when a collision occurred.

3.4.4 Accessibility

Documentation regarding development for Corona SDK was plentiful, but offered advice in a different way than Unity documentation. Tutorials were sparse in relation to beginning game development which meant issues and solutions could not be copied from another's source but instead required logic and understanding of a subject to fix an issue or design a solution to an aspect – there is no room for ignorance and therefore true logic and programming must be applied. For example, Corona provides documentation on how to draw text to the screen for the purposes of showing score etc. However, there is limited resources on how to apply this to collision and decrease score based on said collision. Using some simple logic, "Score: ", was drawn to the screen, a function was then created which will update the score variable every time a collision was detected. At first this did not work, logically it should of done but there was a key piece missing – the event listener. After some time, a listener was created which allowed the engine to 'listen' for every time the score was updated and thus the function worked correctly. This required logical thinking and comprehension – something which had been somewhat lost through drag and drop methodologies.

3.4.5 Heterogeneity

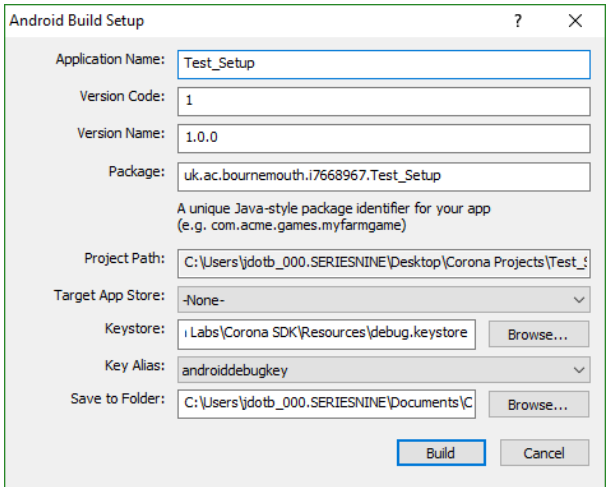


Figure 50. Build Setting within Corona SDK

Corona SDK has the ability to build applications for all mobile operating systems but primarily targets Android and iOS. For this implementation, the game was deployed to Android for reviewing and testing. Although compilation takes place online on Corona Lab servers, it can compile code at rapid rate. Making small changes within Visual Studio Code will be instantly applied to the simulator

and results of these changes are made apparent which makes the development process streamline. Due to the fact that changes are made by editing scripts it means that these scripts can be edited in any way which one desires without any delay and this is exactly what is required when making small, mobile games. Finally, build times are relatively quick, taking only about a minute due to the fact that Lua is a layer on top of C++ so translation is a simplified process. This means that small changes that require testing on actual devices can be carried out with ease which is required for independent mobile development.

4.0 Testing Results and Discussion

4.1 Testing methodology

To test performance of a mobile game it must be tested across different devices with varying scenarios. Due to the nature of video games being unpredictable two levels were created; one which plays as a normal game where the player must avoid hazards and another which tests extreme variables – this level features seventy enemies spawning at once and serves the purpose of performance testing not playability. Six devices were used for testing which vary in hardware and software in order to gauge a fair representation of user devices. The devices are comprised of four phones and two tablets in order to test scalability and performance on larger resolutions. In order to test performance on mobiles a benchmarking application is required.

Figure 51. GameBench Logo

Consulting benchmarking applications and forums in which others seek similar guidance on the most suitable application to use, the application which provides the most optimal service was ‘GameBench’. Unfortunately due to funding, only small tests can be run but this was sufficient in gauging performance for small 2D games. Each level was played for ninety seconds and the benchmarking application provides information on CPU, GPU and RAM usage regarding each level which in turn can give an accurate representation on performance and battery life usage.

4.2 Mobile Phone Testing

4.2.1 Samsung Galaxy III

Figure 52. Samsung Galaxy 3

The first device that was tested falls under the low end spectrum of the collection of devices that were tested. The ‘Samsung Galaxy SIII’ was released in 2012, it uses a dual core CPU which runs at 1GHz, is powered by Android 4.2 (Jelly Bean) and runs at a 480x800 resolution. Finally the Galaxy III uses a Mali-400 MP4 GPU but due

to the age of the device, benchmarking cannot take place on the GPU only CPU and RAM statistics can be measured.

The GameBench application was started and the Corona implementation was tested first followed by the Unity where each level was tested for a maximum of ninety seconds. Results are as follows:

Figure 54. Corona Test: Extreme Level

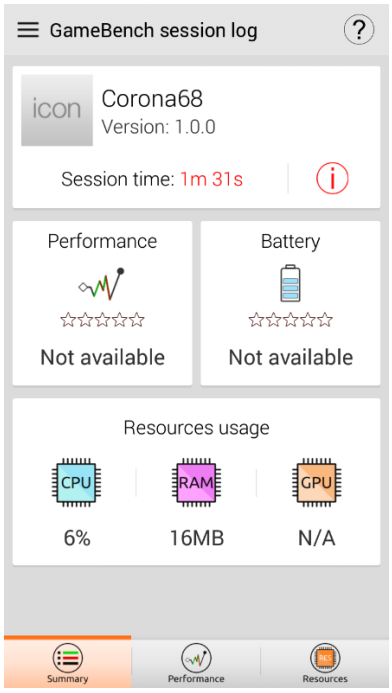
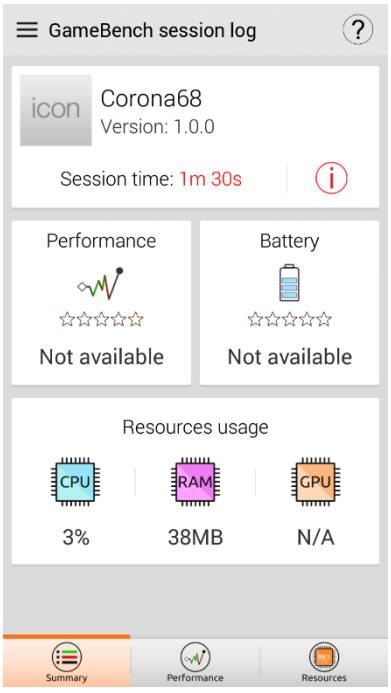


Figure 53. Corona Test: Normal Level



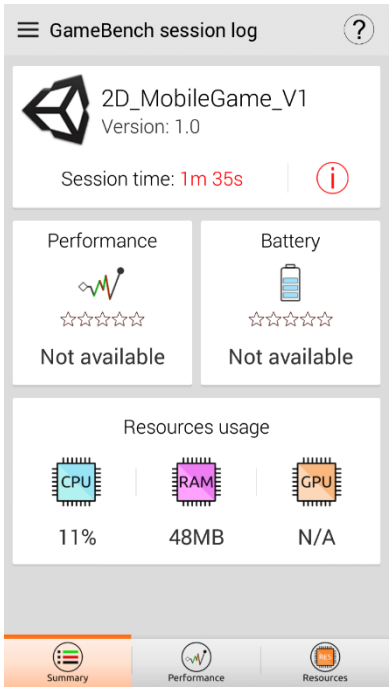


Figure 56. Unity Test: Extreme Level

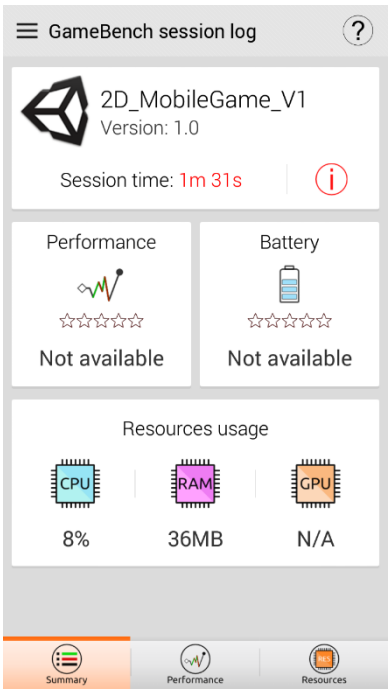


Figure 55. Unity Test: Normal Level

4.2.2 Sony Xperia L

The next device that was tested was the Sony Xperia L which is similar in specifications to the Galaxy III but with a slightly higher performance rating. Released in 2013, it utilises a 1 GHz Qualcomm MSM8230 dual-core CPU, 1GB of RAM and runs at 854x480 resolution. Like the Samsung Galaxy III it uses Android Jelly Bean but GPU statistics can be read and benchmarked. As previously stated, GameBench was run to gauge performance and the results show:

Figure 57. Sony Xperia L

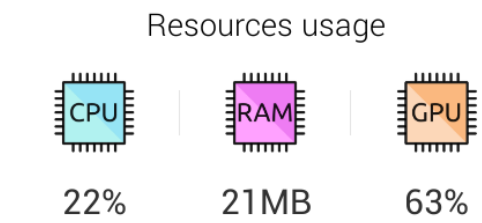


Figure 58. Corona Test: Extreme Level

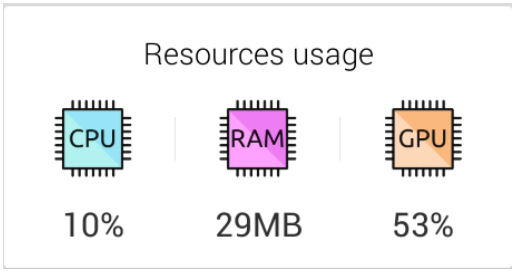


Figure 59. Corona Test: Normal Level

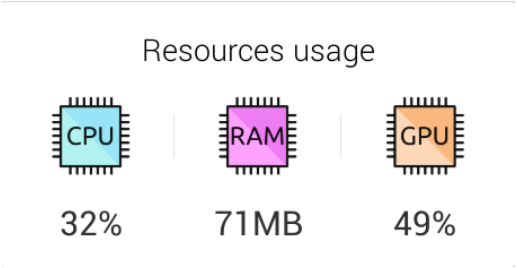


Figure 61. Unity Test: Extreme Level

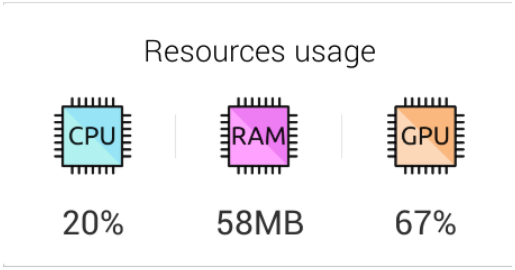


Figure 60. Unity Test: Normal Level

4.2.3 Samsung Galaxy S5

Figure 62. Samsung Galaxy S5

The Samsung Galaxy S5, two models above the Galaxy S3, was tested next. This was the first device which allowed screenshot feedback which meant that if there was a sudden drop in framerate performance one could see exactly where this happened. The phone itself was released in early 2014 and featured a new CPU which allowed a 2.5 GHz clock speed.

In addition, it has 2GB of RAM, a quad core system, and a 1080p display. It runs on Android 5.0 (Lollipop) which enables screenshot functionality when testing and also helps to understand how and why older models will perform differently. Results showed that both versions of the game used less resources overall which is to be expected with a newer device but performance still favoured Corona SDK:

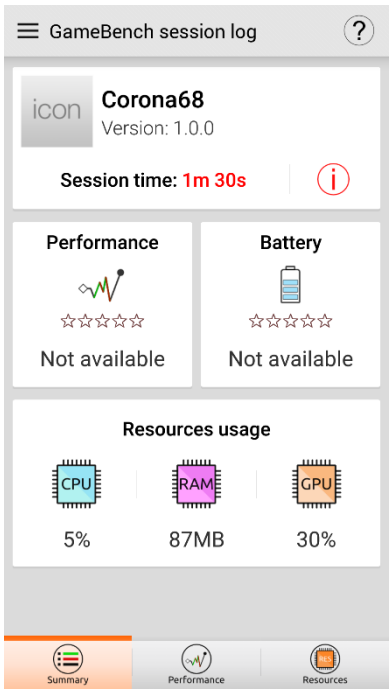


Figure 63. Corona Test: Extreme Level

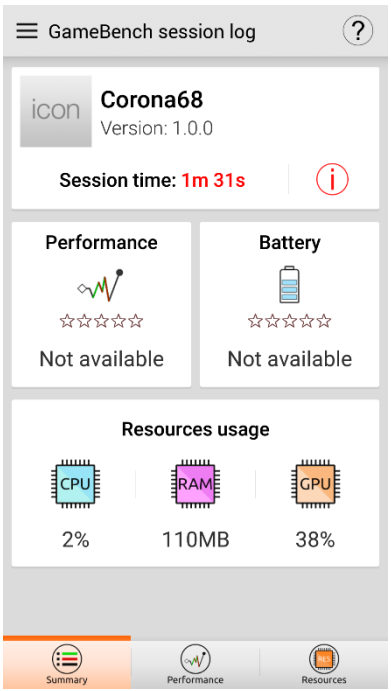


Figure 64. Corona Test: Normal Level

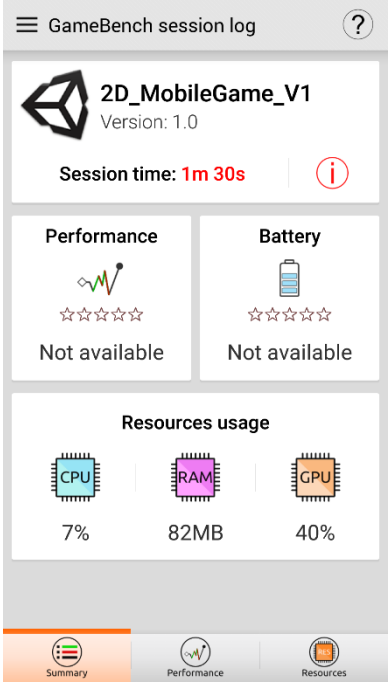


Figure 66. Unity Test: Extreme Level

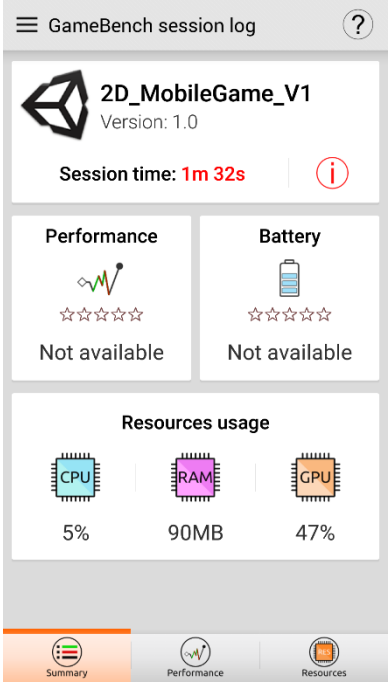


Figure 65. Unity Test: Normal Level

4.2.4 Huawei P9

Figure 67. Huawei P9

The final phone that was tested was the Huawei P9, which had a higher specification than the rest. Testing on this phone gives results which can be applied more accurately to what the most common types of phone being used by the majority of users. The initial release for the phone was in 2016. The P9 has Octa-core (4 x 2.5 GHz) CPU, 3GB of RAM, a 1080p display and runs on Android 6.0 (Marshmallow). This is the most up to date phone which was tested and as such gives an accurate representation of how the game would run on newer models of phone currently seen in the market. It is important to note that GPU statistics could not be measured due to the fact the operating system is relatively new and as a result GameBench has not been updated to support all features.

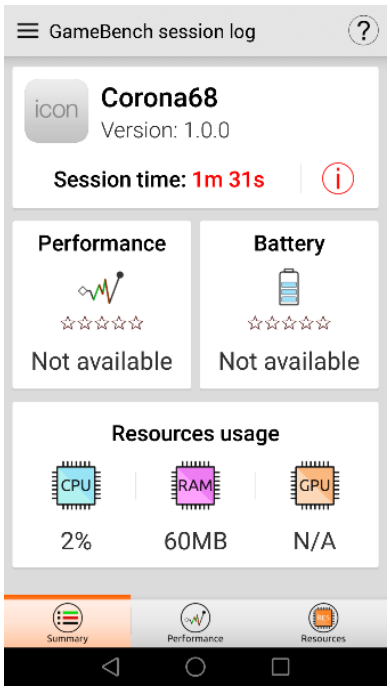


Figure 68. Corona Test: Extreme Level

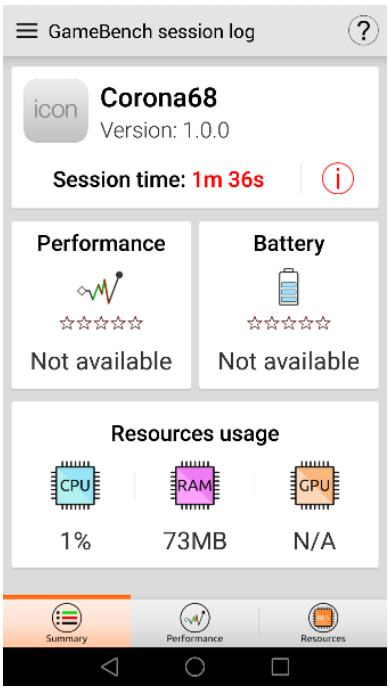


Figure 69. Corona Test: Normal Level

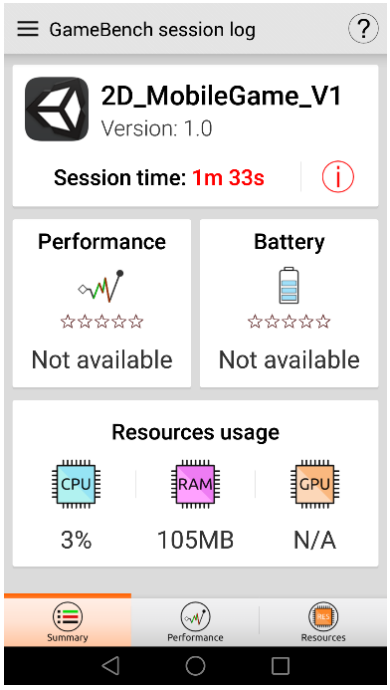


Figure 71. Unity Test: Extreme Level

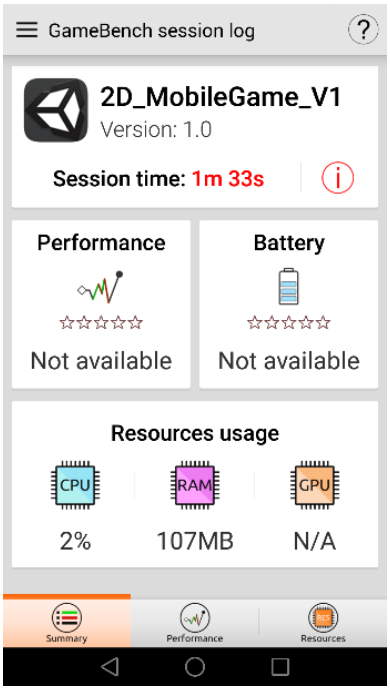


Figure 70. Unity Test: Normal Level

4.3 Mobile Tablet Testing

4.3.1 Google Nexus 7

Figure 72. Nexus 7

In order to gauge performance across devices, different types of device must also be tested. For the purposes of this scenario, two tablets were tested for performance – a low ended tablet and a slightly higher performing one too. The Google Nexus Seven was the first tablet to be tested. The device utilises a quad core CPU running at 1.3 GHz and a ULP Geforce at 4.16 GHz. The resolution is set at 1280x800 and can run up to Android 5.1. Testing this device is useful as the results can be applied to similar devices such as the iPad Mini which is one the most popular tablets currently being used. Results are shown below:

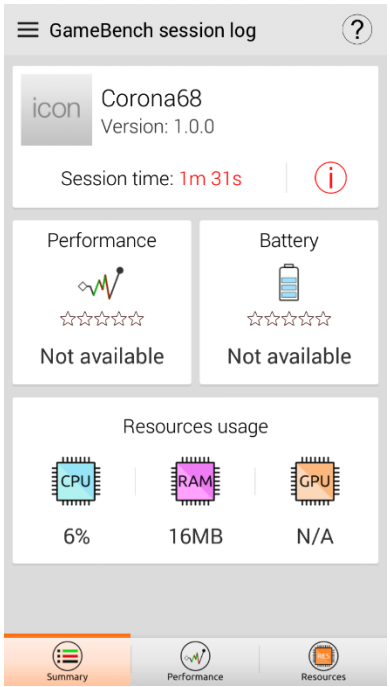


Figure 74. Corona Test: Extreme Level

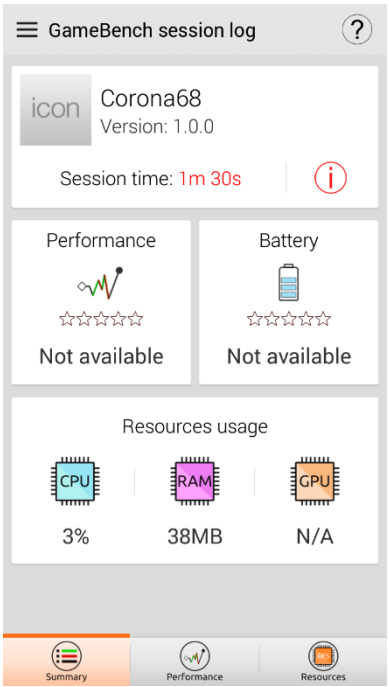


Figure 73. Corona Test: Normal Level

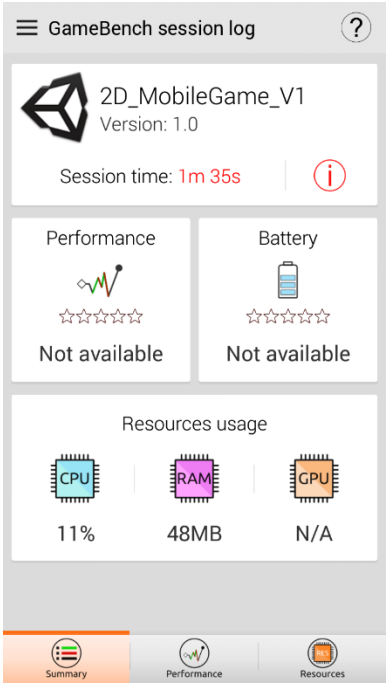


Figure 75. Unity Test: Extreme Level

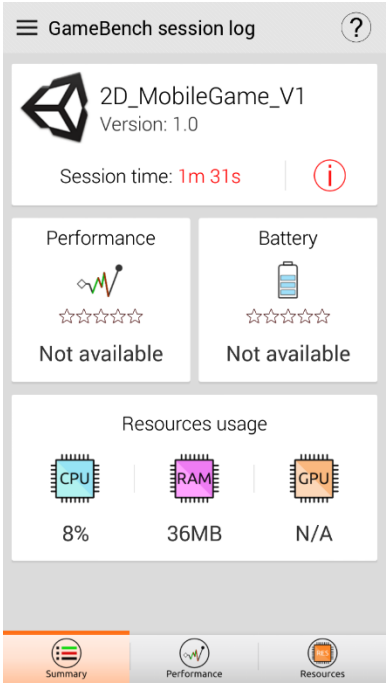


Figure 76. Unity Test: Normal Level

4.3.2 Samsung Galaxy Tab A

Figure 77. Galaxy Tab A

The final tablet and device that was tested falls under the high end category – the Samsung Galaxy Tab A. The tablet was originally released in 2011 and although it is not the highest performing tablet, it certainly holds some merit regarding performance. Running at 1920x1200, it is run on an Octa-Core 1.6GHz CPU with 2GB of RAM. It supports Android 6.0 and allowed for additional benchmarking utilities. It is the highest resolution tablet tested so the testing phase can provide some real world application in relation to other high resolution tablets such as older generation iPads/iPad pros.

Figure 79. Corona Test: Extreme Level

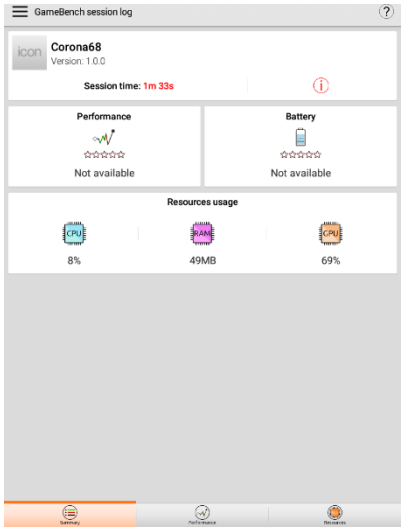


Figure 78. Corona Test: Normal Level

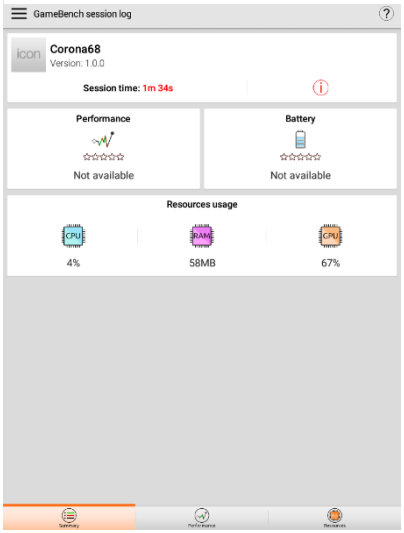


Figure 80. Unity Test: Extreme Level

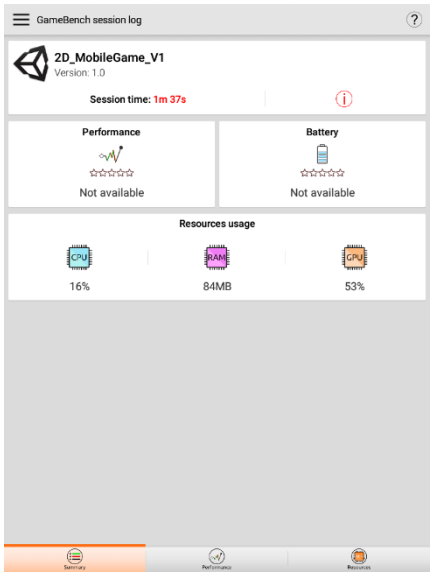


Figure 81. Unity Test: Normal Level

Figure 82. Corona Test: Performance Graph for extreme level

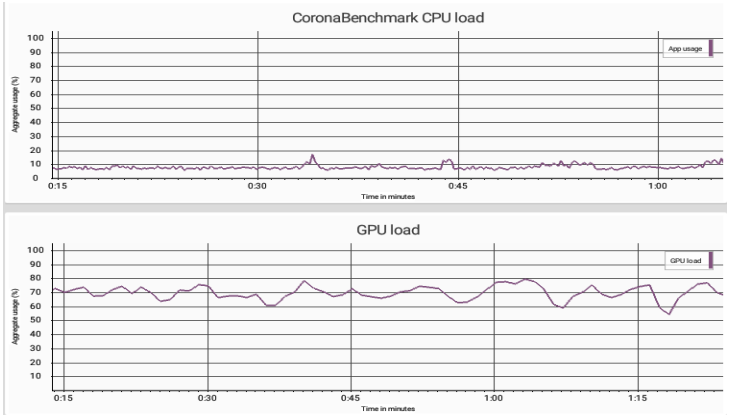
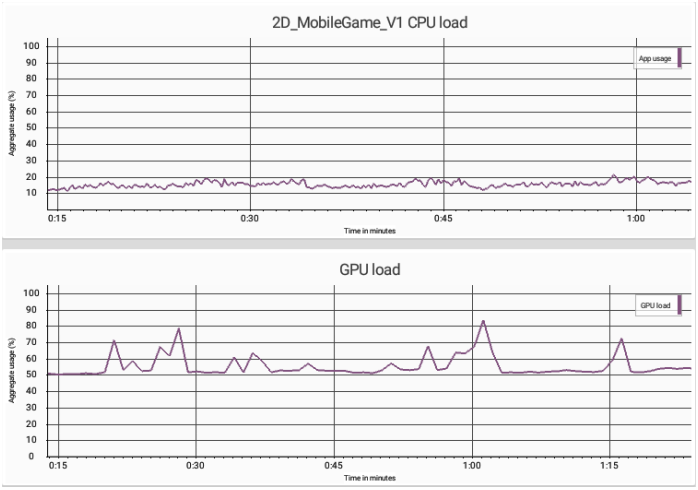


Figure 83. Unity Test: Performance graph for extreme level



4.4 Discussion

4.4.1 Evaluating testing environment

Before making a comparison between results found in testing, it is worth noting that not all results will be completely accurate due to other testing variables. Using the benchmarking application to test performance will cause memory to build whilst it is running so the longer testing was carried out, the higher memory usage. This is made apparent in some examples where RAM usage is significantly higher than previous results. In addition, some phones and operating systems will choose to use more of their CPU or GPU depending on what task is carried out so this is why in some cases CPU usage is low and GPU usage is high and vice versa. Due to the nature of testing some tests ran over the ninety second mark but this should not affect overall data as in the most extreme case timing was inaccurate by only five seconds. Finally, when

testing, the game was somewhat played in order to test all functionalities, such as moving the player to collide with enemies in order to test collision and score updating as well as navigating through the main menu.

To ensure a fair testing environment, the process of testing has been derived from literature relating to mobile application testing. Literature stated that in order to fairly test an application the tests have to reflect any extreme values that may appear – this is the reasoning for using an 'extreme' level as it can give an accurate representation of any extreme values or situations which may appear during regular gameplay. Device based testing was used as literature previously stated that it is suitable for performance testing whereas user testing is not related to the final outcome of this project. Finally, the choice to manually test was chosen for two reasons: lack of programming knowledge and time restrictions. Automatic testing may have offered less biased results but efforts have been made to ensure each test was relatively the same when tested manually.

4.4.2 Test Results Discussion

Results show that in all tests, the Corona implemented game performs more efficiently than the Unity implementation. Specifically, the Unity version consumes more CPU usage than the Corona and in some scenarios it doubles the amount of usage. A lot of this is due to the size of the Unity game, a 3D quad was used for the background effects which will make each frame substantially larger and as a result more performance is required. In addition, the very nature of the Unity engine means devices require more hardware power – it runs on a virtual machine built for Unity so it requires loading this and running it on top of a pre-existing one. An interesting observation is that for the normal level testing the Corona SDK version is substantially less demanding than the Unity version. This means that the Corona version did not perform better due to optimisation as the level is too simple to be optimised, meaning that Unity is innately more demanding which as a result will effect performance for smaller devices such as mobiles and tablets regardless of how well optimised it is.

Although RAM usage is affected by other factors such as how much memory has already been consumed and other applications running, it is apparent that the Unity version requires more RAM than the Corona application. In some cases where a lot of memory has been consumed, the RAM usage will be high for the Corona version but

for the majority of testing, the Unity version will require more. As stated before, this is mostly due to the nature of the Unity engine and applications built from it – it is becoming clear that although Unity may offer more platforms, sometimes quality trumps quantity. In a lot of scenarios, the Corona will use more of the GPU to run the game than the Unity version. However, this is not a negative observation as this is what the graphics chips in phones are designed for and as a result the CPU usage is lowered meaning switching between applications, performing advanced graphical techniques such as particles will perform more smoothly. This is re-enforced by the fact that higher end devices use more of their GPUs rather than CPUs and lower end devices are relatively the same – as technology advances it is important that graphical actions are performed by the GPU and it seems Corona SDK handles this change better than Unity built applications.

4.4.3 Summary

In summary, testing shows that for small mobile games Unity does not perform as well as other software; it consumes more RAM and favours CPU usage over GPU which in turn will drain battery life faster than other applications, make switching back and forth between other applications slow and will drop in performance quicker than other applications. Consulting the performance graphs, it is apparent that Unity is more sporadic in terms of hardware use which will result in greater FPS fluctuation, especially over a long period of time. Finally, the file size for the Corona SDK game was ten megabytes as an APK and twenty megabytes when installed whereas the Unity version was double this which made installation slower and consumed more hardware space – forty to fifty megabytes for this simple game appears overly large.

5.0 Conclusion

The aim of this paper was to determine whether or not Unity was suitable for student or start-up independent developers who are creating 2D mobile games. There was a lack of literature when consulting this subject as Unity has gained popularity recently and alternative non-engine methods are either used by developers with more programming experience or for specific genres/platforms. However, using previous research on comparison of other game engines and how to test mobile applications correctly, a fair methodology was formed in order to answer this aim. In addition, research gained from interviewing other independent developers helped understand what software is actually being used and the reasoning behind this choice.

Results show that Unity may not be the most practical solution for inexperienced developers to create 2D mobile games. Although it is used by a lot of developers and games can be made with in short time frames with relative ease, this does not mean it is the ideal game engine. The development process was relatively easy and without any prior experience one could create the same game within a matter of days. The main issue that arose through this fast development process was a lack of understanding. Creating games with Corona SDK enabled understanding that hadn't been achieved previously in education and personal use. Using logic to dictate how something should work rather than using a peer's work means there is full control over each aspect and this can be applied to a much larger scale – understanding typical programming and game development ideologies will support future development regardless of how one chooses to make it.

In addition, if one were to learn Unity throughout education and only Unity they would be limited in regards to actual knowledge of game development. For example, learning C++ with a graphical API will give a student understanding about the key fundamentals of how to draw an image to a screen and basic mathematics such as dot product etc. This knowledge is invaluable as it can be applied to any game development system whereas using Unity and allowing someone with more expertise to carry out complex tasks will only build ignorance towards more complex systems and as a result will restrict what they can do quickly.

To summarise, Unity or similar engines are great for producing prototypes at a rapid rate, however they can also limit understanding and as a result will produce similar

looking games. The choice of which software to use is somewhat subjective; it depends on which game one wishes to create, whether it is 2D or 3D and what core mechanics make up the game. It also depends on how much expertise is possessed and what tools have been used previously, with that being said, one should not dismiss a methodology if it seems too complex as a lot of the time it will not be. Using Unity is suitable for generating mobile games but amateur developers should not take it as the best option – consider more lightweight alternatives and reflect on the game they are making – in a lot of scenarios Unity is overly complex for the creation of mobile games and other solutions may be easier in the end.

Finally, if this research were to be used for future application then considerations must be made in regard to the testing methodology. The two implementations were created using the most basic solutions in order to gauge performance in regards to amateur developers. If this research were to be carried out with more advanced techniques and learning then general technique could be improved as well as optimisation techniques which would produce better results when testing performance. The application of this research is tailored for developers who are beginning game development and not those who already possess advanced game development skills and as such the conclusions drawn upon apply to this demographic only.

6.0 References

6.1 In Text Citations

[1] Newzoo. 2016. The Global Games Market Reaches $99.6 Billion in 2016, Mobile Generating 37% [online]. Available from: https://newzoo.com/insights/articles/global-games-market-reaches-99-6-billion-2016-mobile-generating-37/ [Accessed 30th April]

[2] Pierce, S., 2010. "Stickiness" in Games, or: Why you can't beat WoW. Gamasutra. Available from: http://www.gamasutra.com/blogs/ShayPierce/20100130/86533/quotStickinessquot_in_Games_or_Why_you_cant_beat_WoW.php [Accessed 4th May]

[3] Haas, J., 2014. A History of the Unity Game Engine [online]. Worcester Polytechnic Institute. Available from: http://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf [Accessed 20th April]

[4] Corona Labs Inc. Available from: https://coronalabs.com/corona-sdk/ [Accessed 20th April]

[5] Stackoverflow., 2012. Difference between framework vs Library vs IDE vs API vs SDK vs Toolkits? [closed] [online]. Available from: http://stackoverflow.com/questions/8772746/difference-between-framework-vs-library-vs-ide-vs-api-vs-sdk-vs-toolkits [Accessed 20th April]

[6] Stackoverflow, 2014. 2D cross-platform game engine for Android and iOS? [closed] [online]. Available From: http://stackoverflow.com/questions/12234457/2d-cross-platform-game-engine-for-android-and-ios [Accessed 20th April]

[7] Petridis, P.; Dunwell, I.; de Freitas, S.; Panzoli, D. (2010). An engine selection methodology for high fidelity serious games [online]. In Games and Virtual Worlds for Serious Applications (VS-GAMES), 2010 Second International Conference on (pp. 27-34). IEEE. Available from: http://ieeexplore.ieee.org/document/5460160/ [Accessed 25th April]

[8] Akekarat, P., 2014. Comparison and evaluation of 3D mobile game engines [online]. Master of Science Thesis (Masters). Chalmers University of Technology. Available from: http://publications.lib.chalmers.se/records/fulltext/193979/193979.pdf [Accessed 25th April]

[9] Kleinschmidt, C. and, Haag, M., 2016. Evaluation of Game Engines for Cross-Platform Development of Mobile Serious Games for Health [online], Volume 223: Health Informatics Meets eHealth [online], 207 – 214. Available from: https://www.ncbi.nlm.nih.gov/pubmed/27139405 [Accessed 26th April]

[10] Kleinschmidt, C. and, Haag, M., 2016. Evaluation of Game Engines for Cross-Platform Development of Mobile Serious Games for Health [online], Volume 223: Health Informatics Meets eHealth [online], 207 – 214, pages 213 – 214. Available from: https://www.ncbi.nlm.nih.gov/pubmed/27139405 [Accessed 26th April]

[11] Slant, NO DATE. Unity vs Corona-SDK [online]. Available from: https://www.slant.co/versus/1047/1064/~unity_vs_corona-sdk [Accessed 27th April]

[12] CoronaLabs, 2013. Corona vs Unity? [online]. Corona Labs Inc. Available from: https://forums.coronalabs.com/topic/35216-corona-vs-unity/ [Accessed 27th April]

[13] Renatus, 2014. Unity vs. Marmalade vs. V-Play vs. Corona vs. Cocos2D: Five Cross-platform Game Engines Compared [online]. Available from: http://renatus.com/unity-marmalade-v-play-corona-cocos2d-cross-platform-game-engines [Accessed 28th April]

[14] J. Gao, X. Bai, W. T. Tsai and T. Uehara, "Mobile Application Testing: A Tutorial," in Computer, vol. 47, no. 2, pp. 46-55, Feb. 2014. [Accessed 28th April]

[15] J. Gao, X. Bai, W. T. Tsai and T. Uehara, "Mobile Application Testing: A Tutorial," in Computer, vol. 47, no. 2, pp. 46-55, page 50, Feb. 2014. [Accessed 28th April]

[16] Amen, B.M; Mahmood, S.M and Lu, J. Mobile Application Testing Matrix and Challenges [online]. Available from: http://airccj.org/CSCP/vol5/csit53503.pdf [Accessed 29th April]

[17] Manouchehri, P., 2012. In-depth: Unit testing in Unity [online]. GamaSutra. Available from: http://www.gamasutra.com/view/news/164363/Indepth_Unit_testing_in_Unity.php [Accessed 30th April]

7.0 Bibliography

7.1 Additional Papers

Amalfitano, D., Amatucci, N., Memon, A., Tramontana, P., Fasolino, A., 2016. A general framework for comparing automatic testing techniques of Android mobile apps. Journal of Systems and Software [online], Volume 125, pages 322-343. Available from: http://resolver.ebscohost.com/openurl?sid=EBSCO%3aedselp&genre=article&issn=01641212&ISBN=&volume=125&issue=&date=20170301&spage=322&pages=322-343&title=The+Journal+of+Systems+&atitle=A+general+framework+for+comparing+automatic+testing+techniques+of+Android+mobile+apps&aulast=Amalfitano%2c+Domenico&id=DOI%3a10.1016%2fj.jss.2016.12.017&site=ftf-live&SToken=AyL2dOu-P-QDPOfEmg0JiydWOof1uhf01DGn1nijdSmDl0b7eITnVVmlMSehAA18Ulyisf96XzRvjnm3lqCK11gg6ayMmrcQWjCn6XlaqCQCZf0PCVFFDh5ObXQIdGTzjG_DsuzKjuSBdmV5HhhvYKHR0xSp4FfSfS4t2omKJ8tTjjqATC9W2H6hDaf7wYsk30aXSRwn23Hs-8o-zrh1qGbSE5ixvYNshn7AXqvCaFgbkHNz3cKU5Imx807oXN4ZEgZ3lRhdgSiGiUH95jlwNh_nAjddxA. [Accessed 30th April]

Barus, A., Tobing, R., Pratiwi, D., Danmanik, S., Pasaribu, J., 2016. Mobile game testing: Case study of a puzzle game genre [online]. Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT), 2015 International Conference on, Bandung: Indonesia 29-30 October. Available from: https://ieeexplore.ieee.org/document/7440194/ [Accessed 31st April]

De Macedo, D., Rodrigues, M., 2011. Experiences with rapid mobile game development using unity engine. Computers in Entertainment (CIE) - Theoretical and Practical Computer Applications in Entertainment, Volume 9 Issue 3, November 2011, Article No. 14. Available from: http://dl.acm.org/citation.cfm?id=2027460 [Accessed 29th April]

Ivanov, L. 2015. 3D game development with unity in the computer science curriculum. Journal of Computing Sciences in Colleges [online], Volume 31 Issue 1, pages 167-173. Available from: http://dl.acm.org/citation.cfm?id=2831402 [Accessed 30th April]

7.2 Webpages and Forums

Altos Adventure, 2015. Available from: http://altosadventure.com/ [Accessed 29th April]

Amuzo, 2017. Available from: http://www.amuzo.com/ [Accessed 3rd May]

Asus, 2013. Nexus 7. Available from: https://www.asus.com/uk/Tablets/Nexus_7_2013/ [Accessed 2nd May]

Blueprint Games, 2017. Available from: http://www.relapse-game.com/ [Accessed 3rd May]

Corona Labs Inc, 2017. Available from: https://coronalabs.com/corona-sdk/ [Accessed 4th May]

Gamebench. 2015. Available from: https://www.gamebench.net/ [Accessed 1st April]

Hollemans, M., 2014. How to Make a Game Like Candy Crush: Part 1 [online]. Raywenderlich. Available from: https://www.raywenderlich.com/66877/how-to-make-a-game-like-candy-crush-part-1 [Accessed 29th April]

Huawei, 2015. HUAWEI P9. Available from: http://consumer.huawei.com/uk/mobile-phones/p9/index.htm [Accessed 2nd May]

Makinggames, Weber, S. 2015. Unreal vs. Unity – Which engine is better for Mobile Games? [online]. Available from: http://www.makinggames.biz/feature/unreal-vs-unity-which-engine-is-better-for-mobile-games,8472.html [Accessed 3rd May]

Mullis, A., 2016. The Beginner's Guide to Android Game Development [online]. Android Authority. Available from: http://www.androidauthority.com/the-beginners-guide-to-android-game-development-692253/ [Accessed 3rd May]

QuizTix Game. Available from: http://quiztix.co/

Samsung, 2012. GT-I9300MBDBTU (Galaxy S3). Available from: http://www.samsung.com/uk/smartphones/galaxy-s3-i9300/GT-I9300MBDBTU/ [Accessed 2nd May]

Samsung, 2014. SM-G900FZKABTU (Galaxy S5). Available from: http://www.samsung.com/uk/smartphones/galaxy-s5-g900f/SM-G900FZKABTU/ [Accessed 2nd May]

Samsung, 2016. SM-T585NZKABTU (Galaxy Tab A). Available from: http://www.samsung.com/uk/tablets/galaxy-tab-a-10-1-2016-t585/ [Accessed 2nd May]

Sony, 2013. Xperia™ L. Available from: https://www.sonymobile.com/global-en/products/phones/xperia-l/ [Accessed 2nd May]

Stackoverflow, 2015. Unity2d vs. Corona [online]. Available from: http://stackoverflow.com/questions/28278244/unity2d-vs-corona [Accessed. 1st May]

Static Games. Available from: http://www.static-games.co.uk/ [Accessed 5th May]

Unity Forums, 2014. Unity or Corona SDK [online]. Available from: https://forum.unity3d.com/threads/unity-or-corona-sdk.283271/ [Accessed 3rd May]

Unity Technologies, 2017 Available from: https://unity3d.com/ [Accessed 5th May]

7.3 Images and Outstanding References

Coherent Labs, 2013. Unity 3D Facebook integration with Coherent UI (tutorial) [online]. Available from: http://coherent-labs.com/blog/unity-3d-facebook-integration-with-coherent-ui-tutorial/ [Accessed 3rd May]

Corona Project Manager, 2012. Available from: https://coronaprojectmanager.com/ [Accessed 4th May]

Dotan, G. 2015. *Unity Infographic* [online]. Available at: http://blog.soom.la/2015/01/unity-infographic.html [Accessed 25th April]

Google Play Store, 2017. Candy Crush Saga [online]. Available from: https://play.google.com/store/apps/details?id=com.king.candycrushsaga&hl=en_GB [Accessed 5th May]

LinkedIn, 2015. Why to Develop 2D Games for Mobile Instead of 3D Games? [online]. Available from: https://www.linkedin.com/pulse/why-develop-2d-games-mobile-instead-3d-ahmad-hammad [Accessed 5th May]

STATISTA, A.R., 2017. Statistics and facts on Mobile Gaming [online]. Available from: https://www.statista.com/topics/1906/mobile-gaming/ [Accessed 10th April 2017].

Technologies, U. 2016. *Unity - fast facts [online].* Available at: https://unity3d.com/public-relations [Accessed 25th April]

Ukie Nesta. 2016. *The UK Games Industry [online].* Available at: https://gamesmap.uk/#/companies [Accessed 25th April]

VERTO ANALYTICS, C.H., 2016. The most popular mobile game genres: Who plays what, when? [online]. Available from: http://www.vertoanalytics.com/the-most-popular-mobile-game-genres-who-plays-what-when/ [Accessed 10th April 2017].

7.4 Game Assets

Clipart Kid, 2016Cute Starfish Clipart. Available at: http://www.clipartkid.com/cute-starfish-clipart-clipart-panda-free-clipart-images-HQdDou-clipart/ [Accessed 20th April]

MyFonts, 2007. Jupiter. Available at: https://www.myfonts.com/fonts/canadatype/jupiter/ [Accessed 22nd April]

OpenGameArt, Jull, 2015. Available at: https://opengameart.org/content/animated-spaceships [Accessed 15th April]

Pixelnet Studios, 2013. Tutorial: Creating a 2D game with Unity. Available at: http://pixelnest.io/tutorials/2d-game-unity/shooting-2/ [Accessed 20th April]

Unity 3D, 2014. 2D Scrolling Backgrounds. Available at: https://unity3d.com/learn/tutorials/topics/2d-game-creation/2d-scrolling-backgrounds [Accessed 18th April]

Yerleske-Campus, NO DATE. bubbles transparent png. Available at: http://yerleske-campus.info/bubbles-transparent-png [Accessed 16th April]

8.0 Appendix

8.1 Interview Questions and Answers

8.1.1 Blueprint games

Final Year Project interview questions, regarding the implementation of a mobile game using Unity and Corona SDK for independent developers Blueprint Games

How big is your team?

We currently have a team size of 7. 2 full-time graduates, 4 students who are currently on their placement year and a project manager.

How many years of experience have you got in this field?

Nearly 3 years. Both I and Katie started our own company during our placement year. After we graduated we decided to continue with that and have been working on our first game since 2015.

How many games have you developed?

We released a small prologue in February 2016. This acted as an introduction to the full game we're working on now.

How long does it take to develop a game on average?

It depends on the size of the game and the team. I would say on average it would take 18 months to develop a mobile game.

What methodology did you choose to develop most of your games?

We chose Unreal Engine 4

Reasons behind why you chose this methodology?

Unreal Engine 4 is a very powerful tool. It includes all the SDKs for every platform that allows you to develop for Windows, Mac, Consoles and Mobile. We used the Unreal Engine during University so we gained a lot of experience with it. It also doesn't require a subscription to use.

Do you have much experience with other engines/different ways of developing?

Not really. We have only ever used the Unreal Engine. It's what we used at University and we have never had a reason to explore other options.

What is your opinion on young developers using engines when creating indie mobile games?

If the developer is serious about game development, I think they should be using an engine. It's a great source of information and learning experiences. Also many game studios require experience with an engine.

Would you agree that creating a mobile game in Unity is easier than using an SDK?

I imagine an SDK such as Corona would be easier to use but it depends on the user. SDKs are perfect for people who may not have the experience, knowledge or time to learn to use a full game engine. SDKs generally focus on a single type of genre such as 2D platformer or RPG and therefore could make games look identical. Full game engines offer a wide range of styles from realistic rendering to stylise art.

When creating mobile indie games do you think one should learn how to create games using an SDK rather than a game engine?

It depends on the type of game and the requirements. Before you even start to develop a game, you should detail exactly what you want in the game and how you want it to work. Then you should decide if an SDK will meet your needs. If it does, then there is no harm in using one. Using a full game engine gives you the flexibility to expand the game should you require.

Do you think a steeper learning curve can be more beneficial to a beginner developer rather than being able to create games with a more drag and drop ideology?

Not really. There is a danger of overwhelming a beginner with pure code. I think starting with a drag and drop ideology/visual scripting will help make coding more understandable. It uses the same logic as code, it's just used in a different way. Going from drag and drop to coding will be faster than just learning to code.

What language would you recommend learning first when beginning game development?

I would recommend C++ first then maybe moving to C#. It depends on the engine they want to specialise in though. Unreal Engine uses C++ but Unity uses C#. C# evolved from C++ so it would be easier to go from C++ to C# as opposed to the other way around.

What were the hardest things you found when first staring game development?

Setting up the engine to allow for collaboration. Generally, just working out the proper workflow for the engine and content management.

What tools do you use and how?

We use Unreal Engine 4. Visual Studio Community 2015 for coding and packaging the game. Perforce for source control so all members of the team can work on the game at the same time. 3DS Max and Maya for modelling, Substance Painter/Designer/Photoshop for texturing and Agisoft for Photogrammetry.

What platform is easiest to build for release and which gets the most downloads?

As we are using the Unreal Engine we can develop for all platforms including new platforms such as VR and Nintendo Switch. We haven't released a multiplatform game yet but I would assume Windows would get more downloads (depending on genre).

How easy do you find updating your games using your current methodology of developing?

Unreal Engine is continually getting updated with the latest SDKs and the latest features. Our project can be updated at a click of a button.

8.1.2 Static Games

Final Year Project interview questions, regarding the implementation of a mobile game using Unity and Corona SDK for independent developers Static Games

1. **How big is your team?**
   Three people, primarily one person developing games
2. **How many years of experience have you got in this field?**
   Three business years – prior studying at university where he learned mostly C#
3. **How many games have you developed?**
   15 games total mostly designed for web but also iOS and Android
4. **How long does it take to develop a game on average (primarily aimed at mobile developers)**
   Two weeks to about three months – depends on what the game is and how you make it
5. **What methodology did you choose to develop the majority of your games? E.g. game engine, SDK etc.**
   Started with UDK, moved to Construct 2. Some Unity but mostly construct 2 which uses C# and little programming
6. **Reasons behind why you choose this methodology?**
   Universal, efficient – quick prototyping
   Easy to use
   Can be built easily to multiple platforms
   Runs on HTML 5
7. **What is your opinion on young developers using engines when creating indie mobile games?**
   A good starting point, why reinvent the wheel?
   If you can make an engine yourself that's good but Unity and other engines are tried and tested methods
8. **Would you agree that creating a mobile game in Unity (or a complete game engine) is easier than using an SDK, if so why do you think this?**
   a. What are the downsides to using Unity (engines)?
      Cost

C# is a big language to start
What are the upsides of using Unity (engines)?
Freedom of control
Tried and tested
Easier than creating something new

9. **When creating mobile indie games do you think one should learn how to create games using an SDK rather than a game engine?**
   Tangible results
   Engines primarily – understanding logic
   Logic first
   Understanding the logic of how certain things work applies across all methodologies. So understanding how certain features work, e.g. Collision functions will not only help in engine development but also game development as a whole
10. **If yes to 12, would you agree that creating games with a pure code solution requires a steeper learning curve as opposed to using a game engine like Unity?**
    Results are easier to manage using something with a GUI
    Can understand things from these results
    Good place to form a base
11. **Do you think a steeper learning curve can be more beneficial to a beginner developer rather than being able to create games with a more drag and drop ideology?**
    Throw someone in the deep end and they will eventually learn to swim or won't but this will scare of competition yet is risky.
    Understanding the logic behind the code and behind what is happening is key
12. **What language would you recommend learning first when beginning game development?**
    C# - or something
13. **What were the hardest things you found when first starting game development? What were the major stumbling blocks?**
    Where do I start? This help breaks down the game
    Scaling is a big issue
    Building to multiple platforms
    Running games on older devices
    Rotation
    Building to iOS
14. **What tools do you use and how? (IDE, Source Control etc.)**
    Back up everything
    Bitbucket/git
    Visual studio with Unity
    Notepad ++
15. **What platform is easiest to build for release and which gets the most downloads?**
    Android is easiest to build to
    IOS gets most download

16. **How easy do you find updating your games using your current methodology of developing?**
    Cacoon.io

The interview went well, Owen was using Construct 2 to develop his games as this meant it was easy to build to web platforms which seems to be their main focus as a company. He felt using an engine was a good thing, it forms a good place to build skills and knowledge before branching into something bigger. The negatives were things that had been discussed already – the pricing, lack of control, performance on mobiles and it limits the skill set of the developer. Overall, he was an advocate for using Unity for reasons such as ease, low skill, and efficiency but also noted that if you can make games in a native language or using SDK/libraries then do that. Once again, the choice appears to be subjective for the type of game you are making and type of developer you are.

Shaders within Unity are important but complex – something which is overlooked by young developers but it very important!

8.1.3 Ian Masters

Final Year Project interview questions, regarding the implementation of a mobile game using Unity and Corona SDK for independent developers
Ian Masters

1. **How big is your team?**

   Three major people on the team – very small team operating mostly from home or public locations
2. **How many years of experience have you got in this field?**
   Making games since 1998 – developed apps for PlayStation called *LocoRoco* and branched out into independent development (look up more history!)
   Mostly using Java stuff
3. **How many games have you developed?**
   40 Games made so far (roughly and these are small 2D mobile games)
4. **How long does it take to develop a game on average (primarily aimed at mobile developers)**
   Varies per game obviously – there is no average
   40 percent, 7 days – 20 percent – month – 10 percent – this is CPI and it is important for games being released. Cost per install basis
   LTV – life time value. Life time value is basically how much money the game will make in its life time. How much does it cost to gain a user compared to how much it will cost to keep a user
5. **What methodology did you choose to develop the majority of your games? E.g. game engine, SDK etc.**
   Corona SDK
6. **Reasons behind why you choose this methodology?**
   Cons
   Slow updates
   2D games – performance

Multiple platforms means updates for a lots of platforms

Pros

Unity has a much larger influence and support

Market share is dominated by Unity

The key thing here is subjectivity – does your game need an engine or can it be written for a specific platform. In addition, it is based on developer too, are you confident with Unity or can you use C++ etc.

7. **Do you have much experience with other engines/different ways of developing?**
   Not a huge amount – it appears as though once you have stuck to a development route this can be used progressively throughout and only changes if the game platform changes e.g. Corona isn't used for consoles but Unity is
8. **Does team size affect this choice?**
   SDK/Native – are you making a game for a specific platform?
9. **What is your opinion on young developers using engines when creating indie mobile games?**
   Good thing, but limits their knowledge and applying to a games company could prove difficult as there is a lack of knowledge
   Dependent on the game itself and developer
   Limits skill set
10. **Would you agree that creating a mobile game in Unity (or a complete game engine) is easier than using an SDK, if so why do you think this?**
    Easier but this is not a bad thing! You can make games quickly, easily and use a market share dominating platform to advertise your game
    b. What are the downsides to using Unity (engines)?
    c. What are the upsides of using Unity (engines)?
11. **When creating mobile indie games do you think one should learn how to create games using an SDK rather than a game engine?**
    Cross platform usage means using Unity is preferable
    However, if you know more than you are more likely to be hireable so learning extras is very useful
    Breaking into the market requires a game which is fun – performance is not the massive issue so it is more about how each implementation handles large or unique circumstances rather than overall performance
    Sure – each version will run fine as is but what about when adding 1000 enemies? These are plausible situations which must be addressed
12. **If yes to 12, would you agree that creating games with a pure code solution requires a steeper learning curve as opposed to using a game engine like Unity?**
    Yes – and this can help when applying for larger companies
13. **Do you think a steeper learning curve can be more beneficial to a beginner developer rather than being able to create games with a more drag and drop ideology?**
    Again, can apply to a large amount of people with varied skill set but Unity is great for quick and easy development

14. **What language would you recommend learning first when beginning game development?**
    Subjective – is there a specific game you want to make?
15. **What were the hardest things you found when first starting game development? What were the major stumbling blocks?**
16. **What tools do you use and how? (IDE, Source Control etc.)**
    Bitbucket, source tree, sublime text, inkscape, gimp, image magic, spine 2d animation tools
17. **What platform is easiest to build for release and which gets the most downloads?**
    Google Play – alpha and beta channel – easier to test and analyse and fire TV (amazon) iOS is fourth!
18. **How easy do you find updating your games using your current methodology of developing?**
    Updating happens across all platforms so must be consistent

Using Unity is good for independent developers to release for cross platform in a quick and easy manner. The only issue which seemed apparent was limitation to one toolset and one language – if you were recognised by another company what else could you do besides c#?

The conclusion of this interview was that development routes depend on what you are developing (mobile – 2D or 3D? console?), who you are, what skills do you have? What are your future plans? Do you plan to always be independent?

Certain companies (Supercell) dominate the market due to being able to know what the user wants, calculate how much they need to spend to gain and keeps these users and in turn can be in the top 10 games on the game stores due to the money they have previously made. They can advertise to large amount of people in the right way and as a result will always dominate the market because they can do it better every time – they key is knowing how to advertise to users and having the money/team to do so

The choice for software is based on the game you make, e.g. a side scrolling or infinite runner type game can be made in Unity – a 2D game such as quizzes or match three type games can be made using an SDK or a lighter package. However, do you want to make the game yourself? Are you an independent developer if you use Unity to create games or are you jumping on a band wagon with the rest?

8.1.4 Karsten Pedersen (Amuzo)

Final Year Project interview questions, regarding the implementation of a mobile game using Unity and Corona SDK for independent developers
Karsten Pedersen

---

1. **How many years of experience have you got in this field?**
   The cleaner, research engineer, Amuzo

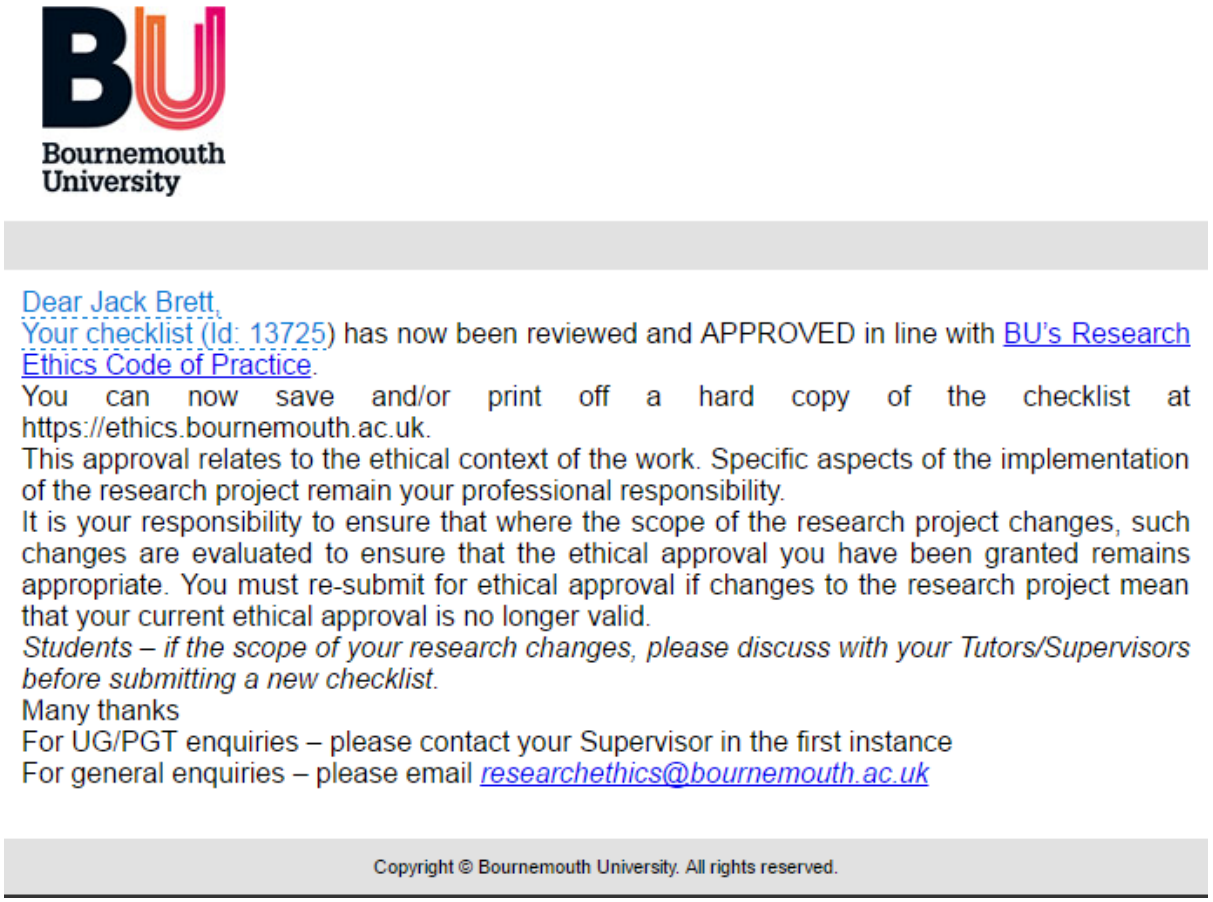
2. **What methodology did you choose to develop the majority of your games? E.g. game engine, SDK etc.**
   Small mobile games – use mainly flash
   Flow driven; from a mock up to prototyping
3. **Do you have much experience with other engines/different ways of developing?**
   UE4, Unity, Quake 3 – custom built
4. **What is your opinion on young developers using engines when creating indie mobile games?**
   Overkill, gives the developer a false view – set goals too high, and will be disappointed when they get nowhere. Adding complexity to an already simple task
5. **Would you agree that creating a mobile game in Unity (or a complete game engine) is easier than using an SDK, if so why do you think this?**
   Easier to use a library (SDK etc.) because it is simple, everything is exposed which makes debugging etc. easier on a small scale.
   d. What are the downsides to using Unity (engines)?
      Understanding the learning curve, limited to knowledge, maintaining the game afterwards is tricky
   e. What are the upsides of using Unity (engines)?
      A common base to work from, universal system – deprecation. A specification rather than a software package
6. **Would you agree that creating games with a pure code solution requires a steeper learning curve as opposed to using a game engine like Unity?**
   No, the libraries are simple and descriptive, whereas Unity has more of an immediate visual appearance. UE4 blueprints are better – visual scripting
7. **Do you think a steeper learning curve can be more beneficial to a beginner developer rather than being able to create games with a more drag and drop ideology?**
   Using Unity allows you to jump far but lacks the understanding
8. **What language would you recommend learning first when beginning game development?**
   C++, opens up any platform – universal language
   C# is Java competition; beat C++
   You learn everything you need to learn
9. **What tools do you use and how? (IDE, Source Control etc.)**
   Linux, OpenGL with Notepad ++
10. **What platform is easiest to build for release and which gets the most downloads?**
    Can't port Linux or older OS with Unity but Android is most popular

Rushing towards your goal rather than a slow learning curve leads to a brick wall with complex issues – shaders are a good example. Try other methodologies; if you haven't tried other ways of developing then you will never know.

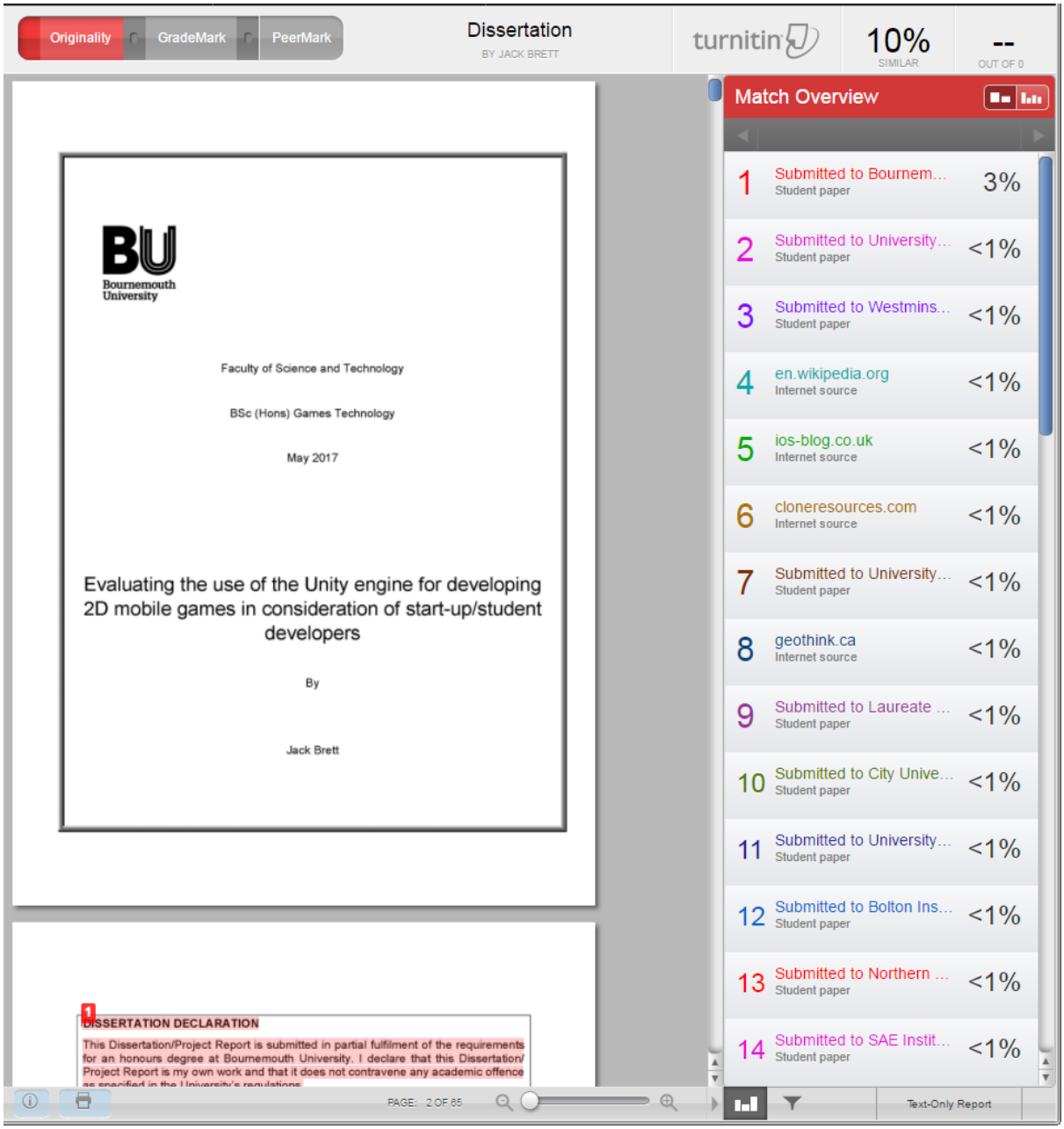
Unity:

- "A toy implementation"

- Multiplatform is important, but Unity does it wrong – limited
- Generic tools lacks optimisation



Dear Jack Brett,
Your checklist (Id: 13725) has now been reviewed and APPROVED in line with BU's Research Ethics Code of Practice.
You can now save and/or print off a hard copy of the checklist at https://ethics.bournemouth.ac.uk.
This approval relates to the ethical context of the work. Specific aspects of the implementation of the research project remain your professional responsibility.
It is your responsibility to ensure that where the scope of the research project changes, such changes are evaluated to ensure that the ethical approval you have been granted remains appropriate. You must re-submit for ethical approval if changes to the research project mean that your current ethical approval is no longer valid.
Students – if the scope of your research changes, please discuss with your Tutors/Supervisors before submitting a new checklist.
Many thanks
For UG/PGT enquiries – please contact your Supervisor in the first instance
For general enquiries – please email researchethics@bournemouth.ac.uk


Originality
GradeMark
PeerMark
Dissertation
BY JACK BRETT
turnitin
10%
SIMILAR
--
OUT OF 0
Bournemouth University
Faculty of Science and Technology
BSc (Hons) Games Technology
May 2017
Evaluating the use of the Unity engine for developing 2D mobile games in consideration of start-up/student developers
By
Jack Brett
1
DISSERTATION DECLARATION
This Dissertation/Project Report is submitted in partial fulfilment of the requirements for an honours degree at Bournemouth University. I declare that this Dissertation/ Project Report is my own work and that it does not contravene any academic offence
PAGE: 2 OF 65
Match Overview
1 Submitted to Bournem... Student paper 3%
2 Submitted to University... Student paper <1%
3 Submitted to Westmins... Student paper <1%
4 en.wikipedia.org Internet source <1%
5 ios-blog.co.uk Internet source <1%
6 cloneresources.com Internet source <1%
7 Submitted to University... Student paper <1%
8 geothink.ca Internet source <1%
9 Submitted to Laureate ... Student paper <1%
10 Submitted to City Unive... Student paper <1%
11 Submitted to University... Student paper <1%
12 Submitted to Bolton Ins... Student paper <1%
13 Submitted to Northern ... Student paper <1%
14 Submitted to SAE Instit... Student paper <1%
Text-Only Report