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

Jack Brett, Alain Simons,
Bournemouth University, Fern Barrow, BH12 5BB, Bournemouth, United Kingdom

{i7668967, asimons}@bournemouth.ac.uk

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.

Keywords: 2D game, Unity, SDK, Generic Software, Mobile game, Education.

1 Introduction

Today, mobile games consume a vast market share within the games industry, it is expected that in 2018 mobile gaming will account for 43 percent of the gaming market revenue and currently there are over a third of Americans playing mobile games daily [1]. In regards to genre; the most popular games are brain puzzle games (with over 37 million users per month playing one), closely followed by matching puzzle games e.g. *Candy Crush* [2]. 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 companies will be run by younger developers but it is safe to assume that at least a small portion of them are. Considering that nearly two billion

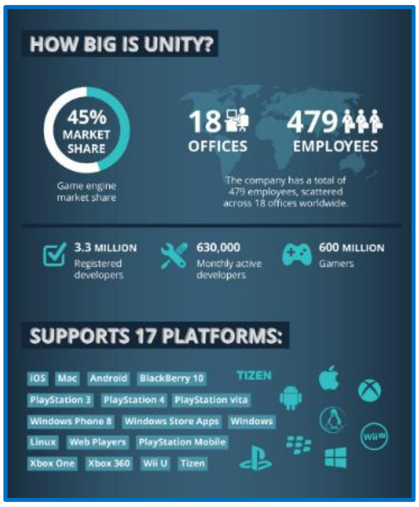


Fig 1. Dominant marketplace position of Unity.

mobile devices are running a Unity-made game, it is probable that quite a large majority of these new companies will be using Unity to develop mobile games.
It has been established that mobile games are becoming increasingly popular and that most users will play a puzzle or match game – a basic, more than likely, 2D game. The problem which arises is that new developers who want to create popular mobile games will assume that Unity is the correct tool for implementation. However, alternatives may offer simpler solutions but younger developers will be discouraged due to the ideology that using an SDK or graphical API is too complicated which may have been reinforced by using Unity through education.
The core objective of this paper is to determine whether Unity is the correct choice for younger 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 to compare performance and set a benchmark across a multitude of devices.

2 Previous Research

2.1 Software Comparison

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 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.
Game engine selection methodology has been defined by [3]. 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 in consideration for amateur developers as some methods will appear to be much easier but occasionally a steeper learning curve can be more beneficial.
A similar paper compared game engines which derived some of the criteria from Petridis' work [4]. 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 appears choosing software for development relied on two main things: experience of the developer and the type of game one wishes to create.

2.2 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 [5] 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 scalability, 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." [6]
Another decision must be made regarding mobile game testing and that is the choice to manually test or to automatically test. In [7] 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. Finally, [8] mentions that "that all code can be subject to change…" in order to test for performance or errors one must expect extreme values.

3 Our Research

3.1 Previous Comparison

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. 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.
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, we compare and analyse the two different implementations of a 2D mobile game, using Unity as the engine and Corona SDK.
Criteria to compare the two development:

- Audio visual fidelity which consists of mainly 2D sprite animation
- Functional fidelity which is scripting and language efficiency,
- Composability – import/export limitations and available content
- Accessibility**:** learning curve, documentation and support, licensing and cost
- Heterogeneity (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 animation)
- Basic enemy AI and a scoring system
- Collision with enemies and a life system
- Touch controls and basic player movement linked to touch controls

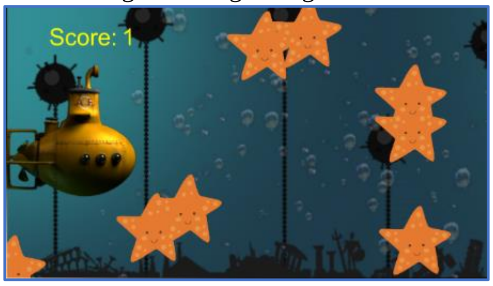


Fig. 2. Example of the game.

3.2 Unity Implementation

Creating the basics of a game within the Unity engine is an easy task; dragging the required assets from a folder and dropping them into the editor was simple and fast. Once assets had been imported, initial backgrounds were placed – here we used multiple images and layered them in front of one another. Parallax scrolling was implemented by transforming the sprite images into textures, placing them onto a 3D quad then attaching a script which offset the textures by a rate which is changeable. It is important to note that this process requires mathematics but due to the rich documentation, understanding said mathematics was not necessary as someone has already calculated these variables. Additionally, the default shader applied to this object was not changed – it was default and changing it meant learning a new library regarding shaders.
Creating the animating ships was carried out using sprite animation. The animator window within the Unity editor is simple and easy to follow; the main issue was once again, ignorance of what was happening. Dragging a sprite sheet with multiple images into the editor meant that Unity could create the animation and when the game played the animation begun – this may appear as a good thing but with over two years of using

Unity in education one should understand how sprite animation works in detail. Creating ‘enemies’ for the player to avoid involved creating a script which moved an enemy left to right with a sine wave for some random movement. Spawning these enemies was carried out by instantiating them within one function and using a Unity function which allows a specific function to be looped with a given time and rate. 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.

Scripting within the Unity editor can be a tedious task if only small changes need to be made. Using mostly Monodevelop (which is included with Unity) to edit code, making a small change then switching back to the editor to test the change took much longer than needed. 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.

3.3 Corona SDK Implementation

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. Using Microsoft Visual Code to edit the code meant that the project loaded instantly and changes could be made at rapid speeds. The initial set up of loading images and creating a layered background was the same as the Unity implementation only carried out using code rather than re sizing an image with a visual editor. Unlike Unity, Corona SDK only uses 2D libraries so parallax scrolling could not be carried out using texture offset of a 3D quad. In any case, a simpler solution was used. By placing an image on the screen and a copy just behind, a function could be set up which moved the first image then the second when it had reached the end of the screen – a process which relied on logic over demonstration.

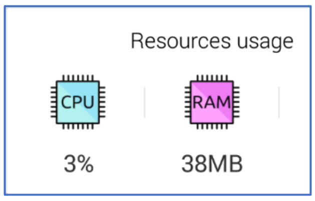
Creating sprite animation was rather straight forward due to the documentation on the Corona website. The idea behind sprite animation is to tell the engine the size of each individual image, for example, a square image of 128 pixels with four images would mean that each image is 32 pixels in size. Implementing touch mechanics in order to move the ship was carried out by using code found online but the basics of it are that the engine will recognize where a touch was started and a runtime event will ‘listen’ for where it was let go.

In terms of difficulty; there is certainly a steeper learning curve in comparison to using Unity but the experience and knowledge gained surpasses Unity. There is no room for ignorance and therefore true logic and programming knowledge must be used. For example, 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. Finally, making small changes such as spawn numbers etc. were compiled almost instantly and as a result made the development process much more streamline.

4 Results

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 70 enemies spawning at once and serves the purpose of performance testing not playability. Five devices were used for testing; from low end mobile phones to high end tablets, for this scenario the lowest end mobile, the mid-range mobile and the high end tablet will be used to demonstrate performance.

Fig 3&4. Corona SDK version performance results. Corona SDK version tested on *Samsung Galaxy SIII* (Low end device). Left shows normal level and right shows extreme values.



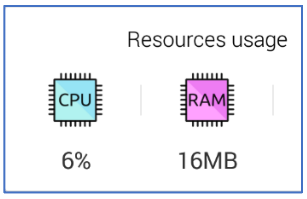
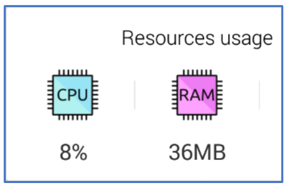


Fig 5&6. Unity version tested on *Samsung Galaxy SIII*. Left shows normal level and right shows extreme level.



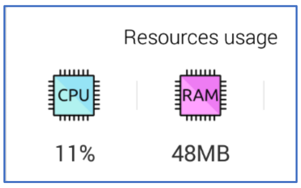
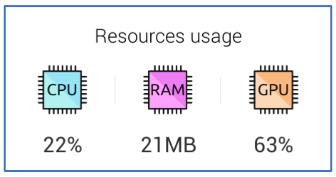


Fig 6&7. *Sony Experia L* Test. Both are on extreme value level, left shows Corona version and right shows Unity version.



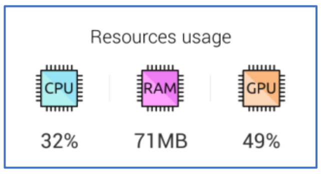
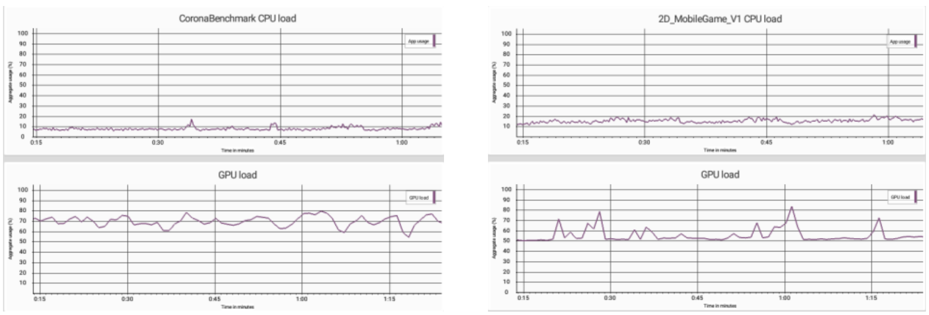


Fig 7&8. Performance graphs for both versions. Right shows Corona and left shows Unity. Testing on Samsung *Galaxy Tab A.* Both tests are performed on the extreme values level.



5 Conclusions

Results show that Unity may not be the most practical solution for inexperienced developers to create 2D mobile games. Although one may presume 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. Testing shows that for small mobile games Unity does not perform as well as other software; it consumes much more RAM and CPU usage which in turn will drain battery life faster than other applications and will drop in performance quicker than other games.

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, 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 very similar games. The choice of which software to use is somewhat subjective; it depends on which game one wishes to create and how much expertise is possessed. Using Unity is suitable for generating mobile games but amateur developers should not take it as the best option – one must consider more lightweight alternatives and reflect on the game they are making – in a lot of scenarios Unity is not necessary for creation of mobile games.

6 References

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

[2] 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 10 April 2017].

[3] 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 12 April].

[4] 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 12 April].

[5] 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 13 April].

[6] 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 13 April].

[7] 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 13 April].

[8] 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 12 April].

[9] Gamebench. 2015. Available from: https://www.gamebench.net/ [Accessed 14 April].