A Case Study on One-Source Multi-Platform Mobile Game Development Using Cocos2d-x

Jinseok Seo, Hun Choi

***Abstract*— In this paper, by introducing a case study on development of a first-person shooter game "Biosis" playable in both iOS and Android platforms, we present guidelines for developing one-source multi-platform mobile games using cocos2d-x game engine. This paper also describes the "ResourceMaker" implemented to share and manage game assets efficiently in our multi-targeted development environment and the level engine by using which game planners can easily apply their designs to game levels. We expect that the presented guidelines will help game developers reduce the time and cost for development in the mobile game ecosystem, the life-cycle of which is very short.**

***Index Terms*—cocos2d-x, mobile game, multi-platform**

I. INTRODUCTION

Recently, as the mobile platforms including smart phones have achieved popular success, the size of the mobile game market is also rapidly increasing [1]. As the market grows, more and more types of smart devices are emerging. Even on platforms that support the same operating system, many various types of devices with different screen resolutions are being announced.

Therefore, it is inevitable that the cost required to develop a game for various platforms and display types as described above is greatly increased. In addition, it is difficult to invest a great deal of money, because the lifecycle of mobile games is very short compared to PC and console games.

Due to the above reasons, many game engines have recently been released that can develop games for multi-platform at low cost. Multi-platform mobile game engines allow you to run a developed game on a variety of platforms without any porting process, they are gaining great popularity not only for major development companies but also for indie developers with relatively low budgets.

In this paper, we present the case study of developing a first-person shooter game called "Biosis" (see Fig. 1) that can be run on both iOS and Android operating systems, and suggest guidelines for efficient development that we have experienced.

The composition of this paper is as follows. Chapter II describes the mobile game engines for multi-platform, which are widely used. In Chapter III, we introduce the Cocos2d-x used in this study and explain how to develop one-source multi-platform games efficiently using this engine. In Chapter IV, we explain an introduction to Biosis, a one-sous multi-platform game developed in this study, and description of "ResourceMaker", a tool developed for efficient game resource sharing and management. This chapter also describes the level engine implemented to reflect the game designers' intention freely. Finally, Chapter V concludes the paper.

Jinseok Seo, Division of Digital Contents Technology, Dong-eui University, Busan, Korea

Hun Choi, Department of Electronic Engineering, Dong-eui University, Busan, Korea

This Work was supported by Dong-eui University Foundation Grant (2012)

Fig. 1: Title Screen of Biosis

II. MOBILE GAME ENGINES FOR MULTI-PLATFORM

Just before the popularization of mobile devices mainly based on smartphones, PDAs (Personal Digital Assistance) and Portable Multimedia Players (PMP) complemented the lack of functions of mobile. Because there were a variety of platforms and devices during this period, efforts were made to provide a cross-platform game development environment for PMP to save developers' effort.

As smartphones become very popular, many multi-platform game engines have been released. Some well-known examples are Unity [3], ShiVa [4], Corona SDK [5], Marmalade [6] and Cocos2d-x [7].

Unity3D is one of the most popular game engines in recent years. Unity was originally developed for 3D games, but recently it has also begun supporting 2D games. The biggest advantage of Unity is that the development environment is very comprehensive, so even the intermediate developers can easily develop high-quality mobile games. On the other hand, it is estimated that it works a little heavier than other game engines, and the lower level control is a little inconvenient. However, considering the recent increase in 3D game share in the mobile game market and the speed of improvement of GPU performance of mobile devices, Unity, which is a relatively inexpensive 3D game engine, is expected to become

the best 3D mobile game engine for small or medium-sized development companies or Indie developers.

Although ShiVa3D is not as widely used as Unity3D, ShiVa3D, which can be regarded as almost similar in function and convenience, has received much attention in recent years. Compared to Unity3D, there is a lack of user forums and documentation, but it is a little cheaper and is a great tool for developers who are accustomed to the Lua language.

Corona SDK is also a commercially successful game engine that can be developed in Lua language. Corona SDK was developed with OpenGL, OpenAL, Box2D, and Lua. Although it does not provide a convenient integrated development environment like Unity or ShiVa, it is used by many developers because of optimized performance.

Unity, ShiVa and Corona SDK described above use a method that executes a user-written script language (for Unity, the intermediate code compiled from C# language) in a pre-developed player engine. In recent years, this type of game engine has been widely used on most platforms, including PCs. This is because, as described in the well-known "90-10 rule (90 percent of the program execution time runs only 10 percent of the code)", implementing only minimal performance-sensitive parts in native code and implementing the rest using scripting languages does not affect the overall software performance. However, using this method has the disadvantage that it is inconvenient to implement the optimized algorithm only by developers or add customized functionalities to game engines.

Marmalade is a standard C++ language based game engine without the above drawbacks. It is very easy to optimize performance because it is implemented with 100 percent native code. Thanks to Marmalade's optimized performance, there are many commercially successful games developed using this game engine, such as "Cut the Rope," "Plants vs Zombies," "Call of Duty," "Need for Speed," and so on. If you are developing using only native code without the support of scripting languages, it is difficult to test or debug on the fly while the game is running. However, in the case of mobile games, the compiled code is not executed directly in the development environment, but is executed remotely in emulators or devices, the advantage of a dynamic scripting language is not great.

With the game engines, we've just described, you should pay for all your games to be released to stores or run directly on devices. Especially, Marmalade and Corona SDK can be a burden for indie developers and students who are learning to make games, because they are required to pay annually. In addition, all the above game engines cannot offer developers complete freedom because the source code is not available.

Cocos2d-x, a game engine used in this study, is a completely free engine, unlike the ones mentioned above, and the source code is open. Therefore, developers can not only modify released engine's code directly, but also add new functionalities freely. Of course, besides Cocos2d-x, there are more free game engines with open source code, including CuvicVR 3D Engine [8], IwGame Engine [9], jumpcore [10] and Mao [11]. However, nowadays it is hard to find an engine that has all the elements (audio, physics engine, particles, various font rendering, GUI, various types of maps, etc.) necessary to develop games as much as Cocos2d-x. A more detailed description of Cocos2d-x will be given in Chapter III, and the game engines described so far are summarized in Table I.

. Table I: Comparison Table of Multi-Platform Mobile Game Engines

Game Engine	2D/3D	Price	Open Source	Language
Unity	2D/3D	Free(personal) $125/m(pro)	X	C#
Shiva	3D	$200(basic) $1,000(adv.)	X	Lua
Marmalade	2D/3D	$149/y(basic) $1,499/y(pro)	X	C++
Corona SDK	2D/3D	Free(basic), $79/y $199/y	X	Lua
Cocos2d-X	2D	Free	O	C++
Unreal	3D	Free, 5% of grs. rev.	O	C++ Blueprint

III. Development of One-Source Multi-Platform Games Using Cocos2d-x

A. *Introduction to Cocos2d-x*

Cocos2d was originally developed as a game engine for a variety of desktop operating systems, including Windows, Mac OS, and Linux. Most of the functions for 2D games, such as scene flow and transition, easy and fast sprite processing, various actions, and tile map, are available by very intuitive APIs. At that time, the Python language was adopted, and the games could be developed quickly and easily without compilation process.

However, Cocos2d did not get much attention until smart phones started to gain popularity and Cocos2d for iPhone was developed. Cocos2d for iPhone was released as an engine to develop games for iPhone after Apple launched iPhone and start the App Store service. The basic engine architecture was adopted from Cocos2d, but the language was based on Objective-C. After several years of improvement, it became a more complete 2D game engine. Thanks to its completeness, the popularity has increased in recent years, so it is called by the name Cocos2d instead of Cocos2d for iPhone.

The biggest disadvantage of Cocos2d for iPhone was that it could only support games for the iPhone. The Cocos2d-x game engine has emerged because of this problem. Cocos2d-x also adopts the engine architecture of Cocos2d for iPhone, but since it is based on the standard C ++ language, it could be ported to various development environments and mobile platforms including Android. In addition, for the Android platform, JNI (Java Native Interface) and Android NDK (Native Development Kit) can be used in developing games.

B. *Sharing Source Code and Resources*

Although it depends on your target platform and development environment, Ccos2d-x is easy to use for most

development environments and operating systems. However, games for Android can be developed in any development environment, but those for iOS can only be developed in Mac OS. Therefore, we should build a development environment on Mac OS, in order to develop games for both platforms in one source code.

The most common development tools for developing Cocos2d-x games on Mac OS are Xcode and Eclipse. Depending on the developer's preferences and preferences, it may be more convenient to choose Xcode as your main development environment, considering the editing capabilities, auto-completion, and debugging of C ++ code.

Even if you use Xcode as your main development environment, you need to create a separate project for the Android platform to build the binary package for Android. In order to efficiently share source code and resources between the two projects, it is necessary to refer to the source code and the resources of other projects in one project.

The following is a sequence of steps for creating projects and sharing source code and resources for both iOS and Android projects.

1. Creating an Android project: Run the script "create-android-project.sh" in the folder where Cocos2d-x is installed to create the project.

2. Creating an Xcode project: Run the script "install-templates-xcode.sh" in the folder where Cocos2d-x is installed to copy template files for Cocos2d-x to an Xcode project. After running Xcode, create a new project with the same name you set in step A.

3. Removing source code and resources for Android project: Delete the "Classes" folder and the "Resources" folder from the project folder created in step A.

4. Setting for sharing source code and resources: In the Android project folder, create a symbolic link to the "Classes" and "Resources" folders in the Xcode project.

Once you have completed the four steps above, even if the source code and resource files are changed or added in Xcode, the main development environment, the binary package for the Android platform will be applied only when you compile again.

C. *Debugging*

The main development environment, Xcode, provides a very easy to use debugging environment compared to Eclipse, and with the help of the tool called Instruments, we can detect memory leaks very intuitively. In addition, the iOS simulator works very fast, allowing testing and debugging several times faster than running directly on the device.

Most of the time, when debugging is completed perfectly in Xcode and iOS Simulator, it works well on iOS devices and Android devices. However, because of the diversity of mobile devices, excessive resource usage can cause problems on certain Android devices, so you should perform validation on your Android device at the main milestone point.

Debugging on an Android device can use the "CCMessageBox" function to pop up a message box directly on the device, but it is much more convenient to error and warning messages using the "__android_log_write" function, because of the nature of the game software operating in an infinite loop. The Android system also allows logging to be broken down into different stages according to priorities, and filtering is also possible. You can use the "adb logcat" command for log filtering and error dumping.

D. *Various Aspect Ratios and Resolution Issues*

Various Android devices have their own screen sizes, screen aspect ratios and resolutions. In addition, recently, iOS has been releasing new products with varying ratios and resolutions as well.

The best method to support all these various display devices is to prepare graphics image files that matches the resolutions and ratios of all the devices you want to support. However, this method also has the disadvantage that the size of the final binary package becomes too large, and the game logic is also very complicated for supporting various aspect ratios and resolutions.

In order to solve the above problem, our study used a method of fixing the aspect ratio and preparing only two or three sets of graphics images for different resolution devices. The aspect ratio can be fixed by using the "setDesignResolutionSize" member function provided by the "CCEGLView" class of Cocos2d-x API. A "CCEGLview" object is obtained by the "getOpenGLView" function of the "CCDirector," a singleton object.

You should pass the horizontal and vertical resolutions as the first and the second arguments of this function, and pass the option flag as the third argument. Since the resolution information transmitted by this function is not an absolute resolution, it usually plays a role of fixing the aspect ratio based on the one having the maximum resolution among the devices to be supported. The third argument is usually the value of "kResolutionAll." This value allows the device to create black space on the edges of the screen without truncating the rendered image if the aspect ratio does not match the device's screen aspect ratio. The reason for using this value is that clipping the rendered image may occasionally drops the important graphical user interface.

The aspect ratio and resolution issue originally has separate solutions for each of Android SDK, iOS SDK, and Cocos2d-x. However, it is often the case that the solutions are changed again each time a new SDK version or a new device is released.

Although this problem is expected to be solved in a near future, this study decided to use a method that is not dependent on a specific SDK. We created a desired subfolder for each resolution under the Resources folder, and then put images with the same name but different resolutions into each folder. Then we specified the desired subfolder for each resolution with the "setResourceDirectory" function, which is the member function of the "CCFileUtils" class' singleton object.

IV. BIOSIS DEVELOPMENT CASE

A. *Introduction to Biosis*

"Biosis," a game developed in this study, is a touch-based first-person shooter, and target platforms are iOS and Android. Most smartphones do not have hardware buttons for

applications, so game players use software buttons to move their character and to fire weapons. Users can play games by touching the rendered image button in the corner of the screen with a finger. Because of this unintuitive interface approach, first-person shooter games are less popular than other game genres on smartphones.

In this game, we have devised a system that does not use the traditional interface method like above, but can attack objects such as monsters directly by touches or gestures. By using this attacking system, players can throw various weapons (grenades, rockets, bullets of shotgun, lasers, etc.) directly to targets. The levels consist of a total of 18 stages, of which four levels spawn bosses (see Fig. 2).

Fig. 2: Boss "Gargoyle" of Biosis

The player uses various weapons to attack the monsters (9 types in total). The player's life is set to the maximum value at the beginning of each level, and when the enemy attack reduces the life value to less than 0, the game ends. A variety of special weapon attacks using gestures consume player's energy, and we must wait for the energy to be replenished again to continue using special weapons.

B. Resource Management Tool: ResourceMaker

One of the most time-consuming aspects of game development process is managing various game resources. Some of the game resources are composed of files such as graphic images, background sounds, and sound effects, and some are composed of data such as strings, monster information, and weapon information. A resource consisting of a file is usually used by passing the path and the name directly as a parameter of a specific function in source code. A resource composed of data may be written in a specific script language or a data file of a developer's format.

In Biosis, we developed a resource management tool, "ResourceMaker," to store all information about resources in a single "plist" (property list) file. The "plist" file is a data structure that is originally supported by Objective-C, the main development language of Mac OS and iOS. It can manipulate data such as arrays, dictionary (maps), strings, and numbers in a hierarchical structure. Cocos2d-x is a C ++ language, but it also supports "plist" files in C ++ for compatibility with Cocos2d for iPhone.

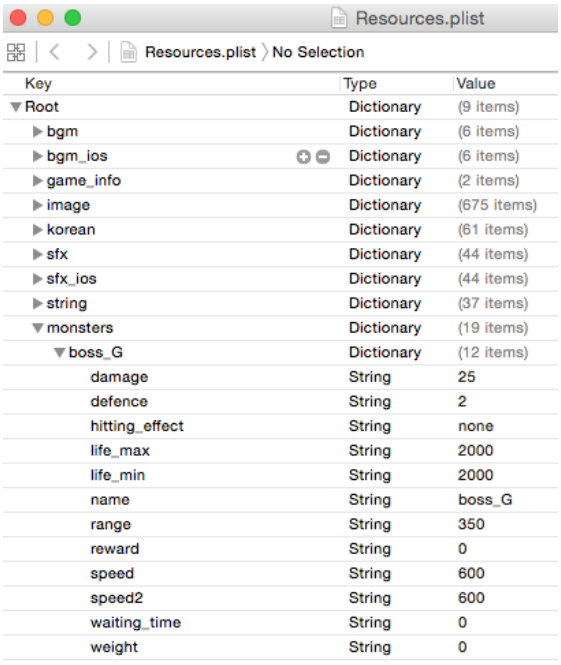

Fig. 3: An Example of "plist" file used in Biosis

To use the plist, we firstly place resource files, such as graphic images, background sounds, and sound effects, in their own subfolders ("image", "bgm", "sfx") under the project's "Resources" folder. Then, "ResourceMaker" automatically navigates to the project's subfolders and stores the resources' information in a "plist" file as a dictionary data structure. For example, if you have a "background_1.png" file in the "image" folder, the key in the dictionary will be the string "IFN_background_1_png" and the value will be the path and filename of the actual resource file, such as "image / background_1.png." If you need a multi-lingual version, you should create subfolders under the "image" folder such as "eng" or "kor" to save the file, then the "plist" file would contain each resource in its own separated dictionary data structure.

When the making of the "plist" file is completed, "ResourceMaker" creates the "Defines.h" header file. In this file, "#define" macros defines the strings, which used as key values in the dictionary data structure, as constant values. Defining these key values as constant values can reduce typing errors when coding a program and prevent access to the wrong resource at compile time. In game programming, source code directly accesses many of resource files as literals that represent file paths and names. Even if there is a typo in the literal that indicates the path and name of a resource file in source code, there would occur no error at compile time, but it could be a big problem because there is a high probability of error when accessing the resource with wrong path and file name at runtime.

Resources composed of data, such as strings, monster information, and weapon information, are usually created in a spreadsheet program such as Microsoft's Excel or Apple's Numbers. Spreadsheet programs are easy to use by game designers who have no knowledge of computer programming, so their intentions can be applied freely without the help of programmers. Data written in a spreadsheet can be saved as a comma-separated values (csv) file, which the "ResourceMaker" program interprets and adds dictionary data to the "plist" file. In addition, the "ResourceMaker" automatically creates "#define" macro statements in the

"Defines.h" header file. The monster information written in the spreadsheet includes life, armor, attack damage, attack range, speed, weight, and reward, and the weapon information is about consumed energy, attack power, attack range, and cool time (the time from the last firing to the next firing).

C. *Level Engine*

For multi-level games like Biosis, designing each level takes a lot of planning work. The level data consists of game events such as the appearance of monsters (including name, position, etc.), dialogues, various power ups (such as health packs, weapons, and items), and changes of background music. Each game event occurs only when certain conditions are met. Examples of conditions include "after a certain amount of time has elapsed" or "when all monsters have disappeared".

In Biosis, all game events are also written in a spreadsheet. The written events data is integrated into the "plist" file by "ResourceMaker." Then, the integrated game events data is processed in turn in the main loop of the level engine.

By using the level engine that is implemented as described above, there is no need to change the source code even if a new level is added or an existing level design is changed. This means that game designers can freely design various levels without the help of a programmers.

V. CONCLUSION

In this paper, we presented a development example of a mobile game called "Biosis" and proposed guidelines for developing one-source multi-platform mobile games using the Cocos2d-x engine. The developed game worked perfectly on both the iOS and Android platforms, without any porting process. In addition, we introduced methods and tools to facilitate resource management and share them among multiple platforms, and added an introduction to the level engine to allow game designers to freely design game levels without the help of programmers.

The proposed guidelines are expected to help reduce time and costs for small and indie game developers in the current mobile game ecosystem which has very short life-cycle.

ACKNOWLEDGMENT

This Work was supported by Dong-eui University Foundation Grant (2012).

REFERENCES

[1] K. Jeong, "Future Direction of Mobile Game Market according to the Advance of Mobile Device Capability," Journal of Digital Contents Society, Vol 11, No. 4, pp. 495-501, Dec. 2010.
[2] Ya-Ri Lee, Jung-Sook Kim, "Cross Platform Game Development Environment for PMP," Journal of Digital Contents Society, Vol 8, No. 3, pp. 377-383, Sept. 2007.
[3] Unity - Game Engine, "http://unity3d.com"
[4] ShiVa3D - Game engine with development tools, "http://www.stonetrip.com"
[5] Develop Cross Platform Mobile Apps and Games | Corona Labs, "http://www.coronalabs.com"
[6] Mobile Application Development, iPhone & Android App Development - Marmalade, "http://www.madewithmarmalade.com"
[7] Cocos2d-x | Cross Platform Open Source 2D Game Engine, "http://www.cocos2d-x.org"
[8] CuvicVR 3D Engine, "http://www.cubicvr.org"
[9] IwGame Engine | DrMop, "http://www.drmop.com/index.php/iwgame-engine"
[10] jumpcore, "https://bitbucket.org/runhello/jumpcore/wiki/Home"
[11] Moai | The mobile platform for pro game developers, "http://getmoai.com"

Jinseok Seo received the M.S. and Ph.D. degrees in Computer Science and Engineering from Postech, Korea, in 2000 and 2005, respectively. Since 2005, he joined the division of digital contents technology, Dong-eui University, Busan, Korea. His main research interests are artificial intelligence for computer games, game engines, virtual reality, and augmented reality.

Hun Choi received the M.S. and the Ph.D. degrees in electronics from Chungbuk National University, Korea, in 2001 and 2006, respectively. From 2006 to 2008, he was Post Doc. at KRISS, KOREA. Since 2008, He joined the division of Electronic engineering, Dong-eui University, Busan, Korea. His research interests include adaptive signal processing, measurement signal processing and communication system.