# Andorid platform and mobile game development

Gunay Karlı, Ena Kurtovic
[1]Faculty of Engineering and IT, Department of IT, Sarajevo, Bosnia and Herzegovina

**Abstract**— As technology improves, there is an ever-growing demand for better quality games and the Android, being the fastest growing platform in recent history is no exception. Mobile gaming is becoming an increasingly popular past-time and as such, it's a constantly evolving field. This research has brought to light the possibilities and hurdles of developing games on the Android, mostly centered around the issue of input and lack of tactile feedback, perhaps the most glaring problem new developers face. Furthermore, the huge variety in screen sizes and resolutions, their implications and problems they create were discussed. Several game engines that support Android have been explored and compared; and finally through a case-study game using the Unity Engine, a working prototype was developed, demonstrating solutions to some of the common pitfalls in the development pipeline.

**Index Terms**— Mobile, gaming, platform, enigins, Android,Unity Engine, computer graphics,

## 1 INTRODUCTION

Gaming is perhaps the fastest booming industry in the world today. In just under 20 years, we have moved from simple 8-bit sprites to nigh-photorealistic images rendered in real time on a home PC system, as briefly discussed in “Sprite Graphics” [3] . While this transition can easily be attributed to the increase in processing power, that is not the only culprit. Aside from technical improvements made over the year, developers have also improved techniques to allow for a much more robust and streamlined development approach [1] [2] [3] .

While that sounds great on paper, the reality is that improvements in technology bring with them an increase in overall complexity, and the ever-increasing user demands, especially in graphical quality, make technique improvements a race against hardware improvements. All that is still only centered around the PC, yet the single biggest factor that makes game development such a technically challenging matter is the sheer number of platforms - since its inception, gaming has been split into PC and console gaming, the two requiring radically different approaches [2], as each of the handful of platforms had its own sets of limitations, requirements and development cycles [4].

Fast forward 20 years into the future and "a handful of platforms" became "a dozen platforms", with Windows, Mac and Linux systems being present on the PC side, while the console market is split into the big three - Nintendo, Microsoft and Sony. However, we have also seen the rise of another "gaming category", namely handheld games. It pioneered with Nintendo's GameBoy, and has been present in one way or another over the years.Here is where things get interesting - insofar, the PC has been the only general-purpose platform, meaning that it was the only one that could do things other than gaming. This allowed the rest of the crew to get away with lower hardware requirements, making them more affordable and easier to work with (with a few exceptions).Thus we reached smartphones - the single biggest advancement in mobile computing of the decade. The last 3 years saw an explosion of smartphone popularity and the development advances that come with it. Today, there are two leading platforms on the market - Apple's iOS and Google's Android [4] [5].

Lastly, the developer can choose the level of .NET support, as well as the amount of byte stripping. Both can reduce the file size of the final .apk, but can also create compatibility issues. With the recent application size increase on the market, these are pretty much obsolete and should only be considered if the final .apk is reaching just above 50MB, and one wants to avoid the additional expansion files [17].

### 1.1. GAME DEVELOPMENT

Game development is very different from traditional application development due to the fact that games require content while traditional applications are mostly tools. Developing a video editor such as Adobe After Effects, for example, is done pretty "close to the chest", with all departments working on the various modules that deal with editing video. Game development in comparison is a more layer-based process. Care has to be placed on every part on the development pipeline separately, including but not limited to, engine development, gameplay development, interface design, sound and music design, voice recording, audio programming, user input, modeling, texturing, animation, story and dialogue writing, etc.

### 1.2. ANDROID MARKET

The Android Market (Renamed to Google Play Store [18] at the time of writing) has joined platforms such as the Apple Market [18] and Steam [16] [18] with its unprecedented ability to cut out publishing companies from a game's development lifecycle. An extremely cheap publishing license allows developers to get their work out very quickly and efficiently, push updates at their own pace and have complete statistical insight into their product. This allows individuals or small companies to easily get their work out to the masses.

It goes without saying that the system is not flawless - saturation being the biggest problem of the bunch. Technically speaking, the internet as a whole could serve a similar purpose as the market - upload your game and it is there for the public to play. However, without publicity how would anyone find out

about it? This issue is creeping into the Android market at an ever increasing pace. More developers getting their games out is great for innovation and diversity, but awful for the supply / demand ratio. Numerous games get uploaded to the Android market every month, and most of them get very little attention [17] [18] . One could look at this as more of a filtering mechanism, where the quality games will rise above the mediocre ones, and that is a valid interpretation, but arguably the over-saturation of quality material is only a matter of time as well.

The second issue with the market system is unique to smartphones - file size limitations. Up until recently, the maximum application size was 50MB [18]. Most 3D high-end games however required much more than that, and as such developers were forced to host game files on a separate server and have their games download this extra content from inside the application.

However, in March 2012, Google expanded the maximum market file size to 4GB [18]. The .apk file is still limited to 50MB to ensure secure on-device storage, but developers are allowed up to 2 expansion files in the format of their choosing, each up to 2GB big. While this is a huge leap for developers, the fact that a limit exists in the first place might pose an issue to a small percentage of games, namely Massive Multiplayer Online (MMO) games, where expansions and additional content are a constant occurrence. Even though 4GB sounds ample, many of today's high-end games are up to 3GBs in size, leaving prospective MMO games only 1GB left for expansion. Games such as these would be required to still host their game data as an in-application download [18] .

## 2. METARIAL AND METHODS

Before dwelling deeper into the design of the case study game, it is important to mention Android activities, OpenGL contexts, what they are and how Unity handles them. In simple terms, an Android activity can be looked at as a process thread. The email application, when opened, creates an activity. Tapping the "compose" button brings up the compose dialog on top of the inbox. That dialog is a new activity, and the inbox activity, now obscured, is stopped. If an activity is now called on top of the compose mail dialog, but does not obscure it completely (like a small confirmation window), the dialog below is paused. If Android requires memory, it clears stopped activities, re-activating them once they become visible [6]. In the case of extreme memory shortage, paused activities can be cleared from the memory. This is very similar to how contemporary operating systems handle process swapping and scheduling [6] [7] [8] [9].

### 2.2. Platforms and Devices

Android is a mobile operating system designed and developed by Google, based on Java [10]. However, it is also open-source, meaning that after licensing Android, a mobile phone manufacturer will implement features of his own. This leads to numerous problems, most notable being compatibility issues that can arise. However, a less prominent issue is the fact that it sometimes takes months for those same manufacturers to push updates onto their Android builds. Bug fixes and Android upgrades are infrequent and slow [8] [13] [14].This is further complicated by the mentality of people who think of their Android devices as traditional phones (for the sake of simplicity, tablets are not mentioned), and never bother to update even if an update is present.

On the side of hardware variety, Android devices are shipped with a specific configuration and cannot be upgraded, meaning that developers need to consider a wider spectrum of configurations when developing their game in order to make sure it runs smoothly on as many devices as possible. As previously discussed, aspect ratios are another important factor that needs to be taken into account, adding yet another layer of complexity. While Android offers automatic methods to stretch activities to match screen resolutions, aspect ratios in games have to be constrained most of the time

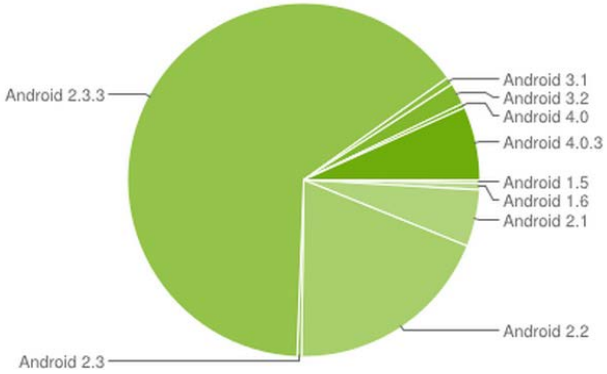


Fig. 2.2 A pie-chart showing a breakdown of the various Android versions as of June 2012 [12].

### 2.3. SYSTEM DESIGN

In theory, this can be used to create the different sections of the game - menus, levels, pause screens etc. but in practice, and in Unity, the entire game is one single activity. The reason becomes apparent when the OpenGL context is considered. In short, the OpenGL context is equivalent to the Java Virtual Machine, in terms that it contains all the necessary data for the rendering of the current frame buffer on the screen. Swapping out an activity destroys the OpenGL context, and it would later on have to be re-constructed from scratch, a very expensive operation.

Using a single context also means that Android would not cache any activities. The reduction in asset loading and saving also helps performance a great deal. In general, the concept of activities does not map well to games.As far as version requirements are concerned, modern games should choose a minimum Android version of 2.3.3. (Gingerbread), ARMv7 devices (Meaning the game will only show up in the market to devices that support Unity Android) and OpenGL ES 2.0, which unlike its 1.x counterpart supports modern shaders.

## 3. RESULTS AND DISCUSSION

The case study game will be a puzzle game, centered around reflecting a laser beam across the field by using rotating

and moving mirrors as well as portals. Once the beam reaches its target destination, a victory text is shown in the middle of the screen and the game ends. The main challenge of the game is figuring out the correct configuration of the elements required to lead the laser to its target destination [14].

### 3.1. CONCEPT

This is further enhanced by the fact that the player does not have a complete overview of the "grid" due to a limited viewport, thus he has to make mental notes of mirrors or portals he can not see. Some of the features that this case study will cover are touch screen camera control, accurately tapping objects, isometric camera boundaries, transparent texture shaders and positioning of UI elements on the variable sized screens.

The game will consist of the following elements: laser beam, source of the laser beam, mirrors that rotate on touch , mirrors

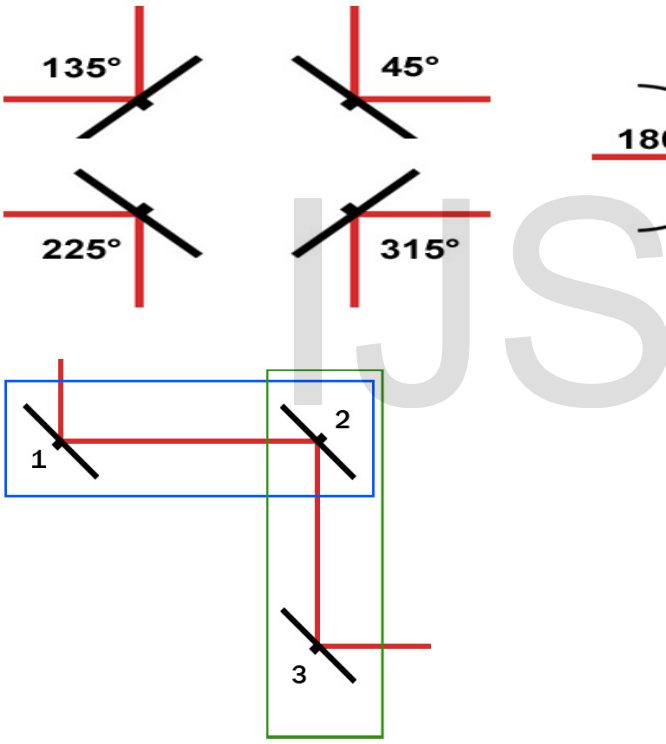


Fig. 3.1 The possible angular positions of mirrors (left) and portals (right) along with the possible laser directions coming in and out of them (red line).

Reflecting the laser around will be done via a trick - all elements, mirrors and portals will be able to emit the laser under the condition that an incoming beam is hitting them from the correct direction, i.e. the beam will no reflected, no reflection calculation will be necessary. Every element that reflects is actually an emitter that emits under certain conditions (being hit by a beam from a certain direction). Both the direction of the incoming beam and the angle of the mirror / portal being hit have to be checked to determine if the incoming laser is being reflected or not. Unity supports line renderers (used to draw the laser) that have several joints facing different directions, and those joints can be dynamically created and removed, but that approach would require sequentially re-calculating the line from start to finish each time a mirror or portal is adjusted, which can become quite time consuming if there are more than a few dozen elements in a given stage. The separate-lines approach allows for easier debugging and better portability.Special care has to be made in cases when there are two mirrors facing in opposite directions, as they can have two different configurations in which they reflect the beam (vertical and horizontal, see Figure 3.1.). This is why it is necessary to check the coordinate difference between the "source" and "target" mirror as well as their rotation.

Fig. 3.2. Mirrors 1-2 are positioned horizontally (blue), while mirrors 2-3 are positioned vertically (green). Despite mirrors 1 and 3 being identical, the resulting reflection is different.

### 3.2. SYSTEM IMPLEMENTATION

Before diving deeper into the Unity Editor and specific code implementation of Mirror Game, an understanding of a few terms is required.Prefabs, stemming from the word "prefabricated" are complete game objects, be it static or dynamic, that are present as available resources in the project. Objects present in levels are instances of prefabs. Changing a prefab will mirror that change onto every instance of the prefab. A scene does not necessarily depote a new level, a scene can also be the main menu, or the scoreboard. Parents and Children are groups of objects in which changes to the transformation of the parent also affects the transformation of all its children. This is not true the other way around. Last but not least, transformation is the current position, scale and rotation of an object [10] [11].

#### 3.2.1.CODING

The most importantt part of developing Mirror Game was the laser emitter. This would be the actual game object that would produce the laser beam. It would be attached to the laser source object (which is nothing but a container model) as well as each mirror and portal, Figure 4.3.

Fig. 3.2.2. The various objects with laser emitters attached to them.

The three crucial things that make up the laser emitter

are the LaserScope shader, the LaserScope script and the PerFrameRaycast script.The LaserScope.shader script takes two arguments - the main texture, in this case a solid color, and the noise texture, used to create the diffuse effect around the laser.

```
private var hitInfo : RaycastHit;
private var tr : Transform;
private var lMask : int;

function Awake () {
    tr = transform;
}

function Update () {
    // Cast a ray to find out the end point of the laser
    hitInfo = RaycastHit ();
    lMask = ~(1<<8);
    Physics.Raycast(tr.position, tr.forward, hitInfo, 50, lMask);
}

function GetHitInfo () : RaycastHit {
    return hitInfo;
}
```

The concept is as follows - "shoot" a vector, or ray (Physics.Raycast) into a certain direction (transform.forward). If the ray hits something, store the resulting RaycastHit information into a variable (hitInfo). The ray travels a maximum distance of 50 and will collide with all objects except those in layer 8.The last bit requires some explanation on how layer masks work. The

```
//If the source is at 0 degrees (y)
if(Mathf.Round(this.transform.parent.localEulerAngles.y)==0) {
    if (Mathf.Round(hitInfo.collider.transform.localEulerAngles.y)==90) {
        if (this.transform.parent.position.x < hitInfo.collider.transform.position.x)
            hitInfo.collider.transform.FindChild("LS_1").GetComponent(LaserScope).isOn=true;
    }

    else if (Mathf.Round(hitInfo.collider.transform.localEulerAngles.y)==180){
        if (this.transform.parent.position.x < hitInfo.collider.transform.position.x)
            hitInfo.collider.transform.FindChild("LS_0").GetComponent(LaserScope).isOn=true;
        else
            hitInfo.collider.transform.FindChild("LS_1").GetComponent(LaserScope).isOn=true;
    }

    else if (Mathf.Round(hitInfo.collider.transform.localEulerAngles.y)==270){
        if (this.transform.parent.position.z > hitInfo.collider.transform.position.z)
            hitInfo.collider.transform.FindChild("LS_0").GetComponent(LaserScope).isOn=true;
    }
}
```

optional final argument of the Raycast() method takes a layer mask, which is nothing but a bitmap telling the method which layers to include or not include in its collision tests. A layer mask that simply says 1 would mean "collide with objects in the first layer only", 10 would mean "objects in second layer only", 1000 is "fourth layer" and so forth.

In this case, the layer mask is created via the bitwise << operator. The given command, 1<<8 reads "1 shifted to the left 8 places", resulting in 10000000. Finally, the inverse operator (~) is applied to the result, bringing the final bitmap to 01111111. This reads as "all layers except for layer 8". This layer contains the Path objects, which are used later on to set the boundaries for movable mirrors.The final script, LaserScope, controls all of the laser logic. First thing to note is that in the Unity editor, scripts can be easily attached to objects by dragging and dropping. Once attached, those scripts become components. Those components can be referenced in other scripts like this:

First off, once the laser is initialized, an animation routine has to be determined in order to give the laser a smooth diffuse animation.

This does nothing more but constantly randomize the animation direction and width of the laser by small steps each frame. The yield command is a special kind of return that ensures that the next time the method is called (once per frame), it will continue from the yield statement. This script, like most other object scripts in the project, has an Update() method. This is a special method that is called once every frame. So if the game runs at 60 frames per second (fps), Update() is called 60 times every second. This is the "main function" of every object. In this particular Update() method, the first thing performed is a condition check:

```
function ChoseNewAnimationTargetCoroutine () {
    while (true) {
        aniDir = aniDir * 0.9 + Random.Range (0.5, 1.5) * 0.1;
        yield;
        minWidth = minWidth * 0.8 + Random.Range (0.1, 1.0) * 0.2;
        yield WaitForSeconds (1.0 + Random.value * 2.0 - 1.0);
    }
}
```

The isOn flag is a simple boolean variable which is triggered based on whether or not the laser emitter is supposed to emit at the current moment. The isTurning boolean is used to check whether or not a mirror or portal is in the process of rotating / moving, during which time all the lasers turn off and user input is temporarily suspended.The next few lines call the renderer object, a component of every drawable object in Unity. The first line enables it, i.e. the laser emitter in question becomes visible. The next 5 lines simply control the various random animation components. The most notable one of those is the one calling the lRenderer variable, which is short for Line Renderer. As discussed in 3.a, line renderers are used to render lines, or in this case lasers.

The next task is to check if the freshly enabled laser is hitting anything.

As seen above, the PerFrameRaycast component has been stored in the raycast variable, which can now be used to

```
// Cast a ray to find out the end point of the laser
var hitInfo : RaycastHit = raycast.GetHitInfo ();
if (hitInfo.transform) {
    lRenderer.SetPosition (1, (hitInfo.distance * Vector3.forward));
    renderer.material.mainTextureScale.x = 0.1 * (hitInfo.distance);
    renderer.material.SetTextureScale ("_NoiseTex", Vector2 (0.1 * hitInfo.distance
                                                * noiseSize, noiseSize));
```

access it. Since PerFrameRaycast returns a RaycastHit object, that is stored in the hitInfo variable. If the ray did not hit anything, this will result in a NULL object returning, so it can easily be checked whether or not there was a hit.The next two lines simply handle the scaling of the two textures used in the laser material, making sure they are properly tiled, not stretched, across the length of the laser.In case the laser did not hit anything, the only thing that needs to be done is give the SetPosition()

method a maxDistance variable instead of the collision distance.Now that the emitter knows that it hit something though, it

```
if (hitInfo.collider.tag=="Mirror") {
```

has to figure out what the object it hits is.

First off, it checks for mirrors (Both rotating and moving ones). Three things need to be examined - the Y-axis rotation (referred to as simply rotation henceforth) of the source, the rotation of the target (mirror hit), and in case they are at opposing angles, their position relative to one another to determine whether they are in a horizontal or vertical setup. It must be noted that the models of the mirrors are rotated 45° clockwise, i.e. 45° is their 0° as far Unity is concerned. This works great as it simplifies calculations and allows for much more natural numbers.

Firstly, it is checked whether the source is at a certain rotation (in this case 0°). Note that the emitter checks the rotation of is parent, as it is itself attached to a mirror or portal (this relation is established by simply drag and dropping one object onto another in the Unity editor). For the sake of safety, angles are rounded before checking. Assuming that the source mirror is indeed at 0° (i.e. 45°), there are only two possibilities - the laser is being emitted either to the north or the east.

### 3.2.3. BUGS AND FIXES

Rotation of models in Maya resulted in incorrect rotation after the models were imported into Unity. Essentially, the model must remain at (0,0,0) position with a (0,0,0) rotation. Any modification should be done at a component level.Transparent textures did not function correctly - the laser appeared as a placeholder purple material, and several transparent png's had their transparent areas filled with white. This was due to the technical requirement of OpenGL ES 2.0 as well as a 32-bit Display Buffer.

Path marker detection was not working correctly on several occasions, each time sporting a different reason. Firstly, the culprit was the non-rounding of the post-move coordinates. Just a bit to both side and the ray detecting the path marker would miss it altogether. Next time it did not work due to the ray hitting the path marker the mirror is sitting on at the current moment. This happened haphazardly and without any coherent pattern, so debugging took longer than the other issues.Perhaps the most annoying bug of all was not even directly related to programming the game, but rather to the Android SDK. Unity is able to compile and run the game directly on an Android device that is connected to the PC via USB. However, installing the actual APK file on the fly kept failing. It is still unclear what caused the bug, but reinstalling the SDK eventually fixed it.

### 3.2.3. MODELING

For the purpose of this paper, only rudimentary models have been used, coupled with a few placeholders. However there are a few notable things that affected further coding that require mention. Most notably, it is imperative that finished models have all of their transformation values set to 0 for translation and rotation and 1 for scale. If this is not the case, Unity will translate, rotate and scale in such a way that the values match those "origin" values.

Fig. 3.2.3. The model for the movable mirror.

```
if (this.isOn == true && this.isTurning == false) {
        renderer.enabled = true;
        renderer.material.mainTextureOffset.x += Time.deltaTime * aniDir * scrollSpeed;
        renderer.material.SetTextureOffset ("_NoiseTex",
                  Vector2 (-Time.time * aniDir * scrollSpeed, 0.0));

        var aniFactor : float = Mathf.PingPong (Time.time * pulseSpeed, 1.0);
        aniFactor = Mathf.Max (minWidth, aniFactor) * maxWidth;
        lRenderer.SetWidth (aniFactor, aniFactor);
```

Fig. 3.2.3. The model for the movable mirror.

## CONCLUSIONS

Many of the methods looked vastly different in their original inception, and numerous bugs popped up all the time. Just like with most programming, many of those issues were caused by "doing it wrong", but the occasional one had its roots set in deeper.

Despite the platform being barely out of its infancy, it is remarkable how streamlined the development for it is. Simple concepts and a powerful SDK really allow for a great deal of creative freedom and flexibility. Even for non-game development, the designer can control virtually every aspect of the application, how it interacts with the system itself as well as other applications. None of the engines mentioned in this thesis would have managed to do what they did if the Android operating system itself was not extremely well-written.

The version and (lack of) update issues are slowly being pushed aside, as the newer versions of Android support nearly all modern technical game features. Thus it is just a matter of time until every Android device will be able to run 3D games, the only restriction being the hardware specifications, much like with PCs.

Also for the end I would like to mention that during this project and game development process Unreal Unity Engine 4 was lounched and it cuts down on development time and ensures faster iteration on creative ideas. Howeverm, there is no doubt that application developing platforms (and technology in general) are is evolving field of current century.

Finally, in a market dominated by gigantic, faceless publishers recycling the same ideas and never straying away from the cookie-cutter status-quo, any venture that allows creative individuals to show the world what can truly be done with the medium once conventional notions are tossed out the window is more than welcome. And this is such a venture.

## REFERENCES

[1] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, Deborah, Estrin, Diversity in Smartphone Usage, First ed. , USA: Microsoft Research, 2008.

[2] Allan Hammershøj, Antonio Sapuppo and Reza Tadayoni , An analysis of Mobile Operating Systems and Software development platforms , First ed. , Copenhagen, Denmark : CMI international conference on social networking and communities , 2009.

[3] Paul Michael Kilgo, Android OS: A robust, free, open-source operating system for mobile devices, Alabama: 2008.

[4] Stephen A. Edwards, Sprite Graphics, First ed. , Columbia: Columbia University, 2010.

[5] Hartmut Schirmacher, Stefan Brabec, A Multi-Site, Multi-Platform System for Software Development, Firsted. , Hamburg: Max-Planck-Institut fur¨ Informatik, 2000.

[6] Martin Mittring, The Technology Behind the 3D Graphics and Games Course "Unreal Engine 4 Elemental demo", Epic Games, Inc., 2012.

[7] Anton Kaplanyan, Light Propagation Volumes in CryEngine 3, First ed. , SIGGRAPH, 2009.

[8] Martin Mittring, Bryan Dudash, The Technology Behind the DirectX 11 Unreal Engine "Samaritan" Demo, Unreal Technology, 2011.

[9] DONALD MUSTARD, INFINITY BLADE H ! INFINITY BLADE H OW WE MADE WE MADE A HI T,, W HAT W E LEARNED E LEARNED LEARNED LEARNED,, A ND WH Y YOU CAN DO IT TOO, First ed. , Chair, 2012.

[10] Martin Mittring, Advances in Real-Time Rendering in 3D Graphics and Games Course , Epic Games, 2012.

[11] Niklas Smedberg, Daniel Wright, Rendering Techniques in Gears of War 2, First ed. , Epic Games, Inc, 2010.

[12] Will Goldstone, Unity Game Development Essentials, Packt Publishing, 2009.

[13] Renaldas Zioma, iOS and Android - Cross-Platform Challenges and Solutions, First ed. , Unity Technologies, 2012.

[14] A.Mallikarjuna, S.Madhuri , Unveiling of Android Platform , First ed. , India: International Journal of Advanced Research, 2013.

[15] Masoud Nosrati, Ronak Karimi, Hojat Hasanvand, Mobile Computing: Principles, Devices and Operating Systems, First ed. , Iran: WAP journal, 2012.

[16] Victor Matos, Android App Development, Cleveland: Cleveland State University , 2012.

[17] Marko Gargenta, Learning Android, United States of America: O'Reilly Media, 2011.

[18] App Store Marketing and Advertising Guidelines for Developers, First ed. , USA: Apple, 2012.

[19] Martin Wallace, Steam, First ed. , Mayfair Gamer, Inc., 2009.