# The limitations of a mobile platform (Android)

In partial fulfillment of GPG230

Following American psychological association guidelines

Noaman Khalil

SAE Institute Dubai

## Abstract

This papers goal is to look over the limitations and challenges faced when building a game for multiple platforms , in the case of this scenario it will be the limitations of android vs the freedom of a PC . The paper would also look over the optimizations and features stripped in order for the successful compilation of the project

## Introduction

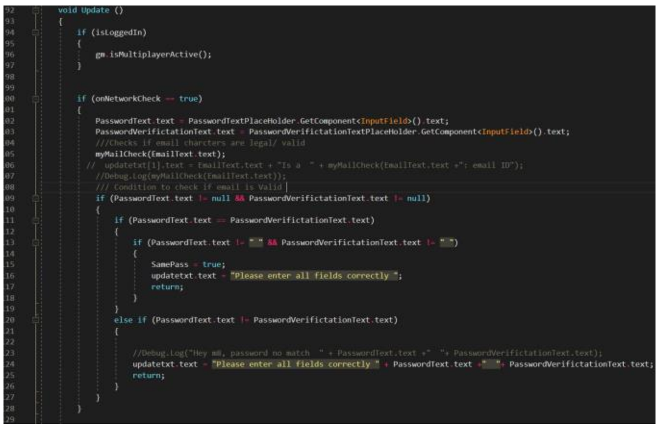
Developing a game with the platform of PC being in mind is much simpler, than say that it is of android, the platforms have many differences in terms of technology with one being capable of running high end graphics while the other running sufficient graphics while being constraint in terms of power and thus the increasing need to optimizations in games being made for multiple platforms.

## Implementation

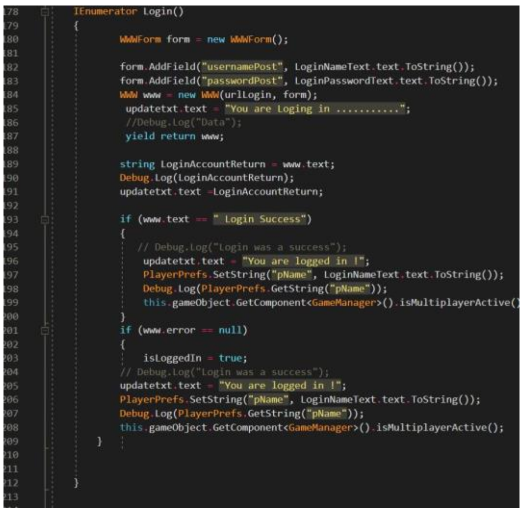
The first step in the implementation of the optimization for android began early on in the project wherein I took knowledge attained by my experience in making an android build in GPG220 and applying to this project wherein I made sure there was no lag between multiple scenes by eliminating the need for multiple scenes, the project is built in the same scene this helps avoid long wait times required to switch scenes on an android device however the trade-off is a single larger scene which takes longer to load initially & is more complex to maintain than multiple smaller screens.

Another optimization is in regards to the mono-behaviour call backs built into unity, Start & Awake don’t consume too much performance as they are only called once when the objects is initialized /created however the Update loop/function (Applies to late update and fixed update too) can be seen as an opportunity as this is called every frame, to limit the operations done by update there can be a small increase in performance, a good way method employed in the confines of this project. (IronEqual, 2017) One small way to avoid update from running constantly is to have conditions wrapped around the code, in the code snipped below it shows a bool that is checked before the update functions code is executed by doing this around 20 lines of code are not executed, this may seem like a small enhancement however in the case of multiple scripts this should make a significant difference. (IronEqual, 2017)

A simple solution which is to use enumerators can be implemented for things that will take a long time such as long  calculations or data which can be processed over multiple frames , a implementation of this is also in the example below.



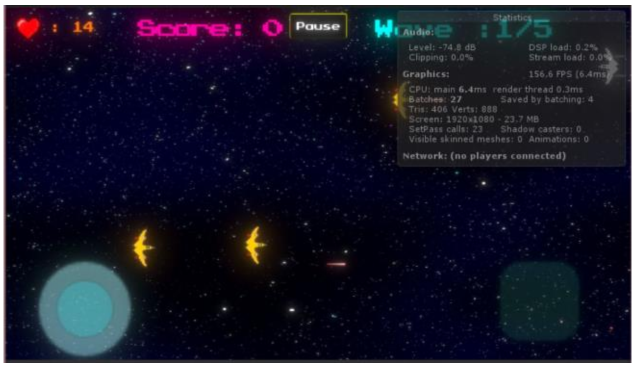
```
void Update ()
{
    if (isLoggedIn)
    {
        gm.isMultiplayerActive();
    }


    if (onNetworkCheck == true)
    {
        PasswordText.text = PasswordTextPlaceHolder.GetComponent<InputField>().text;
        PasswordVerifictationText.text = PasswordVerifictationTextPlaceHolder.GetComponent<InputField>().text;
        ///Checks if email charcters are legal/ valid
        myMailCheck(EmailText.text);
      //  updatetxt[1].text = EmailText.text + "Is a  " + myMailCheck(EmailText.text +": email ID");
        //Debug.Log(myMailCheck(EmailText.text));
        /// Condition to check if email is Valid
        if (PasswordText.text != null && PasswordVerifictationText.text != null)
        {
            if (PasswordText.text == PasswordVerifictationText.text)
            {
                if (PasswordText.text != " " && PasswordVerifictationText.text != " ")
                {
                    SamePass = true;
                    updatetxt.text = "Please enter all fields correctly ";
                    return;
                }
            }
            else if (PasswordText.text != PasswordVerifictationText.text)
            {

                //Debug.Log("Hey m8, password no match  " + PasswordText.text +"  "+ PasswordVerifictationText.text);
                updatetxt.text = "Please enter all fields correctly " + PasswordText.text +"  "+ PasswordVerifictationText.text;
                return;
            }
        }
    }
```



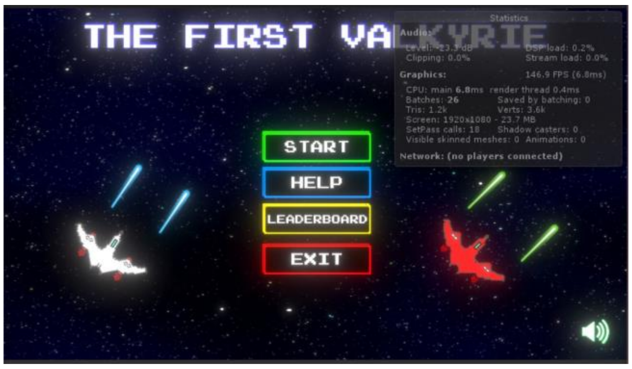
```
IEnumerator Login()
{
        WWWForm form = new WWWForm();

        form.AddField("usernamePost", LoginNameText.text.ToString());
        form.AddField("passwordPost", LoginPasswordText.text.ToString());
        WWW www = new WWW(urlLogin, form);
         updatetxt.text = "You are Loging in ...........";
         //Debug.Log("Data");
         yield return www;

        string LoginAccountReturn = www.text;
        Debug.Log(LoginAccountReturn);
        updatetxt.text =LoginAccountReturn;

        if (www.text == " Login Success")
        {
          // Debug.Log("Login was a success");
            updatetxt.text = "You are logged in !";
            PlayerPrefs.SetString("pName", LoginNameText.text.ToString());
            Debug.Log(PlayerPrefs.GetString("pName"));
            this.gameObject.GetComponent<GameManager>().isMultiplayerActive()
        }
        if (www.error == null)
        {
            isLoggedIn = true;
        // Debug.Log("Login was a success");
        updatetxt.text = "You are logged in !";
        PlayerPrefs.SetString("pName", LoginNameText.text.ToString());
        Debug.Log(PlayerPrefs.GetString("pName"));
        this.gameObject.GetComponent<GameManager>().isMultiplayerActive();
    }


}
```

Another optimization is in regards to the scripts themselves rather than a single mono-behaviour call-back where in the script can be enabled when needed, this was the case with some components such as the photon Network manager, now this is only to be used in the case where the player chooses the multiplayer aspect of the game however, it needs to be disabled when the player is in-between menus / not in an active game scene.

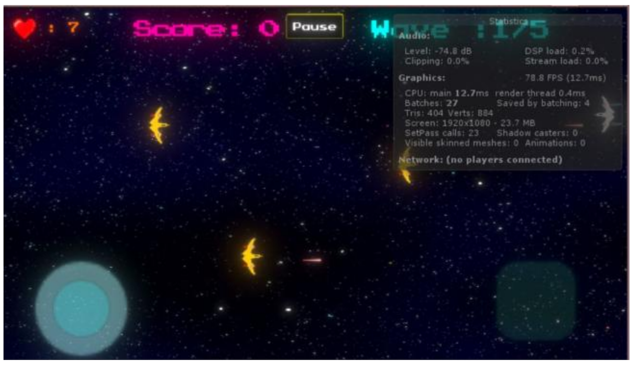
The optimizations so far have yielded excellent framerates and performance in the builds and in engine which can be seen below however real android performance will be greatly different.



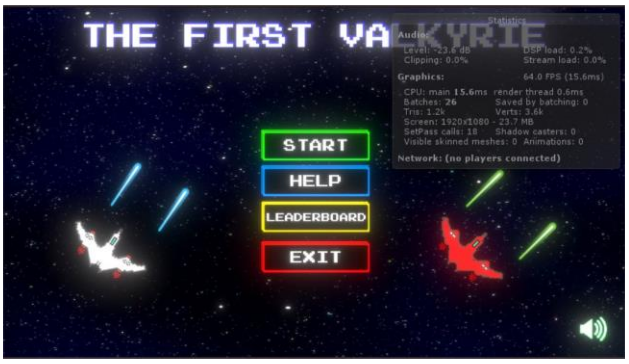
```
Statistics
Audio:
  Level: -74.8 dB              DSP load: 0.2%
  Clipping: 0.0%               Stream load: 0.0%
Graphics:                      156.6 FPS (6.4ms)
  CPU: main 6.4ms  render thread 0.3ms
  Batches: 27           Saved by batching: 4
  Tris: 406  Verts: 888
  Screen: 1920x1080 - 23.7 MB
  SetPass calls: 23     Shadow casters: 0
  Visible skinned meshes: 0  Animations: 0
Network: (no players connected)
```



```
Statistics
Audio:
  Level: -23.3 dB              DSP load: 0.2%
  Clipping: 0.0%               Stream load: 0.0%
Graphics:                      146.9 FPS (6.8ms)
  CPU: main 6.8ms  render thread 0.4ms
  Batches: 26           Saved by batching: 0
  Tris: 1.2k            Verts: 3.6k
  Screen: 1920x1080 - 23.7 MB
  SetPass calls: 18     Shadow casters: 0
  Visible skinned meshes: 0  Animations: 0
Network: (no players connected)
```



```
Statistics
Audio:
  Level: -74.8 dB              DSP load: 0.2%
  Clipping: 0.0%               Stream load: 0.0%
Graphics:                      78.8 FPS (12.7ms)
  CPU: main 12.7ms  render thread 0.4ms
  Batches: 27           Saved by batching: 4
  Tris: 404  Verts: 884
  Screen: 1920x1080 - 23.7 MB
  SetPass calls: 23     Shadow casters: 0
  Visible skinned meshes: 0  Animations: 0
Network: (no players connected)
```

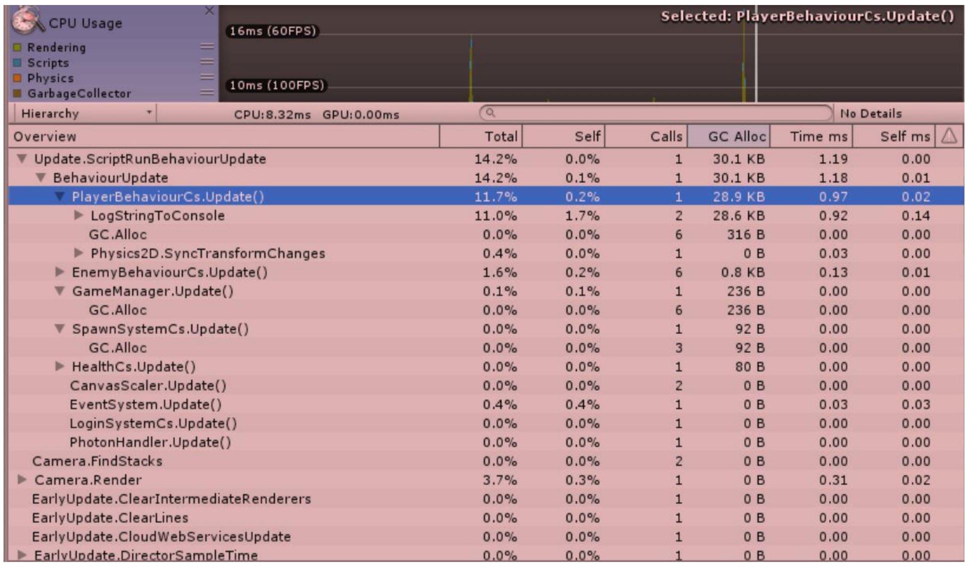


```
Statistics
Audio:
  Level: -23.6 dB              DSP load: 0.2%
  Clipping: 0.0%               Stream load: 0.0%
Graphics:                      64.0 FPS (15.6ms)
  CPU: main 15.6ms  render thread 0.6ms
  Batches: 26           Saved by batching: 0
  Tris: 1.2k            Verts: 3.6k
  Screen: 1920x1080 - 23.7 MB
  SetPass calls: 18     Shadow casters: 0
  Visible skinned meshes: 0  Animations: 0
Network: (no players connected)
```

The images above illustrate the performance on an android device (s7edge ) and as the performance indicates that the mobile platform only gets 50% of the performance that the PC test machine did , this is a clear indication of the power limitations of android however these numbers are pulled while the profiler is open thus the real performance by my estimates should be around 60% or more in the best case scenarios and about 40% of the pc benchmarks in the worst case scenario which would be a lower end android device.

Another optimization is in regards to how the 2d colliders work in unity, according to the "Practical Guide for Optimizing Unity on Mobiles" unity rebuilds the collider's every time an animation is played and thus I have taken to a dentists approach to it wherein I have avoided animations altogether, this paired with 2d colliders over mesh colliders for 2d objects reduced the number of vertices and triangles. (Simonov/Unity technologies, 2015)

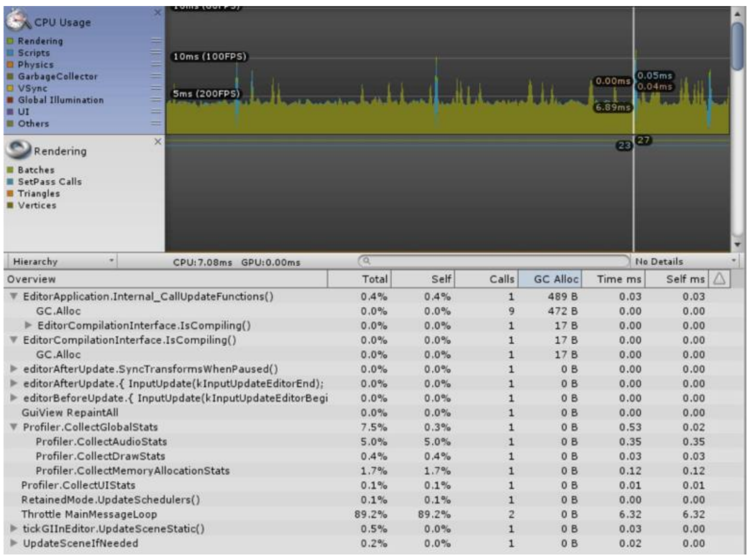
Next is the most crucial tool within the confines of unity, which is the unity profiler, this can help the user find memory leaks and plug them , this lays out the amount of memory a script might be utilizing per frame.



CPU:8.32ms GPU:0.00ms

| Overview | Total | Self | Calls | GC Alloc | Time ms | Self ms |
|---|---|---|---|---|---|---|
| ▼ Update.ScriptRunBehaviourUpdate | 14.2% | 0.0% | 1 | 30.1 KB | 1.19 | 0.00 |
| ▼ BehaviourUpdate | 14.2% | 0.1% | 1 | 30.1 KB | 1.18 | 0.01 |
| ▼ PlayerBehaviourCs.Update() | 11.7% | 0.2% | 1 | 28.9 KB | 0.97 | 0.02 |
| ▶ LogStringToConsole | 11.0% | 1.7% | 2 | 28.6 KB | 0.92 | 0.14 |
| GC.Alloc | 0.0% | 0.0% | 6 | 316 B | 0.00 | 0.00 |
| ▶ Physics2D.SyncTransformChanges | 0.4% | 0.0% | 1 | 0 B | 0.03 | 0.00 |
| ▶ EnemyBehaviourCs.Update() | 1.6% | 0.2% | 6 | 0.8 KB | 0.13 | 0.01 |
| ▼ GameManager.Update() | 0.1% | 0.1% | 1 | 236 B | 0.00 | 0.00 |
| GC.Alloc | 0.0% | 0.0% | 6 | 236 B | 0.00 | 0.00 |
| ▼ SpawnSystemCs.Update() | 0.0% | 0.0% | 1 | 92 B | 0.00 | 0.00 |
| GC.Alloc | 0.0% | 0.0% | 3 | 92 B | 0.00 | 0.00 |
| ▶ HealthCs.Update() | 0.0% | 0.0% | 1 | 80 B | 0.00 | 0.00 |
| CanvasScaler.Update() | 0.0% | 0.0% | 2 | 0 B | 0.00 | 0.00 |
| EventSystem.Update() | 0.4% | 0.4% | 1 | 0 B | 0.03 | 0.03 |
| LoginSystemCs.Update() | 0.0% | 0.0% | 1 | 0 B | 0.00 | 0.00 |
| PhotonHandler.Update() | 0.0% | 0.0% | 1 | 0 B | 0.00 | 0.00 |
| Camera.FindStacks | 0.0% | 0.0% | 2 | 0 B | 0.00 | 0.00 |
| ▶ Camera.Render | 3.7% | 0.3% | 1 | 0 B | 0.31 | 0.02 |
| EarlyUpdate.ClearIntermediateRenderers | 0.0% | 0.0% | 1 | 0 B | 0.00 | 0.00 |
| EarlyUpdate.ClearLines | 0.0% | 0.0% | 1 | 0 B | 0.00 | 0.00 |
| EarlyUpdate.CloudWebServicesUpdate | 0.0% | 0.0% | 1 | 0 B | 0.00 | 0.00 |
| ▶ EarlyUpdate.DirectorSampleTime | 0.0% | 0.0% | 1 | 0 B | 0.00 | 0.00 |

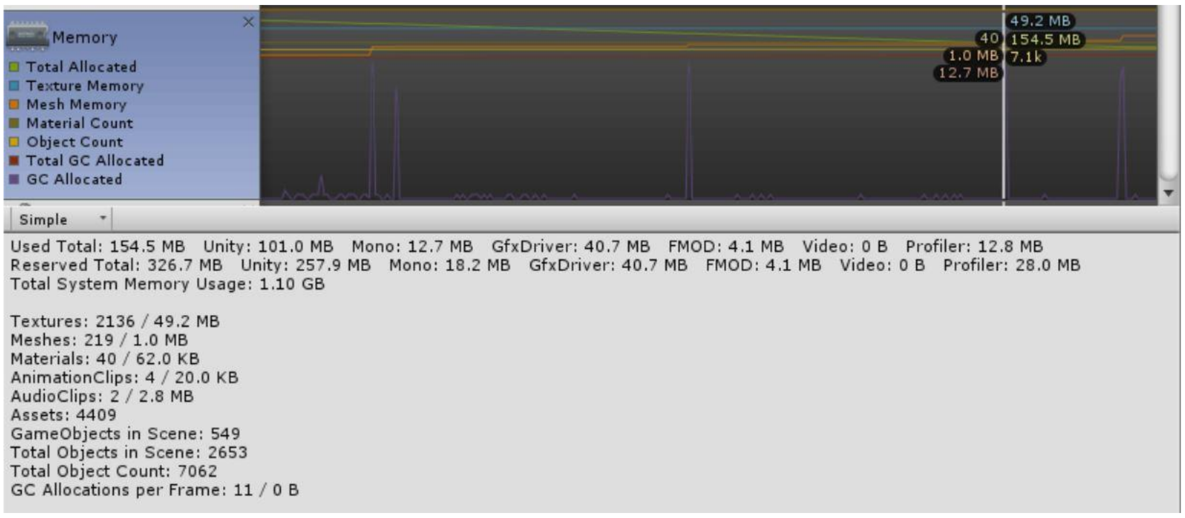
The GC allocation indicates the amount of Garbage being generated, right now the example illustrates garbage collection on the PC platform while its android profiling is shown in the image below wherein the garbage collection peak is similar to the pc counterpart.



CPU:7.08ms GPU:0.00ms

| Overview | Total | Self | Calls | GC Alloc | Time ms | Self ms |
|---|---|---|---|---|---|---|
| ▼ EditorApplication.Internal_CallUpdateFunctions() | 0.4% | 0.4% | 1 | 489 B | 0.03 | 0.03 |
| GC.Alloc | 0.0% | 0.0% | 9 | 472 B | 0.00 | 0.00 |
| ▶ EditorCompilationInterface.IsCompiling() | 0.0% | 0.0% | 1 | 17 B | 0.00 | 0.00 |
| ▼ EditorCompilationInterface.IsCompiling() | 0.0% | 0.0% | 1 | 17 B | 0.00 | 0.00 |
| GC.Alloc | 0.0% | 0.0% | 1 | 17 B | 0.00 | 0.00 |
| ▶ editorAfterUpdate.SyncTransformsWhenPaused() | 0.0% | 0.0% | 1 | 0 B | 0.00 | 0.00 |
| ▶ editorAfterUpdate.{ InputUpdate(kInputUpdateEditorEnd); | 0.0% | 0.0% | 1 | 0 B | 0.00 | 0.00 |
| ▶ editorBeforeUpdate.{ InputUpdate(kInputUpdateEditorBegi | 0.0% | 0.0% | 1 | 0 B | 0.00 | 0.00 |
| GuiView RepaintAll | 0.0% | 0.0% | 1 | 0 B | 0.00 | 0.00 |
| ▼ Profiler.CollectGlobalStats | 7.5% | 0.3% | 1 | 0 B | 0.53 | 0.02 |
| Profiler.CollectAudioStats | 5.0% | 5.0% | 1 | 0 B | 0.35 | 0.35 |
| Profiler.CollectDrawStats | 0.4% | 0.4% | 1 | 0 B | 0.03 | 0.03 |
| Profiler.CollectMemoryAllocationStats | 1.7% | 1.7% | 1 | 0 B | 0.12 | 0.12 |
| Profiler.CollectUIStats | 0.1% | 0.1% | 1 | 0 B | 0.01 | 0.01 |
| RetainedMode.UpdateSchedulers() | 0.1% | 0.1% | 1 | 0 B | 0.00 | 0.00 |
| Throttle MainMessageLoop | 89.2% | 89.2% | 2 | 0 B | 6.32 | 6.32 |
| ▶ tickGIInEditor.UpdateSceneStatic() | 0.5% | 0.0% | 1 | 0 B | 0.03 | 0.00 |
| ▶ UpdateSceneIfNeeded | 0.2% | 0.0% | 1 | 0 B | 0.02 | 0.00 |

The final results can be considered be considered acceptable, as the game does not create large chunks of garbage and utilizes under 1.2gb of memory which can be seen in the memory compiler below however the actual usage of memory will be much higher as the test case uses the unity remote app which uses more system memory (on android) to run the profiler and thus the real world memory usage should be well under the 1.2gb threshold. More about the profiler can be found on Gamasutra. (Tiercelin/Gamasutra, 2017)



```
Used Total: 154.5 MB   Unity: 101.0 MB   Mono: 12.7 MB   GfxDriver: 40.7 MB   FMOD: 4.1 MB   Video: 0 B   Profiler: 12.8 MB
Reserved Total: 326.7 MB   Unity: 257.9 MB   Mono: 18.2 MB   GfxDriver: 40.7 MB   FMOD: 4.1 MB   Video: 0 B   Profiler: 28.0 MB
Total System Memory Usage: 1.10 GB

Textures: 2136 / 49.2 MB
Meshes: 219 / 1.0 MB
Materials: 40 / 62.0 KB
AnimationClips: 4 / 20.0 KB
AudioClips: 2 / 2.8 MB
Assets: 4409
GameObjects in Scene: 549
Total Objects in Scene: 2653
Total Object Count: 7062
GC Allocations per Frame: 11 / 0 B
```

## Conclusion

Developing for multiple platforms requires significantly more time and effort by the developers in order to achieve acceptable framerates however the limitations of the content is only limited by the target market and or platform as the tools and optimizations needed for such platforms are numerous and yet to be explored .

## References

IronEqual. (2017, August 18). Unity: Android Optimization Guide ? IronEqual ? Medium. Retrieved from https://medium.com/ironequal/android-optimization-with-unity-3504b34f00b0

Simonov/Unity technologies, V. (2015, October 17). Practical Guide for Optimizing Unity on Mobiles. Retrieved from https://www.slideshare.net/valentinsimonov/practical-guide-for-optimizing-unity-on-mobiles

Tiercelin/Gamasutra, N. (2017, August 24). Unity: Android Optimization Guide. Retrieved from https://www.gamasutra.com/blogs/NielsTiercelin/20170824/304424/Unity_Android_Optimization_Guide.php