Performance Analysis On Idle Breeder Game

[1]Steven, [2]Agung Riyadi
[1,2]Multimedia and Network Engineering/Batam State Polytechnic
[1,2] Batam Centre, Jl. Ahmad Yani, Tlk. Tering, Kec. Batam Kota, Kota Batam, Kepulauan Riau 29461, Indonesia
[1]stevenwuu18@gmail.com, [2]agung@polibatam.ac.id

ABSTRACT
The performance analysis is an essential process for evaluating the efficiency and effectiveness of a system or software. This study focused on conducting performance analysis of the mobile game "Idle Breeder" on the Android platform. The analysis specifically examined rendering computation, scripts, physics, animation, garbage collection, global illumination, user interface, and other aspects using Unity Profiler software. The findings of the study revealed that the game performed well in various testing scenarios, providing valuable insights for mobile game developers to optimize their game's performance on the Android platform using Unity Profiler. However, it is important to consider that environmental conditions and hardware factors may also influence overall game performance. This research contributes to the advancement of knowledge in improving the performance of mobile games and encourages further exploration in this area. The results of this study can be useful for game developers, researchers, and practitioners in the field of mobile gaming to enhance the performance of their games and ultimately provide a better gaming experience for users.

Keywords **-** Analysis, Performance, Profiler, Unity

1. Introduction

Mobile gaming has become increasingly popular over the years, thanks to the rapid development of mobile technology [1]. One of the most critical aspects of developing mobile games is optimizing the game's performance, especially on the Android platform, which has various device configurations [2]. As the number of Android devices in the market continues to increase, game developers need to ensure that their games run optimally across all devices [3].

Idle Breeder is a mobile game developed for the Android platform that simulates a virtual pet breeding experience. As with any mobile game, performance is critical to providing a seamless and enjoyable gaming experience [4]. Therefore, it is essential to conduct a performance analysis of the game to identify any areas of improvement that can be made to optimize its performance. This study aims to analyze the performance of Idle Breeder using the Unity Profiler software and suggest recommendations for improving its performance on the Android platform [5]. The findings of this study will help mobile game developers to enhance their game's performance and provide a better gaming experience for users.

2. Research Method

The research methodology used by the author is qualitative and quantitative methods by measuring optimization standards. To find out the optimal result value, the author uses the *Optimization Cycle* stage from *makaka.org* [6].

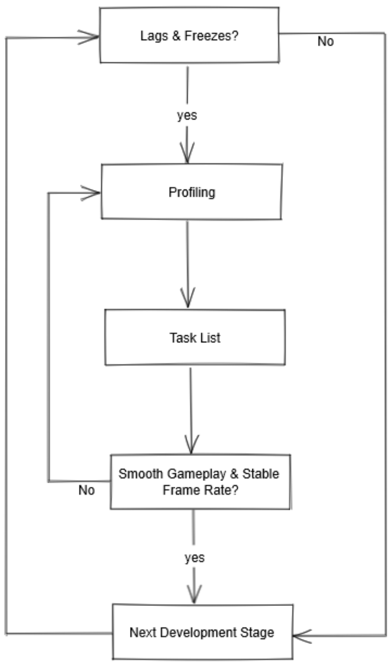
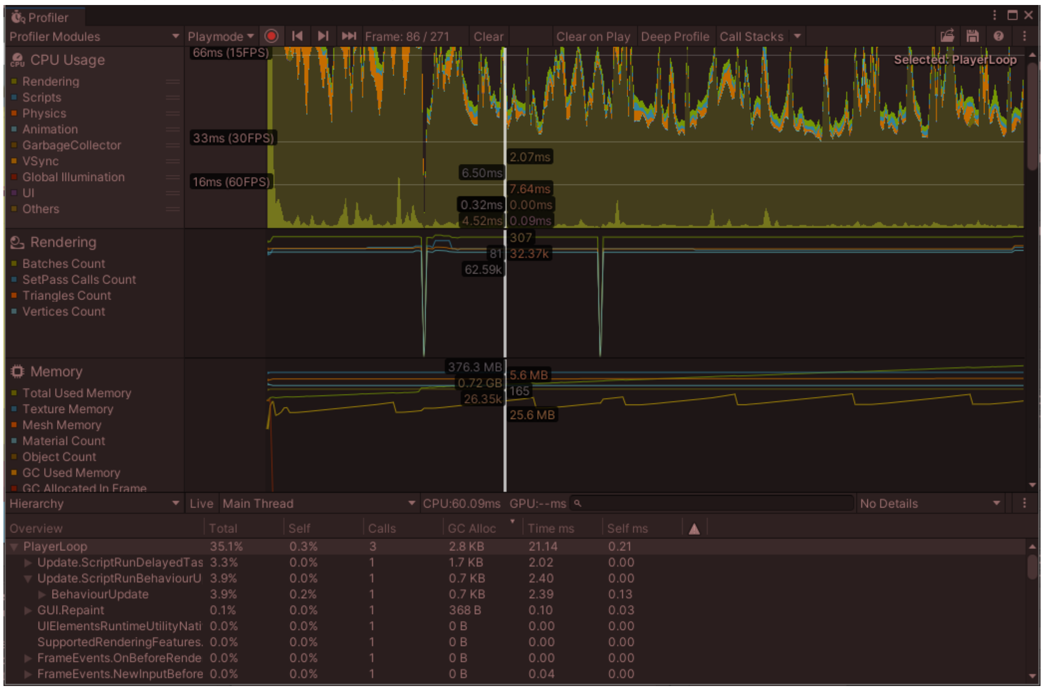


Figure 1. Optimization Cycle

There are 5 stages in performing optimization from Figure 1. The first stage, is a stage to find out whether the game needs optimization or not. If when playing the game there is *lag* or *freeze* then the game needs to be optimized.

The next stage is the *Profiling* stage, the author uses *Unity Profiler* to help the optimization process and read every process run by the game specifically [7].



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
PlayerLoop	35.1%	0.3%	3	2.8 KB	21.14	0.21
Update.ScriptRunDelayedTas	3.3%	0.0%	1	1.7 KB	2.02	0.00
Update.ScriptRunBehaviourU	3.9%	0.0%	1	0.7 KB	2.40	0.00
BehaviourUpdate	3.9%	0.2%	1	0.7 KB	2.39	0.13
GUI.Repaint	0.1%	0.0%	1	368 B	0.10	0.03
UIElementsRuntimeUtilityNati	0.0%	0.0%	1	0 B	0.00	0.00
SupportedRenderingFeatures.	0.0%	0.0%	1	0 B	0.00	0.00
FrameEvents.OnBeforeRende	0.0%	0.0%	1	0 B	0.00	0.00
FrameEvents.NewInputBefore	0.0%	0.0%	1	0 B	0.04	0.00

Figure 2. Unity Profiler

The first is *CPU Usage*, which is a colorful bar that indicates the performance of each process in the game, such as *Rendering*, *Script*, *Physics*, *Animation*, *GarbageCollector*, *VSync*, *Global Illumination*, *UI*, and *Others* [8].

The author sampled one frame of the graph, the details of which are clarified in the *Hierarchy* option. Hierarchy displays the performance generated by the game and is clarified by providing details, such as *Total*, *Self*, *Calls*, *GC Alloc*, *Time ms*, and *Self ms*.

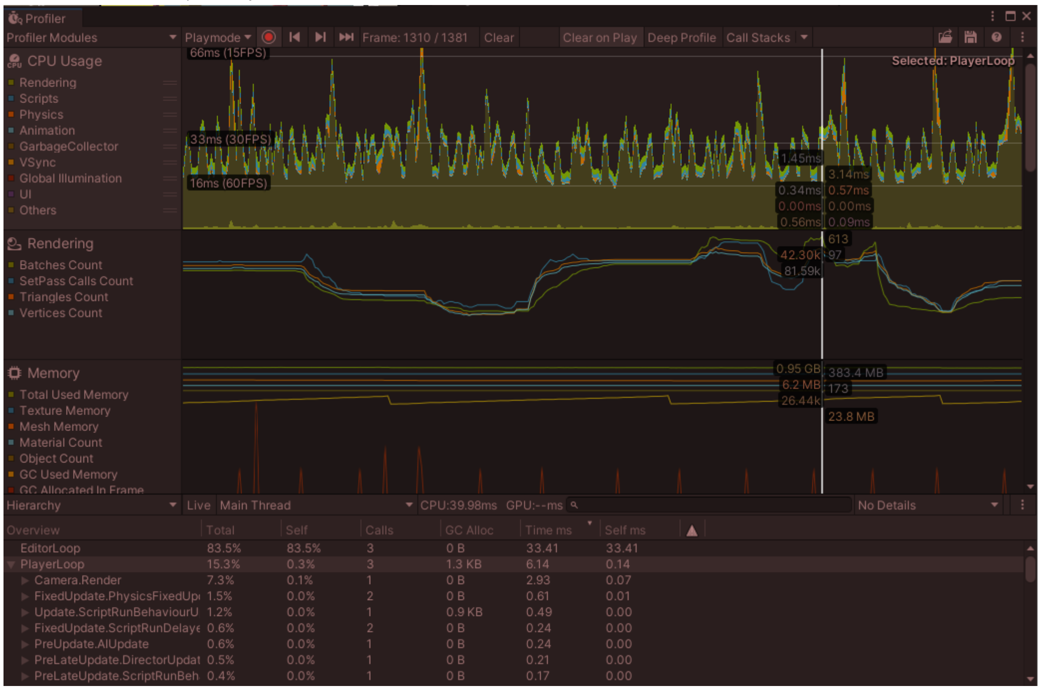
After *profiling*, the next step is to do a *Task List*, which determines which parts to optimize. There are various ways to perform optimization, such as *Batching*, *Object Pooling*, *GPU Instancing*, *Lightmapping*, *Texture*, *Occlusion Culling*, *Scripting*, and *Tris and Vertexes* [9]. Each of these processes can optimize game performance [10].

Batching is a method of combining identical objects into one, making it easier for the CPU to recognize the objects without performing a new object reading analysis. *Object Pooling* is a method to minimize the number of objects created repeatedly so as to reduce CPU performance. *GPU Instancing* is a process that makes the GPU read the object, so that it does not burden the CPU to read the object's texture. *Texture* is the image quality of an object, the more complex the texture, the more it will affect game performance. *Occlusion Culling* is a method to eliminate objects that are not visible when the camera is rendering to reduce the memory used to read the entire map. *Scripting* is the creation of commands that affect performance when multiple heavy commands cause repetitive behavior of an object, which can ultimately reduce game performance. *Tris and Vertexes* are triangles in each mesh generated and read by the camera in a *frame*.

The next stage is to play the game and compare with the previous performance to determine if the game still lags or freezes. If it still *lags* or *freezes*, it will return to the *Profilling* stage.

3. Result and Analysis

The analyzed game is a *top-down idle* game built using the *Unity Engine*. The following are the analysis results from *Unity Profiler*.

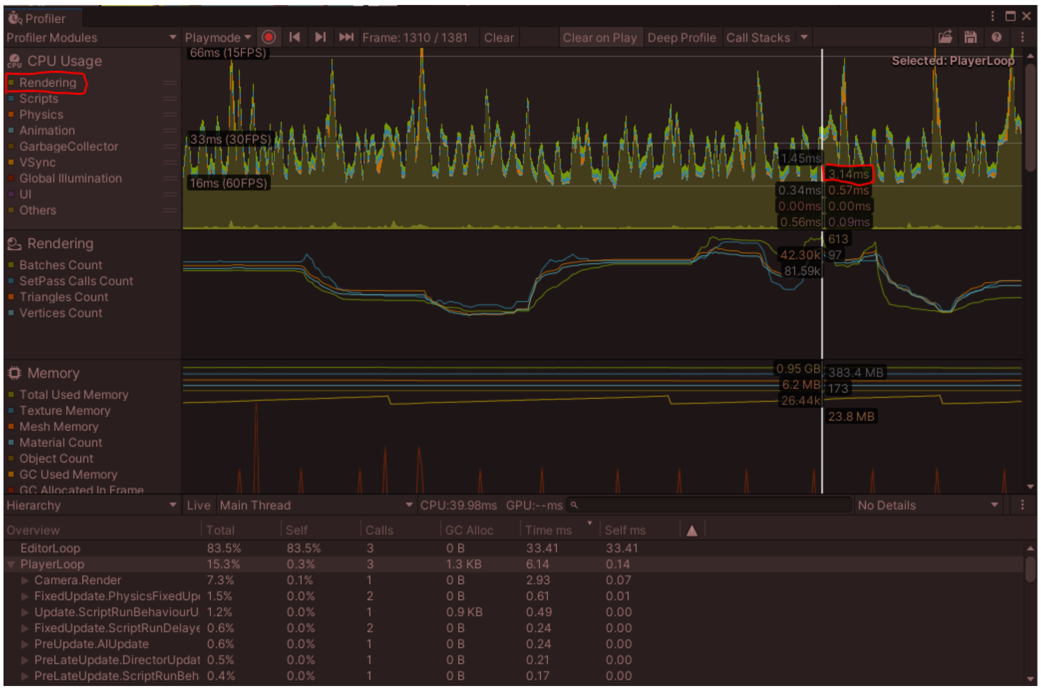


Hierarchy | Live | Main Thread | CPU:39.98ms GPU:--ms

Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
EditorLoop	83.5%	83.5%	3	0 B	33.41	33.41
PlayerLoop	15.3%	0.3%	3	1.3 KB	6.14	0.14
Camera.Render	7.3%	0.1%	1	0 B	2.93	0.07
FixedUpdate.PhysicsFixedUp	1.5%	0.0%	2	0 B	0.61	0.01
Update.ScriptRunBehaviourU	1.2%	0.0%	1	0.9 KB	0.49	0.00
FixedUpdate.ScriptRunDelaye	0.6%	0.0%	2	0 B	0.24	0.00
PreUpdate.AIUpdate	0.6%	0.0%	1	0 B	0.24	0.00
PreLateUpdate.DirectorUpdat	0.5%	0.0%	1	0 B	0.21	0.00
PreLateUpdate.ScriptRunBeh	0.4%	0.0%	1	0 B	0.17	0.00

Figure 3. Game Profiling Results

A. Rendering Analysis



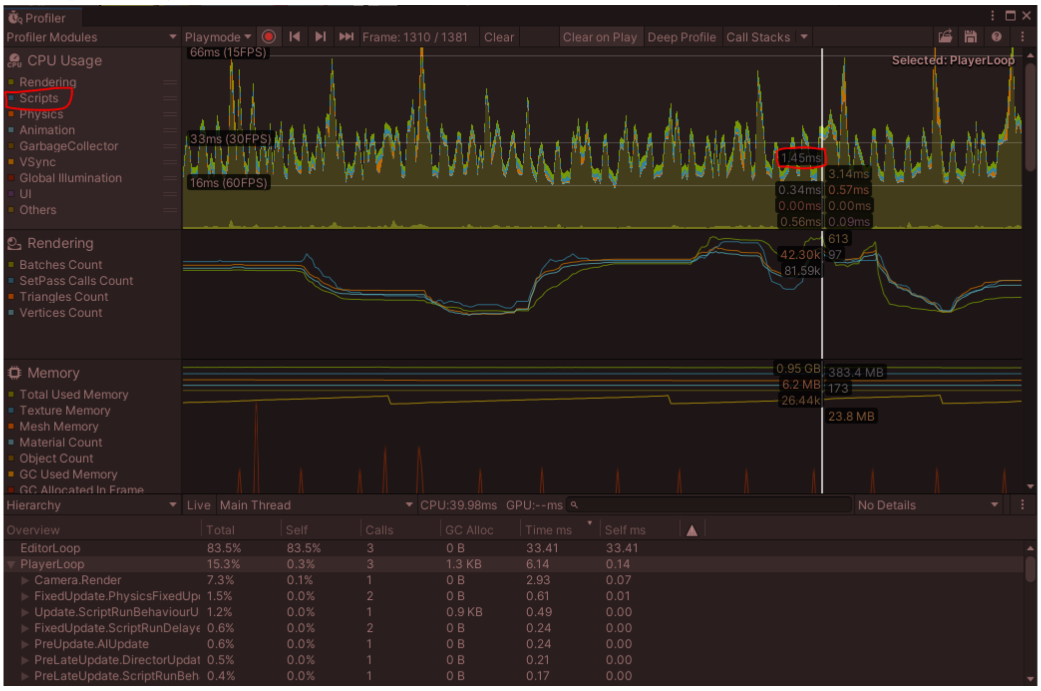
Hierarchy | Live | Main Thread | CPU:39.98ms GPU:--ms

Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
EditorLoop	83.5%	83.5%	3	0 B	33.41	33.41
▼ PlayerLoop	15.3%	0.3%	3	1.3 KB	6.14	0.14
▶ Camera.Render	7.3%	0.1%	1	0 B	2.93	0.07
▶ FixedUpdate.PhysicsFixedUp	1.5%	0.0%	2	0 B	0.61	0.01
▶ Update.ScriptRunBehaviourU	1.2%	0.0%	1	0.9 KB	0.49	0.00
▶ FixedUpdate.ScriptRunDelaye	0.6%	0.0%	2	0 B	0.24	0.00
▶ PreUpdate.AIUpdate	0.6%	0.0%	1	0 B	0.24	0.00
▶ PreLateUpdate.DirectorUpdat	0.5%	0.0%	1	0 B	0.21	0.00
▶ PreLateUpdate.ScriptRunBeh	0.4%	0.0%	1	0 B	0.17	0.00

Figure 4. Game Profiling Results of the Rendering section

Based on Figure 4, it can be seen that the *Rendering* computation result on a particular *frame* takes 3.14 ms. The resulting computation time is quite good because for optimal game quality, the entire process must have a minimum of 60 FPS with a total computation of 16 ms. Therefore, this result shows that there is still enough room for computation of other aspects that require more time.

B. Scripts Analysis



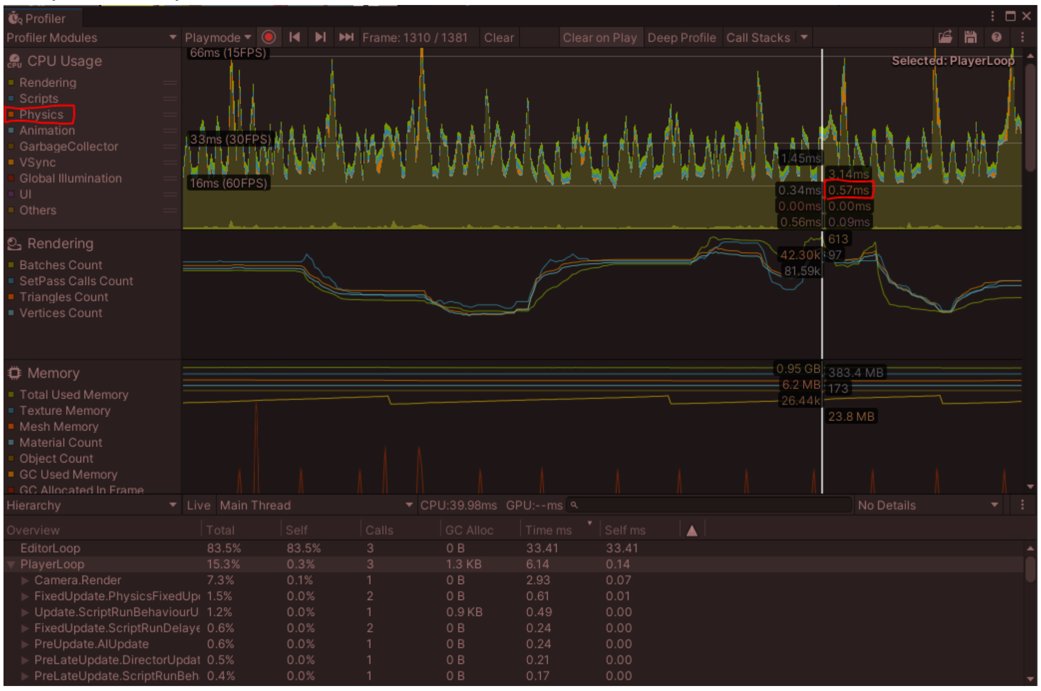
Hierarchy | Live | Main Thread | CPU:39.98ms GPU:--ms

Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
EditorLoop	83.5%	83.5%	3	0 B	33.41	33.41
▼ PlayerLoop	15.3%	0.3%	3	1.3 KB	6.14	0.14
▶ Camera.Render	7.3%	0.1%	1	0 B	2.93	0.07
▶ FixedUpdate.PhysicsFixedUp	1.5%	0.0%	2	0 B	0.61	0.01
▶ Update.ScriptRunBehaviourU	1.2%	0.0%	1	0.9 KB	0.49	0.00
▶ FixedUpdate.ScriptRunDelaye	0.6%	0.0%	2	0 B	0.24	0.00
▶ PreUpdate.AIUpdate	0.6%	0.0%	1	0 B	0.24	0.00
▶ PreLateUpdate.DirectorUpdat	0.5%	0.0%	1	0 B	0.21	0.00
▶ PreLateUpdate.ScriptRunBeh	0.4%	0.0%	1	0 B	0.17	0.00

Figure 5. Game Profiling Results of the Scripts section

In Figure 5, it can be seen that the time required for computing *Scripts* is 1.45 ms, this result shows that the computation of Scripts in the game being analyzed is quite good, because it still has enough room to compute other aspects.

C. Physics Analysis

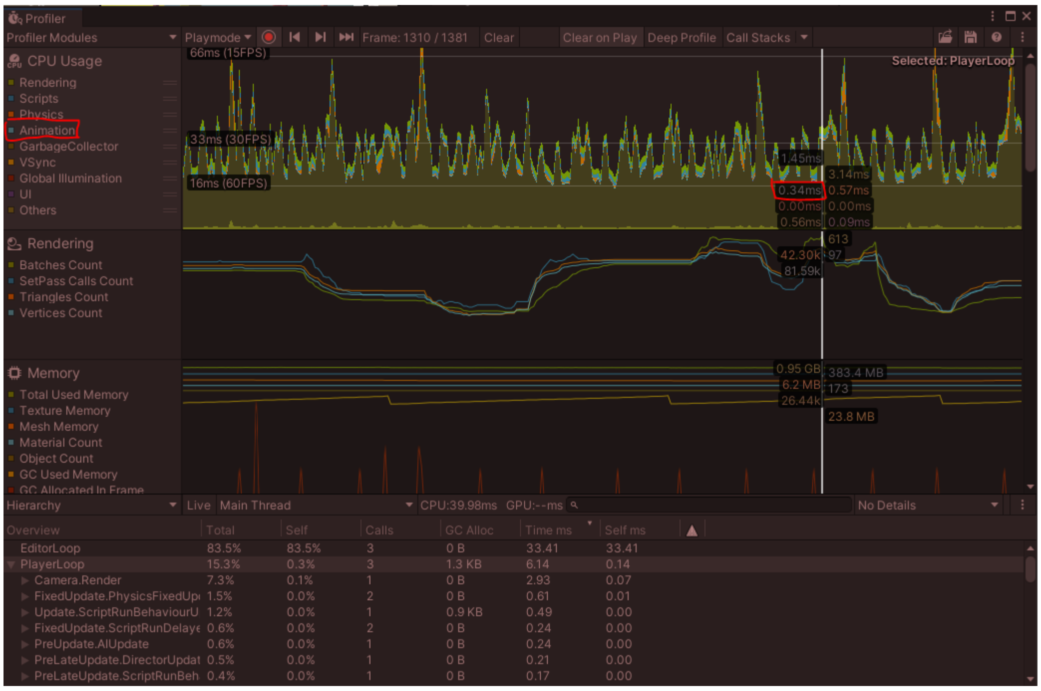


Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
EditorLoop	83.5%	83.5%	3	0 B	33.41	33.41
PlayerLoop	15.3%	0.3%	3	1.3 KB	6.14	0.14
Camera.Render	7.3%	0.1%	1	0 B	2.93	0.07
FixedUpdate.PhysicsFixedUp	1.5%	0.0%	2	0 B	0.61	0.01
Update.ScriptRunBehaviourU	1.2%	0.0%	1	0.9 KB	0.49	0.00
FixedUpdate.ScriptRunDelaye	0.6%	0.0%	2	0 B	0.24	0.00
PreUpdate.AIUpdate	0.6%	0.0%	1	0 B	0.24	0.00
PreLateUpdate.DirectorUpdat	0.5%	0.0%	1	0 B	0.21	0.00
PreLateUpdate.ScriptRunBeh	0.4%	0.0%	1	0 B	0.17	0.00

Figure 6. Game Profiling Results for the Physics section

Computational results in Figure 6, the time required for *Physics* computation is quite efficient, amounting to 0.57 ms, this result shows that the *Physics* computation is quite optimal, and provides enough space to perform computations on other aspects of the game.

D. Animation Analysis

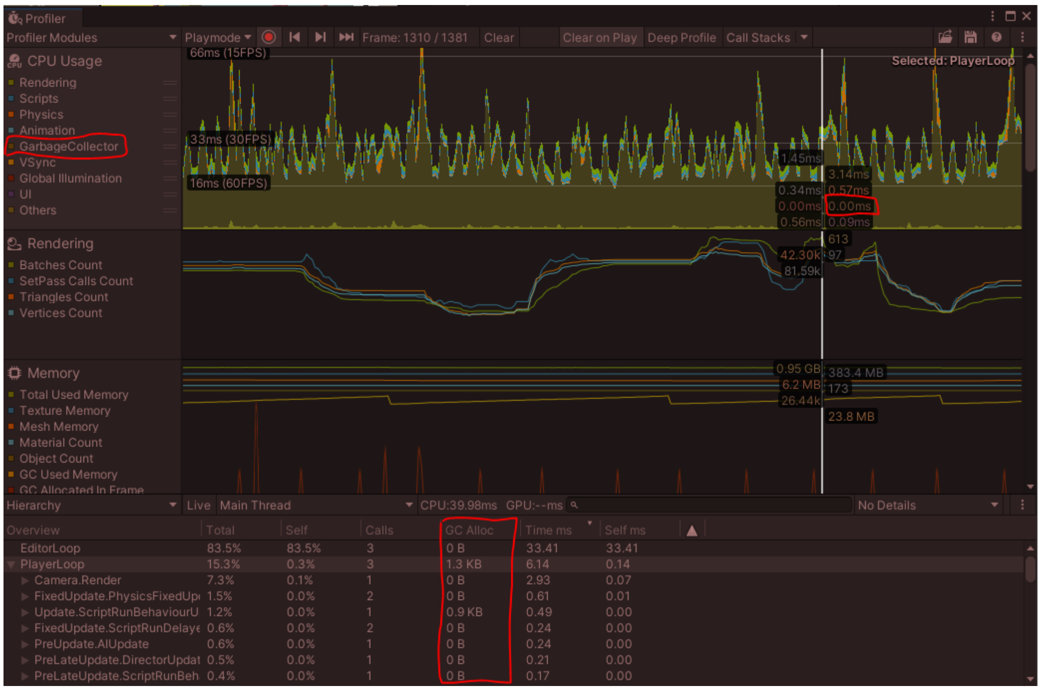


Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
EditorLoop	83.5%	83.5%	3	0 B	33.41	33.41
PlayerLoop	15.3%	0.3%	3	1.3 KB	6.14	0.14
Camera.Render	7.3%	0.1%	1	0 B	2.93	0.07
FixedUpdate.PhysicsFixedUp	1.5%	0.0%	2	0 B	0.61	0.01
Update.ScriptRunBehaviourU	1.2%	0.0%	1	0.9 KB	0.49	0.00
FixedUpdate.ScriptRunDelaye	0.6%	0.0%	2	0 B	0.24	0.00
PreUpdate.AIUpdate	0.6%	0.0%	1	0 B	0.24	0.00
PreLateUpdate.DirectorUpdat	0.5%	0.0%	1	0 B	0.21	0.00
PreLateUpdate.ScriptRunBeh	0.4%	0.0%	1	0 B	0.17	0.00

Figure 7. Game Profiling Results for the Animation section

When analyzing Figure 7, it was found that the time taken to compute *Animation* was 0.34 ms. This result shows that the *Animation* computation is very good as it still has a lot of room left to compute other aspects of the game.

E. Garbage Collector Analysis



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
EditorLoop	83.5%	83.5%	3	0 B	33.41	33.41
PlayerLoop	15.3%	0.3%	3	1.3 KB	6.14	0.14
Camera.Render	7.3%	0.1%	1	0 B	2.93	0.07
FixedUpdate.PhysicsFixedUp	1.5%	0.0%	2	0 B	0.61	0.01
Update.ScriptRunBehaviourU	1.2%	0.0%	1	0.9 KB	0.49	0.00
FixedUpdate.ScriptRunDelaye	0.6%	0.0%	2	0 B	0.24	0.00
PreUpdate.AIUpdate	0.6%	0.0%	1	0 B	0.24	0.00
PreLateUpdate.DirectorUpdat	0.5%	0.0%	1	0 B	0.21	0.00
PreLateUpdate.ScriptRunBeh	0.4%	0.0%	1	0 B	0.17	0.00

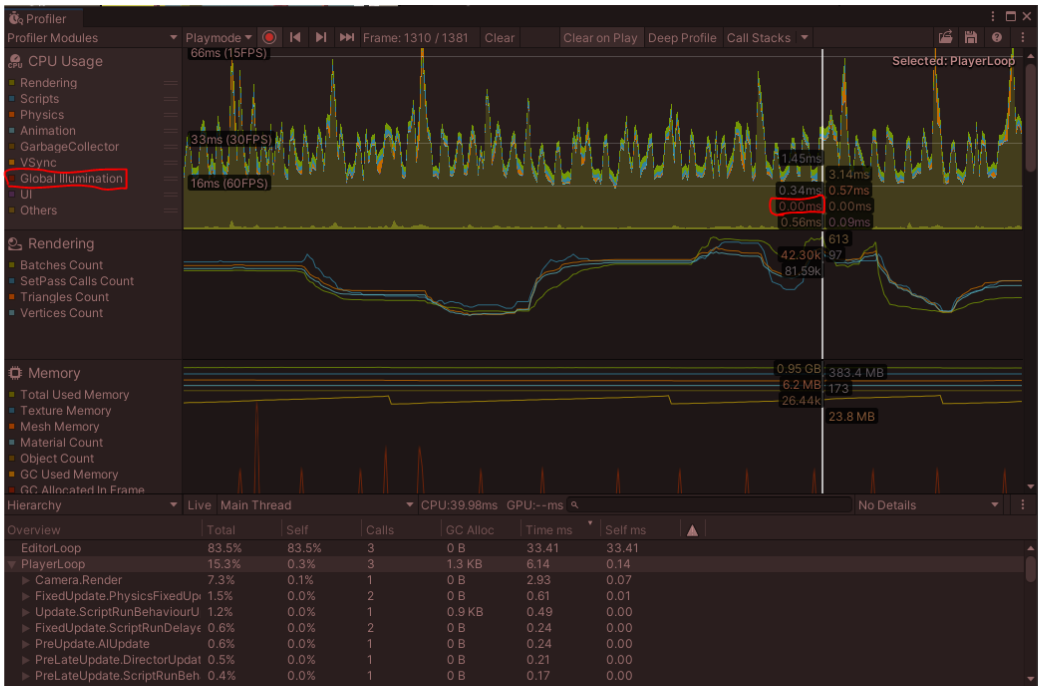
Figure 8. Game Profiling Results of Garbage Collector section

In Figure 8, the *Garbage Collector* computation results show a very efficient time, requiring only 0.00 ms. However, when looking at the *Hierarchy* section, it can be seen that the *GC Alloc* reaches 1.3 KB. This can be a fatal problem if it happens every *frame*, because it can cause memory performance to become heavier and interfere with the player's experience in playing the game. Therefore, it is necessary to check thoroughly to ensure that there are no problems with memory at every *frame*, so as to maintain the quality and user experience of playing the game.

F. VSync Analysis

In *Profiling* the game, the author did not compute *VSync* to avoid its influence on the *frame rate* in the game and its impact on the results displayed in the *Profiler*. This is done to ensure that the results obtained by *Profiling* truly represent the performance of the game without any confounding factors such as *VSync*.

G. Global Illumination Analysis

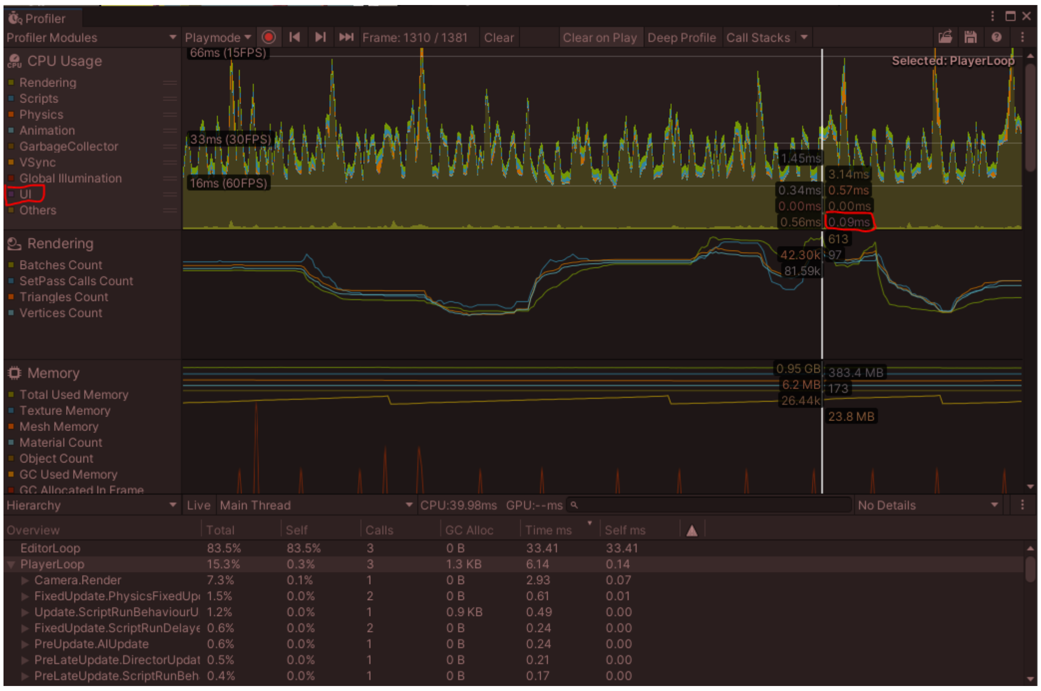


Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
EditorLoop	83.5%	83.5%	3	0 B	33.41	33.41
PlayerLoop	15.3%	0.3%	3	1.3 KB	6.14	0.14
Camera.Render	7.3%	0.1%	1	0 B	2.93	0.07
FixedUpdate.PhysicsFixedUp	1.5%	0.0%	2	0 B	0.61	0.01
Update.ScriptRunBehaviourU	1.2%	0.0%	1	0.9 KB	0.49	0.00
FixedUpdate.ScriptRunDelaye	0.6%	0.0%	2	0 B	0.24	0.00
PreUpdate.AIUpdate	0.6%	0.0%	1	0 B	0.24	0.00
PreLateUpdate.DirectorUpdat	0.5%	0.0%	1	0 B	0.21	0.00
PreLateUpdate.ScriptRunBeh	0.4%	0.0%	1	0 B	0.17	0.00

Figure 9. Game Profiling Results of Global Illumination section

In Figure 9, the *Global Illumination* computation time shows a time of 0.00 ms. This result shows excellent computational efficiency as it still leaves plenty of time to perform computations on other aspects of the game. This is important for improving game performance and providing a better gaming experience for the player.

H. UI Analysis

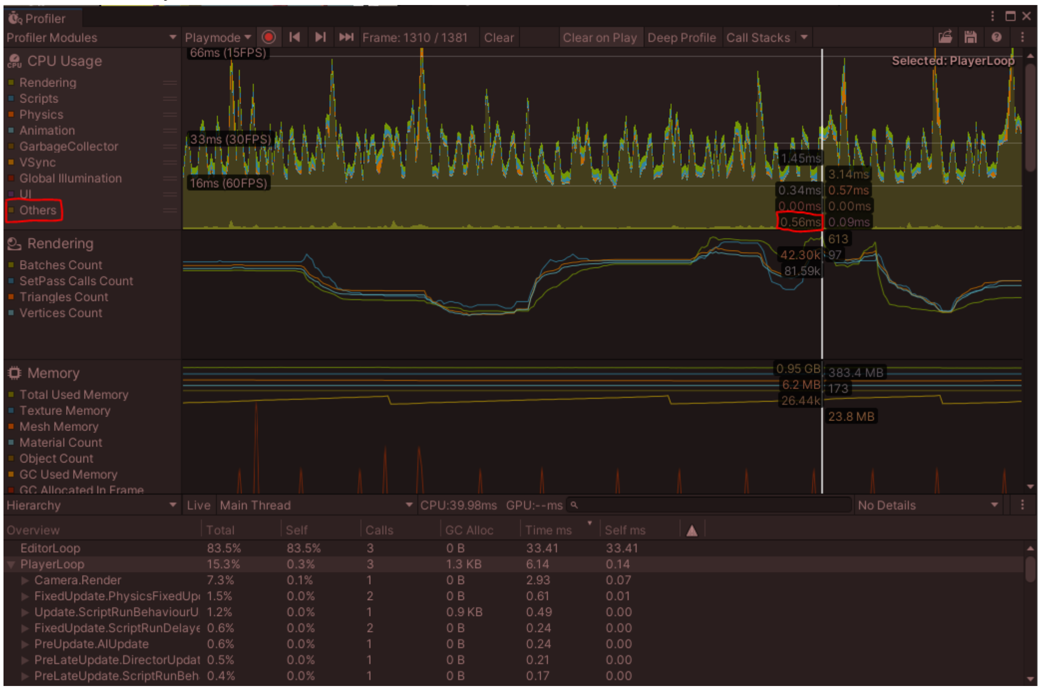


Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
EditorLoop	83.5%	83.5%	3	0 B	33.41	33.41
PlayerLoop	15.3%	0.3%	3	1.3 KB	6.14	0.14
Camera.Render	7.3%	0.1%	1	0 B	2.93	0.07
FixedUpdate.PhysicsFixedUp	1.5%	0.0%	2	0 B	0.61	0.01
Update.ScriptRunBehaviourU	1.2%	0.0%	1	0.9 KB	0.49	0.00
FixedUpdate.ScriptRunDelaye	0.6%	0.0%	2	0 B	0.24	0.00
PreUpdate.AIUpdate	0.6%	0.0%	1	0 B	0.24	0.00
PreLateUpdate.DirectorUpdat	0.5%	0.0%	1	0 B	0.21	0.00
PreLateUpdate.ScriptRunBeh	0.4%	0.0%	1	0 B	0.17	0.00

Figure 10. Game Profiling Results of UI section

In Figure 10, it can be seen that the computation time required to perform the *UI* task is 0.09 ms. This result indicates that the *UI* computation in the game is still very efficient and leaves enough time to perform computations on other aspects that may be more complex and take longer to complete. Therefore, this result shows that the game has done a good job of optimizing the *UI* tasks, thereby improving the overall performance of the game.

I. Others Analysis



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
EditorLoop	83.5%	83.5%	3	0 B	33.41	33.41
PlayerLoop	15.3%	0.3%	3	1.3 KB	6.14	0.14
Camera.Render	7.3%	0.1%	1	0 B	2.93	0.07
FixedUpdate.PhysicsFixedUp	1.5%	0.0%	2	0 B	0.61	0.01
Update.ScriptRunBehaviourU	1.2%	0.0%	1	0.9 KB	0.49	0.00
FixedUpdate.ScriptRunDelaye	0.6%	0.0%	2	0 B	0.24	0.00
PreUpdate.AIUpdate	0.6%	0.0%	1	0 B	0.24	0.00
PreLateUpdate.DirectorUpdat	0.5%	0.0%	1	0 B	0.21	0.00
PreLateUpdate.ScriptRunBeh	0.4%	0.0%	1	0 B	0.17	0.00

Figure 11. Game Profiling Results of Others section

Figure 11 shows the computation results for the *Others* aspect, which takes 0.56 ms. This relatively short computation time indicates good efficiency on the *Others* aspect, so there is still plenty of time to perform computations on other aspects of the game.

4. Conclusion

Based on the results of the analysis that has been done in this paper, it can be concluded that the computational performance of the Idle Breeder game is relatively good. This can be seen from the relatively light computation time, which does not burden the computer system used. However, keep in mind that there are several other factors that can affect the overall performance of the game, such as environmental factors and the hardware used. Therefore, further research is needed to optimize the overall performance of the game.

References

[1] Yuanzhe, L. Zezheng, X. Yu, H. Peng, X. Jingyan, L. (2022). Psychosocial Impacts of Mobile Game on K12 Students and Trend Exploration for Future Educational Mobile Games. Frontiers in Education.

[2] Peng, H. Kai, Z. (2014). Strategy research on the performance optimization of 3D mobile game development based on Unity. Journal of Chemical and Pharmaceutical Research.

[3] Koulaxidis, G. Xinogalos, S. (2022). Improving Mobile Game Performance with Basic Optimization Techniques in Unity. Multidisciplinary Digital Publishing Institute.

[4] Hort, M. Kechagia, M. Sarro, F. Harman, M. (2021). A Survey of Performance Optimization for Mobile Applications. University College London.

[5] Unity, Team. (2021). Optimize your mobile game performance. Unity For Games: Unity Technologies.

[6] Sirota, A. (2023). Unity Optimization Tips: Mobile & Desktop. Makaka Games. [Online]. Available: https://makaka.org/unity-tutorials/optimization. [Accessed: 25-Mar-2023].

[7] Ferriera, C. Hughes, S. (2015). Unity* Optimization Guide for x86 Android*. Intel Corporation: Intel.

[8] Blåfield, J. (2021). Optimizing mobile games in a Unity environment [Thesis]. Finland: JAMK University of Applied Sciences.

[9] Siarkowski, K. Sprawka, P. Wójcik, MP. (2017). Methods of optimizing the performance of the Unity 3D engine based on a third-person perspective game. Journal of Computer Sciences Institut.

[10] Riwinoto. Tan, W. (2021). ANALISIS PERFORMA PROTOTYPE GAME PADA PLATFORM ANDROID. JOURNAL OF APPLIED MULTIMEDIA AND NETWORKING.