Written Report

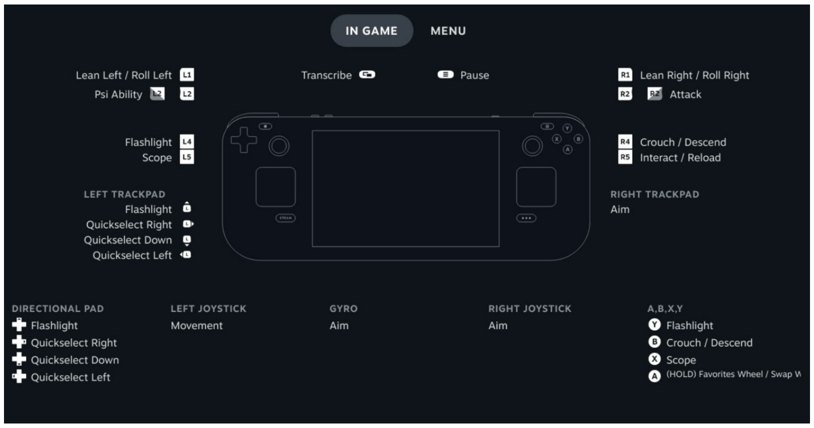
1 Project Implementation

The unreal project that I had developed was targeted for a Windows platform with a fairly better graphic processor and the same implementation for a Valve Steam Deck platform could pose some serious challenges. This project can be divided into separate units which were developed independently at different stages of the project development.

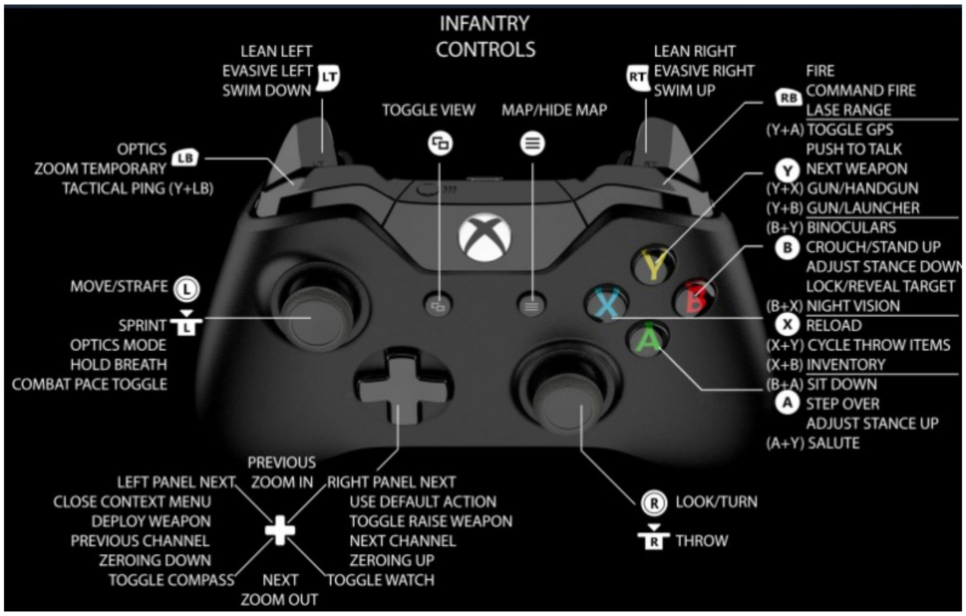
1.1 The Drone

I have used the package UAV MQ-1 Predator (West) which is a free resource available in Unreal Engine Marketplace for the creation of the drone in this project. The drone blueprint makes use of the skeletal mesh as well as the materials and other meshes from this package. Some of the interesting factors of the assets in this package is that they all use very high quality vehicle meshes and they use emissive material driven lights for an increased performance. These meshes and materials can showcase an excellent visual in the Windows platform but it might not be as visually appealing in Steam Deck.

However, the controller input in the Steam Deck is almost similar to an Xbox controller and in this project the inputs are mapped for playing with Keyboard and mouse, as well as an Xbox controller. This means that the player can easily control the drone while playing in a Steam Deck without much serious modifications to the already implemented input mappings. For implementing the input mapping for Steam Deck, the developer only need to add the key mappings for the Steam Deck.



IMG 1.1 : Steam Deck Controller (Valve Corporation, n.d.)



IMG 1.2 : Xbox Controller (Valve Corporation, 2020)

1.2 Other Actors and Meshes

A good portion of the actors in this project, meshes for the terrain, road networks and buildings are made from using Blender. The .fbx file exported from Blender is imported in Unreal Engine and used as static meshes and appropriate materials for these meshes. Also, certain actors like Directional Light and Sky Light is also used to get a good lighting in the level. Skylight actors simulate the indirect lighting that comes from the sky, by using a high-resolution cubemap that captures the surrounding environment. This allows for realistic lighting and reflections in the virtual environment. These lightings allows for objects to cast and receive shadows from other objects in the scene, creating a more believable and immersive experience. Rendering of these lighting and shadows require good computing power.

Although Steam Deck make use of 8 RDNA 2 CUs, 1.0-1.6GHz (up to 1.6 TFlops FP32) GPU, Zen 2 4c/8t, 2.4-3.5GHz (up to 448 GFlops FP32) CPU and 16 GB LPDDR5 on-board RAM (5500 MT/s quad 32-bit channels) (Valve Corporation, 2022) in it’s most powerful model, the rendering of these lights and shadows can not be as visually appealing as a Windows or Xbox platform. Since Steam Deck supports a refresh rate of 40Hz, users can enjoy the perfect middle-ground between 30 and 60 FPS, making this one of the biggest Steam Deck improvements since its release. However, Xbox Series X|S console family can support a frame rate of up to 120FPS, providing a much smoother gameplay experience.

1.3 Metahumans

In this project, metahumans are used for the crowds. Metahumans are spawned in the city using a Mass Spawner actor. Steam Deck's ability to render metahumans specifically would depend on the quality of the models and animations created for them within the Unreal Engine. Creating believable and realistic in Steam Deck can present several challenges. Metahuman characters require fairly high levels of detail. Also, metahuman characters require realistic simulations of physics and materials, such as clothing and hair. This can be a strain on the Steam Deck GPU.

2 Optimisation Techniques

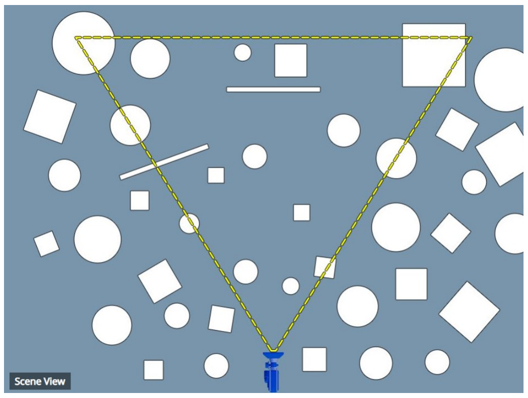
There are several optimization techniques that can be implemented in the Unreal Engine when developing games for Valve's Steam Deck. Some of these include:

2.1 Level of Detail (LOD) systems

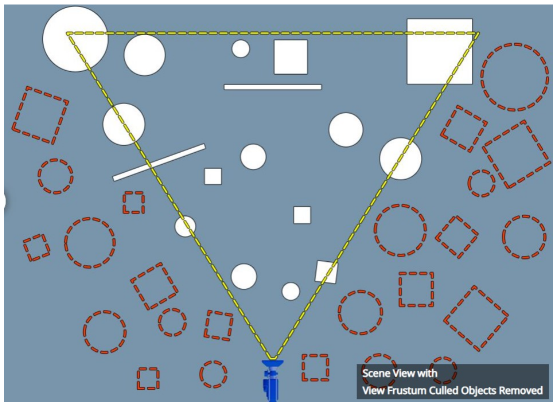
LOD Systems can be used to reduce the level of detail of the objects as we move away from these objects. This can help in optimizing the game performance by reducing the number of vertices that need to be processed at any given time, it also avoids the micro triangles problem and potentially looks better for objects placed further away in the scene. As an object goes further from camera we can see less of the detail of that object. "View-dependent LOD uses current view parameters to represent good quality of current view. A single object may thus spans several levels of detail" (Tan, et al., 2008).Visually, it will be hard to see difference between same object that consists of 200 triangles and another object with 2000 triangles from 20 meters away, so there's little point in having that many triangles that add nothing to the scene. A good rule of thumb is to reduce the number of triangles by 50% in each level of detail.

2.2 Occlusion Culling

Occlusion culling is a method which determines which objects in the scene are visible from the player's point of view and only rendering those objects. This can help to improve the performance. Unreal Engine provides methods of culling for visibility and occlusion. If we start with only what the camera can see from its position, there are only a handful of objects visible. However, we know that there are a lot of objects that make up the scene that just are not visible from the camera position. However, the objects that are outside the camera's view can be culled. (IMG 2.1) Culled objects need not be rendered, leaving only a few objects rendered. (IMG 2.2) These objects need to be checked for visibility. So, a query will be sent to the GPU to test the visibility state of each of these objects. Those objects whose visibility is blocked by another object is culled from view.



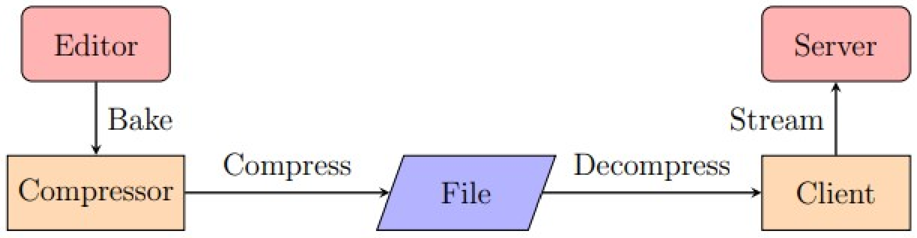
IMG 2.1 : Scene view with all objects (Epic Games, n.d.)



IMG 2.2: Scene View with objects culled (Epic Games, n.d.)

2.3 Lightmap baking

Pre-calculating lighting and shadows for static objects in the scene using lightmap baking can help to improve performance by reducing the number of calculations that need to be done in real-time. However, this will remove dynamic casting of shadows as the light will be static. This process is done by initially rendering the scene from the point of view of a light source and then saving the resulting information in a texture called a lightmap.(IMG 2.3) Lightmap can be created either by using Lightmap Auto-Generation tool in Unreal Engine or by creating a custom lightmap with UV editing tools. Creation of custom lightmaps for large projects with large number of actors can be time-consuming. This challenge can be rectified by using an auto-generated light map.


$$\begin{tikzcd}
\text{Editor} \arrow[d, "\text{Bake}"] & & \text{Server} \\
\text{Compressor} \arrow[r, "\text{Compress}"] & \text{File} \arrow[r, "\text{Decompress}"] & \text{Client} \arrow[u, "\text{Stream}"']
\end{tikzcd}$$

IMG 2.3: Lightmap baking (Olsson, 2015)

It is important to note that this is not a constant flow as the left part (until File) is executed during development while the right part is executed when a scene is loaded in the game. “The compression can be arbitrarily complex while the decompression must occur in near real-time” (Olsson, 2015).

2.4 Optimizing the game's assets

Usage of the correct formats and compression settings for the game's assets, as well as reducing the number of assets used in the scene, can help to improve performance. Repetitive rendering of the same assets can be avoided to optimize the performance. This also include optimization of the codes used in the game. This can be achieved by reducing the number of function calls, using more efficient data structures and multithreading. These optimization can be done by the game developer as well as the game designer. Programmers should look for any bad code which might take more execution time and also implement any code optimization technique which would speed up code execution. However, as game s are more graphics oriented, graphic designers should optimize the game assets from the start of the development process. As Laura(2013) mentions it is important to create 3D models that can be used multiple times in a level of a game in order to make the most out of the given memory. It is also not a very good idea to make too big units that are to be placed in the level individually as the reusability of such units decreases. (Lohikosk, 2013)

3 Optimized Performance on Steam Deck

In order to optimize the game for a Valve Steam Deck platform, LOD settings should be implemented. This means that the assets that are closer in the camera view will be rendered with higher details whereas the assets which are further away in the camera view will have a much lesser level of detail.

If we implement occlusion culling for the optimization of the scenes, any assets that are not visible in the camera will not be rendered and this can increase the performance of the game in Steam Deck.

Dynamic shadows change in real-time based on the movement of objects or lights in the scene. By implementing baked lightmaps for the optimization of the games, dynamic shadows will not be generated in the game. The players could only see pre-calculate the lighting and shadows. Since the lightmap is pre-calculated, it will not change with the movement of objects or lights.

Furthermore, by optimizing the game assets and the codes used in the game, the load on the CPU and GPU of the Steam Deck will be reduced. It can help in increasing the frame rate of the game which makes the game feel more smooth and responsive. It can also reduce the time it takes to load all the assets in the game and in turn reduce the time needed to start the game.

4 References

Valve Corporation, n.d. *Steam Controller.* [Online]
Available at: https://steamcommunity.com/sharedfiles/filedetails/?id=2804823261
[Accessed 12 January 2023].

Epic Games, n.d. *Visibility and Occlusion Culling: An overview of Visibility and Occlusion Culling methods in Unreal Engine 4.* [Online]
Available at: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/VisibilityCulling/
[Accessed 12 January 2023].

Lohikosk, L., 2013. *Optimization of 3D Game Models : A qualitative research study in Unreal,* s.l.: Institutionen för naturvetenskap.

Olsson, T., 2015. *Baking And Compression For Dynamic Lighting Data,* Sweden: Orebro University.

Tan, K. H., Bade, A. & Daman, D., 2008. *A review on level of detail,* s.l.: Penerbit UTM Press.

Valve Corporation, 2020. *Xbox Controller Guide.* [Online]
Available at: https://steamcommunity.com/sharedfiles/filedetails/?id=759868839

Valve Corporation, 2022. *Tech Specs.* [Online]
Available at: https://www.steamdeck.com/en/tech
[Accessed 12 january 2023].