Comp 280 - Creative Computing Tasks

Networking

Features

Some non-networking features that are a part of the gameplay:

* Coping game object mesh in runtime.
* Mesh generation (generates a mesh from the data of a player copied mesh)

Networking has been implemented in the game “The Hued Trails”. It is in the form of:

* Spawning multiple players in a scene world when these players connect to the server.
* Syncing player position and rotation
* Syncing wand rotation (To show where the player is looking)
* Syncing a pointer object for every player (to show where the players are pointing to)
* Generating and syncing two sets of 15 tiles from the host and shearing it to everyone else.
* Spawning copied game objects (pasting them)
* Syncing pasted objects. The position, material, mesh.

Network Code

With the use of 13 scripts (Not counting the required “Forge Networking Remastered” package) This project was complete.

CopyableObjectsFollowingPlayer.cs makes the objects that can be copied follow the player as they progress further in the level.

CopyDate.cs contains all the data from the copied object chosen by the player.

CopyObject.cs contains all the functionality needed to take the player input and copy the object that has been pointed to.

GameLogic.cs spawns the required game objects like the player and the copied game objects when pasted.

LocationOfCopy.cs handles the spawning of the game object at the correct position, but only for the owner (the player who spawns it).

PasteObject.cs handles the assigning of the copied data and assigns it to the the already existent mesh component from the Copy prefab game object.

PlayerLook.cs handles the mouse movement and syncs it on the network so the other player can see where you are looking.

PlayerMovement.cs controls the player input for movement and sends the new position to sync it with the other player.

RandomPanelSpawns.cs randomizes panels that that can be spawned and are used to pasting the objects on to. This script also records all the data (if this player is the host) and sends it to everyone else that is not the host. If this script is called when it is not the host, it receives all the already generated data and generates the correct tiles, so every player sees the same tiles in the same order.

RaybitchSynk.cs is connected to the pointer object and sends the position of this object so the other players see where they are pointing to.

SyncPastedGameObject.cs handles the looking up what the data is for the currently required pasted object and sends the exact data, by firstly converting it into a couple of strings and when received it converts it back into usable data arrays. When the data has been received it sets the mesh component that is otherwise handled by the PasteObject.cs and sets the required values.

WandFollower.cs just follows the rotation of the camera so the wand can stay on target.

Before I had to get some feedback from some peers, the game was going to be pasting just some cubes that were different colours. But I got the feedback that I needed something that is a more complex for sinking between the networks. So, I decided to sync the mesh parameters which allowed me to paste any kind of mesh.

To achieve that I needed a way to sink arrays, however Forge Networking Remastered does not support that. I tried to re-wright how some of the networking code so I can make it supported. However, this did not work. (I have left my progress commented in the Networking scripts that I edited however it does not work).

My second way of sending it through was by using a string. I used a couple of for loops with a switch statement to create a reversable string that can be converted back into the required data. To convert it back I used string.Split and passed the special symbols I used when converting to string so I can convert it back. This all gave me a way to send an array in the form of a string on the network without needing the support from the Networking package. (Look at script SyncPastedGameObject.cs for the exact algorithm)

Artificial Intelligence

AI behaviors

The AI has 6 main behaviors. And they are (ordered from importance in the BT), wave behavior, follow behavior, stop following behavior, load the campfire with wood behavior, rest behavior, and wonder behavior.

Wave behavior consists of 9 action nodes and 3 composite nodes. It looks if the player has waved in front of the AI and if so, it stops what it is doing and waves back.

Follow behavior is a bit more complicated but it is similar to the wave behavior. It consists of 20 action nodes, 8 Composite nodes and 4 Decorator nodes. When the AI sees the player waving to follow them. It plays animations, it starts to look at them constantly and every frame its destination position is set as the looking action runs in the same time. As the AI has this tired state. If the AI is tired, but it needs to still follow the player it is going to switch to a tired walking animation.

In this branch is also the Stop following behavior which ends the AI follow behavior and, in some situations, runs a random animation to show that the AI is quite sad that its journey is over with the player.

The Gathering wood for the campfire behavior also has a tired and non-tired side to it. There are 29 action nodes that, but this is because of the tired and non-tired branches. If the tired behavior was not needed only 14 action nodes would have been needed for this behavior. There are also 12 Composite nodes to make this behavior work. The behavior makes the AI to go to a tree and punch it to get some wood. When the wood is gathered the AI goes back to the campfire and loads it with the gathered wood.

If AI is tired (the energy variable is too low) the AI will start to walk slower and have a different animation. When the AI is not required to do anything else it will go to rest at the campfire until the AI is being called somewhere by the player or the campfire is low on strength.

The Wonder behavior is very simple as it just chooses a random location to go to and the AI moves to that location.

Final Accomplishments

The BT allows a very modular way of doing AI behavior as in total there are 75 action nodes that make the current BT, but the actual number of different scripts that make up all the functionality is 20. With 20 different action node scripts I have been able to make a BT with 6 main behaviors.

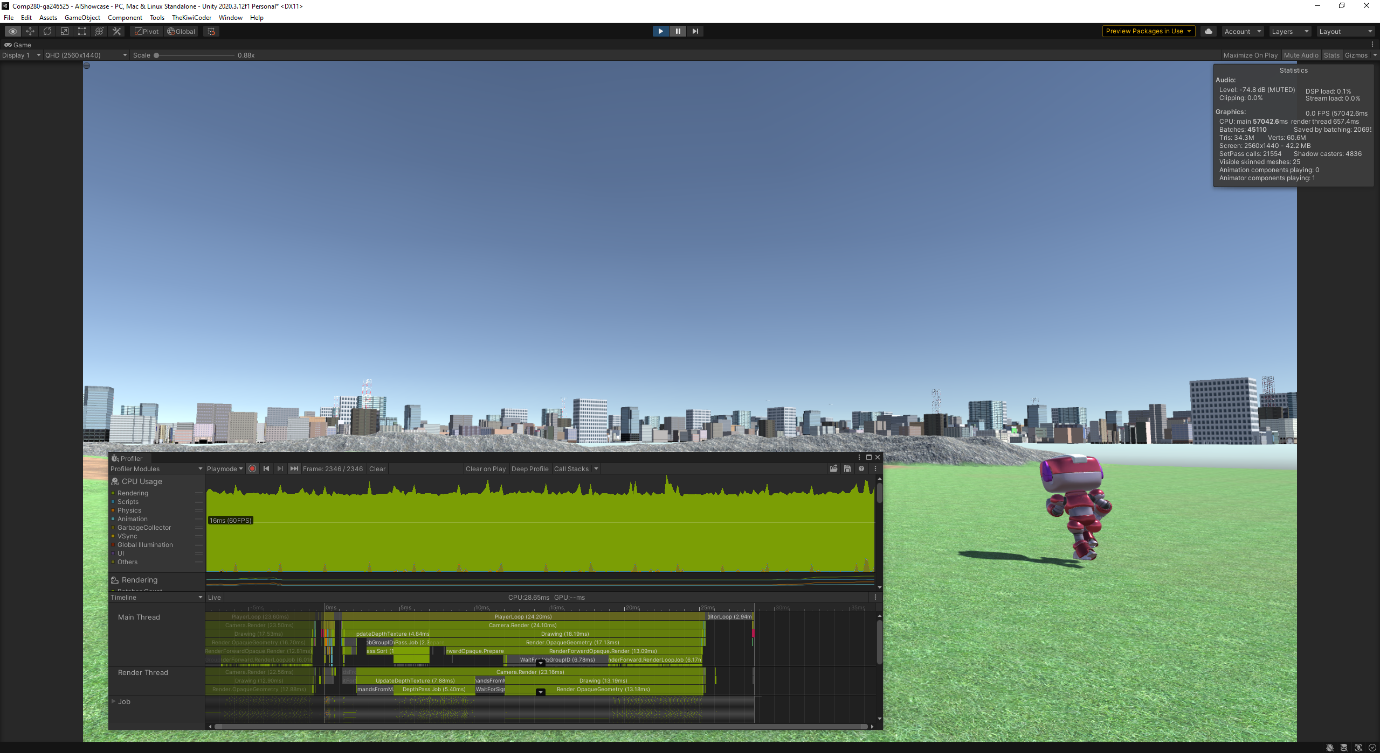
There are of course some other scripts that are not action nodes, but are connected to the campfire, AI and Player so more relevant variables are with the most obvious game object.

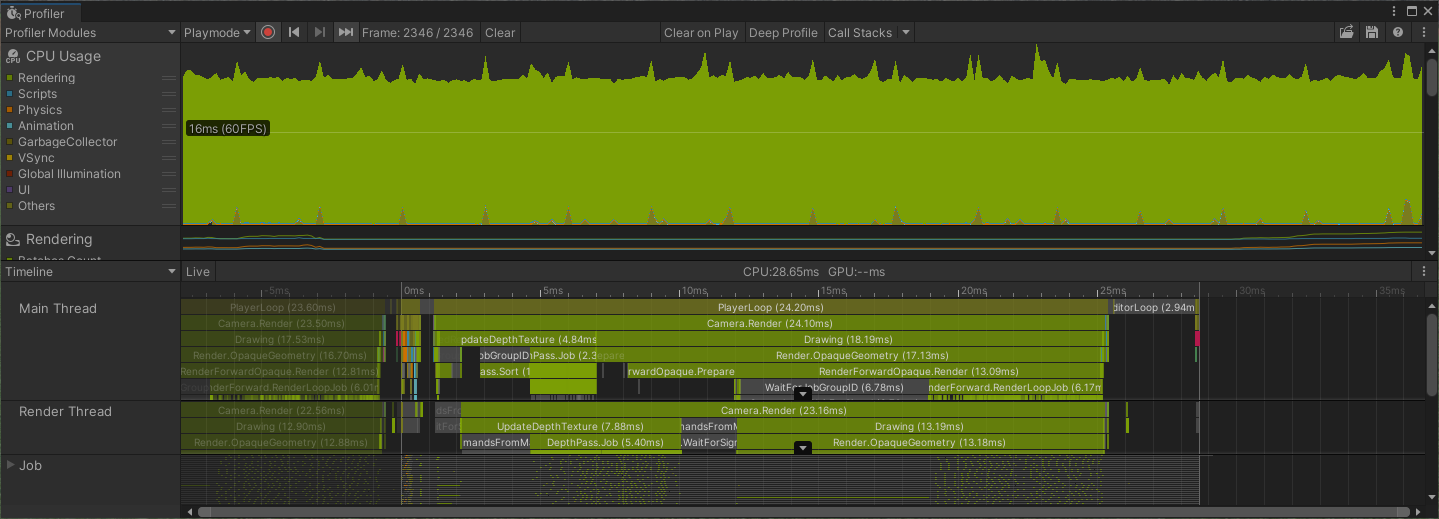
Optimization

Issues

In the AI showcase scene, there is a “City” around the allowed play area for the player for some environmental feel in the game. However, all these game objects that make the surrounding area are causing huge performance issues in the game. Before adding the surrounding “City” the game had average of 160 FPS and it was feeling very light and easy to run. The issue now is that the game is very performance intensive because of all these game objects in the distance.

Profiling

To find out what the problem is and how big of impact it has on the game the profiler will show all of this information to us.

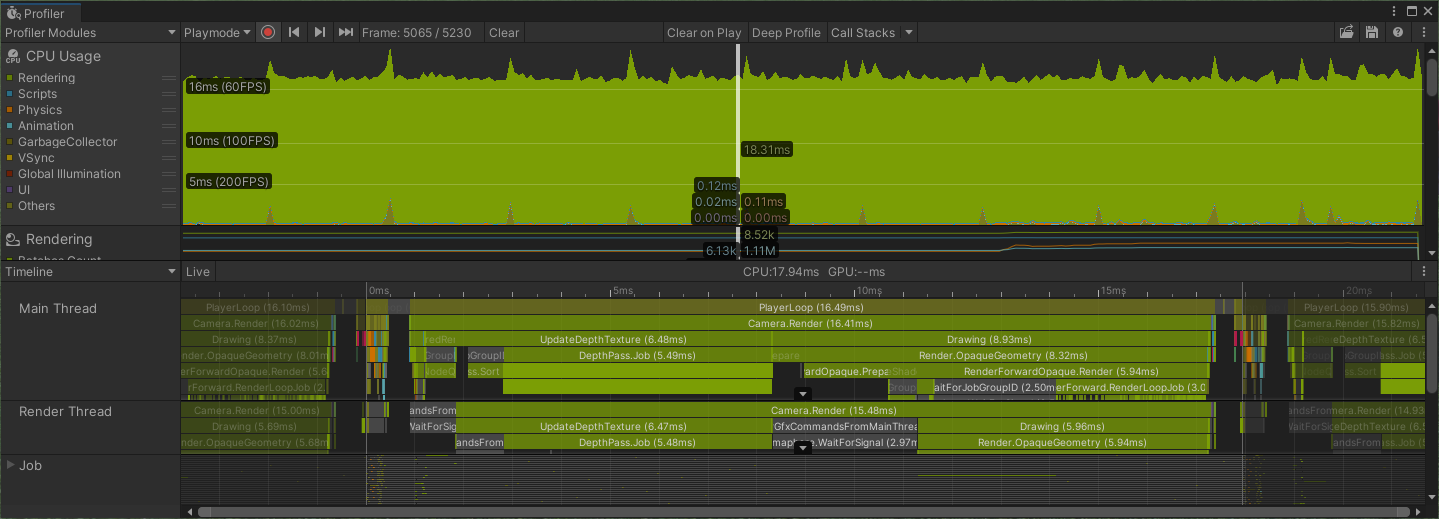


When running the game in the editor and we have the profiler open we can see that we are running way below 60FPS (probably like 40FPS). And if we inspect the Timeline>MainThread we can see that the PlayerLoop takes 24.20 ms to output every frame. A huge chunk of this 24.20 ms is due to the Camera.Render. The Camera Renderer is taking 24.10 ms of the 24.20 ms the Player Loop is taking. So it is obvious that rendering in the game is very slow and we must improve it somehow.

Fixes

A very easy and good way of fixing the awful game performance is to use the build in Unity Occlusion Culling. By setting all of the static game objects to Static>Occluder Static and Occludee Static. After Baking the Occlusion from the Occlusion window, we can see the following results from the profiler:A video game screen capture

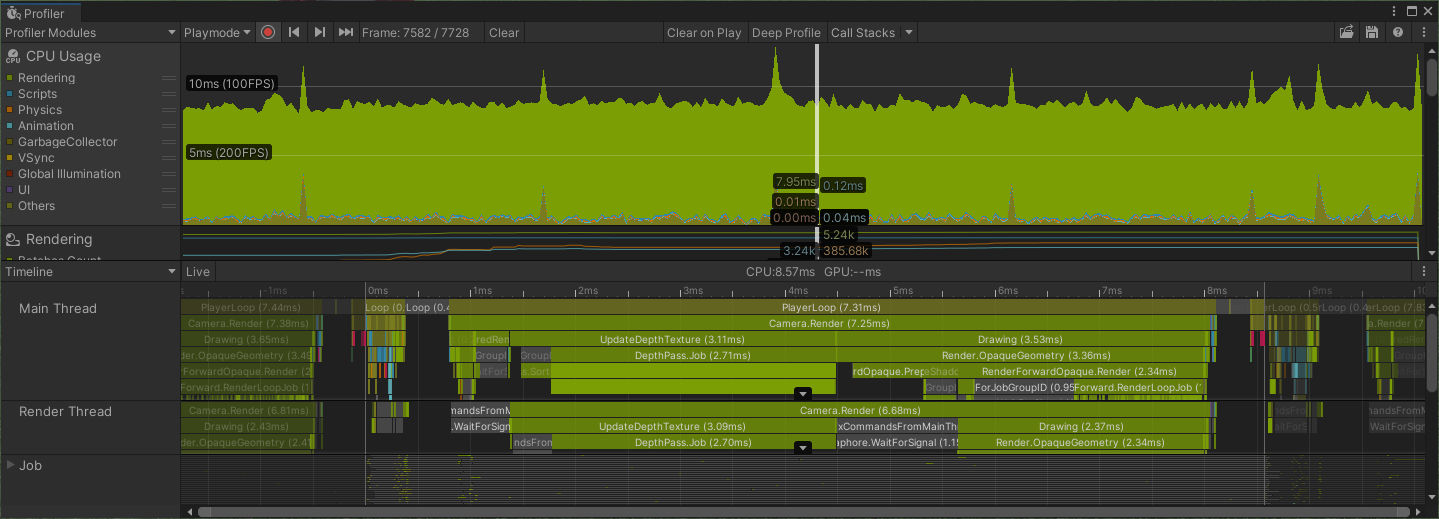
Description automatically generated with low confidence



Only by using Occlusion Culling we increased our FPS to just below 60 and reduced the Camera.Render from 24.10 ms to 16.41 ms solid 32% improvement to the game’s performance.

Additional way that we can better the performance is by reducing the Camera View Range. As The View range before the fix was 1000 it caused the camera to render things that were too far from the player which is unnecessary. By reducing the view range to 550 we see the following results:

A picture containing text, sky, grass, monitor

Description automatically generated

The game performance has once again increased from just below 60 FPS to an average of 115 FPS. And the Camera.Render time was reduced from 16.41 ms to 7.25 ms which is a 55% improvement.

After these two optimizations we have increased the FPS of the game from 45 to 115.

I also tried baking the light in the scene to gain some more performance, but it did not help the game in any meaningful way. This is not because light baking is not a good optimization method, but because the scene has only one Directional light and no other light sources, so it does not provide any performance benefit.

Human Computer Interaction

Heuristics

In the world of usability experts like Jakob Nielsen, who has a Ph.D. and is a User Advocated and principal of the Nielsen Norman Group, as well as Alita Joyce (User Experience Specialist) and Page Laubheimer (User Experience Specialist) have deeply discussed, in three separate articles, Heuristic Evaluation.

Error prevention

In this specific case as the game does not have any save states or ways to save the progress. In other words, as the game is short it does not have any saving functionality, because it is intended to be played and finished in one go. Therefore if the player closes the game without meaning to they will loose of their progress and need to start from the beginning. This is where Error prevention comes in. When we present the option to the player to quit the game we need to be sure that the player really intends to quit. Just like how it is discussed in the three articles from Jakob Nielsen, Alita Joyce and Page Laubheimer the game needs to have a “confirmation option”. If the player presses the quit button again that will mean that they have intentionally pressed the quit button.

Recognition rather than recall

Users do much better when they do not need to memorize too much information before they can use a website, any application, or a game. This is all confirmed by Jakob Nielsen and Alita Joyce (articles in bibliography). To help the users to not have to memorize any particular amount of information we can just show it all on the side presented as hints to remind what the users can do.

Help and documentation

For the users to be able to use anything it is best that this particular product has documentation or a help page to show what the users can do and/or what is the purpose of this particular product. In websites for example it is very frequent for the users to see a FAQ (Frequently Asked Questions) which can help anyone with the most common question without the need for contacting support.

Something similar can also be created for a game. An example would be a page explaining the goal of the game and simple instructions on how to achieve it.

Identifying Issues

Currently the game does not confirm if the player intended to press the quit button and instantly closes the game once they have pressed quit. This can cause a lot of problems with player losing their saves and then complaining to the developers or publishers.

Currently the game has controls for both games in the main menu. However, they are not accessible if the players are already in game. If the players forget the controls, they will not be able to quit the game and look at the controls because they will lose of their progress.

When someone plays the game and has chosen if they want to play the Multiplayer or the AI playground, absolutely no information is presented on what the goal is and how they can achieve it.

Fixes

To fix the quitting problem the game now has a window that pops up and asks the player if they really want to quit the game.

To fix the problem where player might forget what the controls are. The game now has a small box on the left that clearly shows every control that might be useful to the players.

To fix the documentation problem. The game now shows in a text format what the purpose of this game is and some information on how to play to achieve the goal.

Bibliography:

Nielsen, J., 2020. *10 Usability Heuristics for User Interface Design*. [online] Nielsen Norman Group. Available at: <https://www.nngroup.com/articles/ten-usability-heuristics/> [Accessed 1 January 2022].

Joyce, A., 2019. *10 Usability Heuristics Applied to Video Games*. [online] Nielsen Norman Group. Available at: <https://www.nngroup.com/articles/usability-heuristics-applied-video-games/> [Accessed 1 January 2022].

Laubheimer, P., 2015. *Preventing User Errors: Avoiding Unconscious Slips*. [online] Nielsen Norman Group. Available at: <https://www.nngroup.com/articles/slips/> [Accessed 1 January 2022].