Processes followed to optimize the scene and provides explanations as to why the changes were made.

First thing that we did was to remove all unnecessary rigid body and all the trampoline/trampoline movement script of all the trampolines. We added the rigidbody only on the parent of the airtrampoline. Like this, we can just add the rigidbody and the trampoline/trampoline movement script to only 1 object in the world which will increase considerably the performance of the scene. For the ground trampoline, we could also put them as “static” since they are not moving at all. This will also increase the performance if we add lightning in the scene. We also changed the trampoline movement so that the function will move the global rigidbody and not the trampoline. Since all the trampolines move the same way, this method make the process of it way easier like this.

One of the easiest correction was to clear some easy code problem like deleting all the “debug.log(“”)” in almost every script. We also deleted every function that was useless like “void start()” and “void update” when we didn’t need it. Even though it doesn’t do anything visible during the app is running, all those small things are still call and still run and affect the program to work smoothly. The only script that we kept the “void update” wasn’t even the same one, it was an other one call “fixedupdate” that work better for the performance since this will not check every frame but update every X time that we set on the properties of the unity project.

Instead of linking every trampoline to their particle’s effects, we changed that so that the prefabs have the script and the particle is already linked in the prefab. Since this will be done in the prefab, the program doesn’t need to check all the way to the hierarchy to find where the particle is but only check on the prefab hierarchy (that will save time and performance since there is a lot less item active)

For the score, instead of checking into an update function, we changed that so that the score will look to change when the score needs to change only. That will decrease the amount of checking through the info in the program by a lot and won’t check for nothing and will save performance. The script will know because we linked the text into the inspector (public options).

The biggest change was probably the ballSpawner script. Since the spawner was spawning a infinite amount of balls, it was obvious that it wasn’t a good optimization. First thing that we did was to change the way the item was spawn. Instead of creating a new object every time, we only created a list that will contain a X amount of ball (editable on the inspector). When create enough ball, we decide to always keep the same balls and just disable them for a short amount of time. When the life spend was 0, we disable it, kick it from the list. Later, a function check if he is active, if it’s the case, it will put it back to the list with the original transform so that he will automatically respawn at the origin. Like this, the program will only create enough instance that he needs and will never let the user make the game crash by himself just by waiting.

What I learned:

I learned that this is really easy to fix some optimization and that we should definitely not overlook the power of being smart on how we create the script to make sure that we are not using to much for nothing. Even though it looks simple to do optimization, it is also really easy to make this really hard when you have a huge project 100x bigger than this one. I feel like this is not just something you do at the end of the project but something we should make it a habit to do during all the project from the beginning. I also feel like optimization is a like “working smart” and in a clever way to make the logic simple and less complicated.