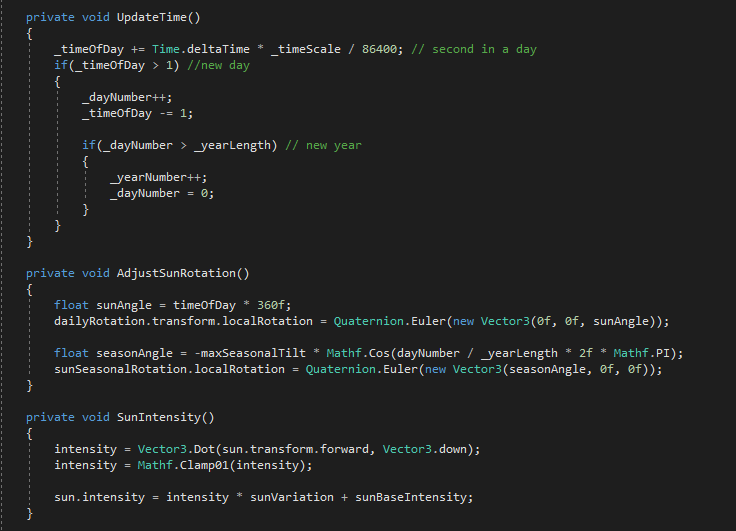
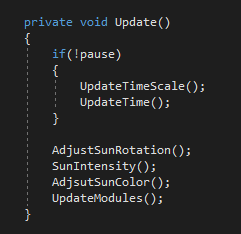
# Documentation

The game runs using a Unity Networking package called Mirror. Mirror allows the user to code both server and client coding within the same files. It does this by using commands and client RPC calls. A mirror command calls code on the host/server, whereas a mirror client RPC calls code on all the other clients. If a client calls a CMD, it will execute all proceeding code on the host machine, and following that, the host machine calls a RPC, which causes all clients to execute that code.

An example of this would be the day-night cycle that is in the game. Since it would not involve any client control, the calls are only be made on the server. Then from there, the server will make calls to the client, telling them where to update the position for the cycle. This would mean that every player would see the same, synced cycle.

To start making code into networked code, you begin with code that runs client-side. At this stage, things are not be synced up; each player will see a different thing. Next, you want to separate functionality out between what device will be running the code. For our day-night cycle, we want the host to handle all movement of the “sun” and “moon” as well as the colour changes on the screen.

To make a unity object into a mirror network object, we first must attach a network identity script to the object itself. This allows the network to keep track of it throughout the game. Once we have done that, we can change the scripts behaviour from being “MonoBehaviour” to “NetworkBehaviour”. After using the mirror package (at the top) our script, will now be able to make network calls.



Above is some of the code for our day-night cycle. Starting with the update method. We can start by using an if statement to check if the server is running, and that the platform running the code, is a server and not just a client. Ie

if (isServer && !isClientOnly)

We do this to make sure it is just the host that will run this. Next, for each of the main methods, we want to move most of the code from the methods themselves into Mirror commands. We do this by creating methods that have the Cmd prefix (ie CmdUpdateTime). On the line above the method, we need to add [Mirror.Command]. This signifies to Mirror that the follow method will be a command for it to use. We then move the code from the original method into the Mirror command, and then call the command from the original method.

Now we need to make a Mirror RPC to go with the corresponding Mirror command. These methods run on the client machines, and are called inside the Mirror command. We need a new method, with the Rpc prefix (ie RpcUpdateTime), and on the line above this method, we need to add [Mirror.ClientRpc]. However, this time we want to pass values from the command into the Rpc since we are doing all our calculations/adjustments on the host. When then just set the local values to those that we pass into the method, and now our day-night cycle should be synced.

Not all code has to be repeated in both the command and the rpc methods. There will be times where you just want code to execute on the server, or just on the client. This is simple, and all you have to do is add the code to the command or the rpc respectively. All code that is not inside a Mirror command will just run locally to that client.