19th January 2016

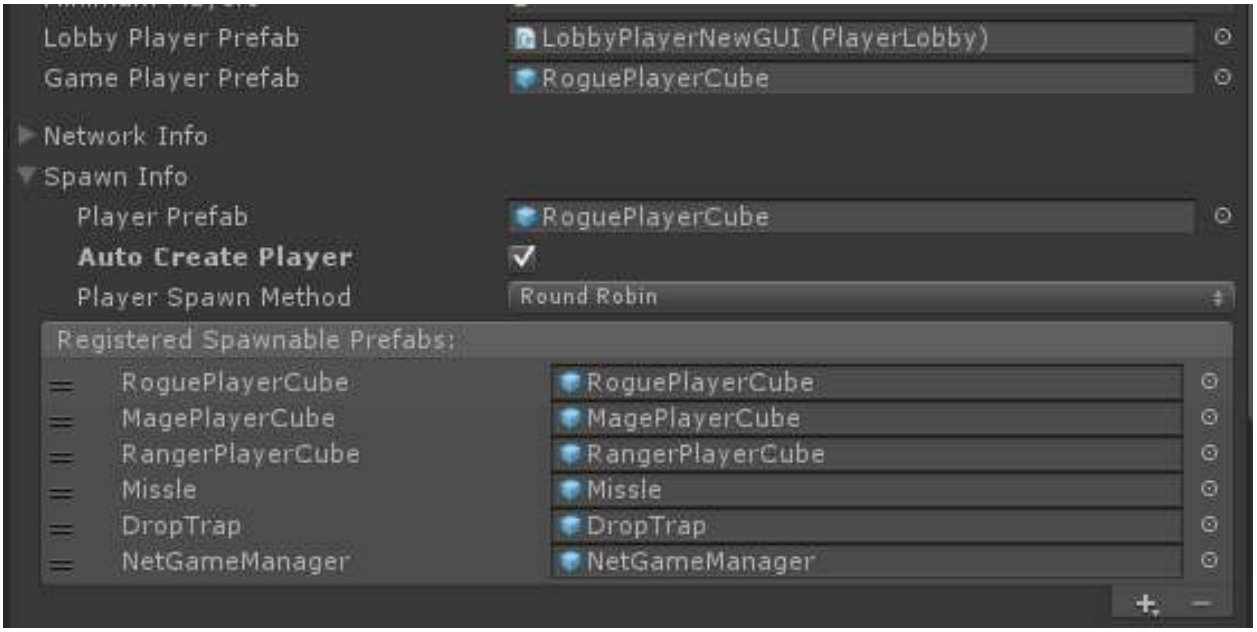
Using UNET to spawn different Player Objects

As I was going through some development on a multiplayer game I am currently working on for creating a prototype I found that I was somewhat constrained in my ability to select an object to spawn. Since I was using the UNET Network Starter project (Found here [http://forum.unity3d.com/threads/unet-sample-projects.331978/]) as my base I thought it would allow me to somewhat easily set those values, without much trouble. Though of course, I was wrong.

So in my project I have an abstract base class for all players, which all use similar controls so that I can easily create tangents. So for my project, and for this example, let's say I have three different characters/classes for a game: Mage, Ranger, and Rogue. I want to be able to pick these characters from the lobby menu, and then sequentially spawn as that character once the match has begun. Seems simple enough, right?

So one thing to keep in mind with UNET and any networking really, is that the host/server is in charge, this means that the host tells you what character you are going to be playing.

So heres an image of what the **NetManagerLobby** gives you as options:



[http://1.bp.blogspot.com/-fwIo3obALkQ/Vp7tSjX-UgI/AAAAAAAABEk/OOC7ptCYBbg/s1600/Image1.PNG]

So things to note, we have a **LobbyPlayerPrefab**, a **GamePlayerPrefab**, and under Spawn Info you'll see Player Prefab, and **Registered Spawnable Prefabs**. So let's go through these quickly

- **LobbyPlayerPrefab**: This is the prefab UNET will use when you join a lobby; it creates that object, and registers you as a connected client.
- **GamePlayerPrefab**: This is the prefab that UNET will use for instantiating players in the game once it starts.
- **Player Prefab**: This is actually the same as GamePlayerPrefab, but UNET I believe takes this value to apply to GamePlayerPrefab property.

You can find more info on all of these things here if you're curious [http://docs.unity3d.com/Manual/class-NetworkLobbyManager.html] :

So let's say I have two players, one wants to play as the Rogue, and another as the Mage. How would we do this?

First off we need some UI for selecting our class/character, I had managed to do this with a reworked version of the **ColorControl** script present, it can change the color across the network, why not an int right? So I will assume you are able to handle this on your own in one way or another, what you'll want is a way for players to select a class (I used an enum) and have just store it similar to the color.
Now that you can select your character through the UI, we need to be able to get this to work with the **NetworkManagerLobby** (**NetManagerLobby**) so that it knows which object to spawn.

I thought of 3 options of how to tackle this:

1. Change the network object Spawning code, and Find out each players select type, spawning the appropriate one for each player
2. Change the **GamePlayerPrefab** value as I change my character type, so that it's all done locally, and individually on each client
3. Spawn an Empty object, and after spawning it, tell it which player object it should be loading

So these are ranked in the order that I would prefer, as 1 gives me the most control over exactly what's happening in the spawning. 2 is super simple, but doesn't give me the control that I really want, but if it works, I can live with that. 3 though is the one I really would hate myself with if I had to do it, just because it's making me have to spawn more objects than I need, that the network manager ultimately needs to track in one way or another, not to mention it's not really elegant in any way. So let's avoid this one if we can.
So with some googling you'll find a large variety of options but none seem correct, at least for this situation. So let's go look at some API, where we will find this [http://docs.unity3d.com/Manual/UNetManager.html] which seems overall pretty useful including a nice piece of code that looks useful

```
public virtual void OnServerAddPlayer(NetworkConnection conn, short playerControllerId)
{
    var player = (GameObject)GameObject.Instantiate(playerPrefab, playerSpawnPos,
Quaternion.identity);
    NetworkServer.AddPlayerForConnection(conn, player, playerControllerId);
}
```

Let's try it out, so I had added it into the **NetManagerLobby** code, but the problem seems to be that it will spawn an object for lobby and the game, meaning we have a whole bunch of characters. Shit. Okay so on the right track but not exactly what we need, so let's look at **NetworkManagerLobby** [http://docs.unity3d.com/ScriptReference/Networking.NetworkLobbyManager.html] API and maybe we can see why.

After getting there I see a couple interesting functions:

- **OnLobbyServerCreateGamePlayer**
- **OnLobbyServerCreateLobbyPlayer**

Both of these seem to refer to those prefab objects we saw earlier, so this is good! Let go look deeper and find some examples of code, and how to spawn what we need! Oh wait… both links bring you to a page with little useful information. That's not great, maybe we can go peek at the function definition, and find out what Unity is doing… Wait… we can't do that either.

Okay so we've hit a dead end, so the only information we know is that we get a **NetworkConnection** object, and a short that's the **playerControllerId**, and that it returns a GameObject, of some kind. Let's sit on this, and try the second solution.
So the second one seems simple enough, we just need to set the **GamePlayerPrefab** to whatever prefab we have. So important thing to remember is that we need to add these objects to the Registered Spawnable Prefabs list, so why don't we use that to our advantage! So we have 3 classes and the objects, and an enum to sort them. So in enum space

- Rogue = 0

- Mage = 1
- Ranger = 2

So let's keep that for the list, that way we can just get the enum type and automatically set it to the corresponding prefab when we get a class change call from other clients, using a line like

```
gamePlayerPrefab = spawnPrefabs[type];
```

And then we're good to go
.
So let's test this out, and make sure it works and call it a day!

And run it with two clients, let's set the host as the rogue, and the client as the mage. Running, and let's make sure it worked!

It didn't… both players spawned the rogue… let's think. What might be causing this?

Hey, remember when I said this "*So one thing to keep in mind with UNET and any networking really, is that the host/server is in charge, this means that the host tells you what character you are going to be playing.*"

Well apparently I had forgotten, and realized that method number two won't work, because we need a way of telling the server what we are, we can't tell the local **NetworkManagerLobby** because unless we're the host, it won't spawn our object, only the host client will be able to do that for us.
Okay so now we know that we need to tell the server what index type we are, so let's go back quickly to what we had before and see what we can salvage.

So we have **NetworkConnection** object and a **PlayerControllerId** and looking at them I can see that there's a **connectionId**, and after some debug logs in both **OnLobbyServerCreateGamePlayer** and **OnLobbyServerCreateLobbyPlayer** functions:

```
public override GameObject OnLobbyServerCreateLobbyPlayer(NetworkConnection conn, short
playerControllerId)
{

    Debug.Log("NetworkConnection: " + conn.connectionId);

    return base.OnLobbyServerCreateLobbyPlayer(conn, playerControllerId);
}
```

We see that when 2 players join, they'll have a constant Connection ID, Great!
So now we know that a player will have the same connection Id through a session, so we need to store the ids as players join in, as well as they're class type. Not a problem a Dictionary<int, int> can handle that. So now we have something like this that we're looking at

```
public override GameObject OnLobbyServerCreateLobbyPlayer(NetworkConnection conn, short
playerControllerId)
    {
        if(!currentPlayers.ContainsKey(conn.connectionId))
            currentPlayers.Add(conn.connectionId, 0);
```

```
        return base.OnLobbyServerCreateLobbyPlayer(conn, playerControllerId);
    }
```

So since we don't need to worry about the lobby player object, we're just going to use the base function and leave it be.

Now the tricky part, we have to find a way of getting an actual override to **OnLobbyServerCreateGamePlayer** working the way we want it to. So if we look at that old function that was an example before:

```
public virtual void OnServerAddPlayer(NetworkConnection conn, short playerControllerId)
{
    var player = (GameObject)GameObject.Instantiate(playerPrefab, playerSpawnPos,
Quaternion.identity);
    NetworkServer.AddPlayerForConnection(conn, player, playerControllerId);
}
```

We can try to steal that functionality and maybe, just maybe it will work, if not we have to move to option 3, which I still really don't want to do. Okay so now we have a function that looks like this:

```
public override GameObject OnLobbyServerCreateGamePlayer(NetworkConnection conn, short
playerControllerId)
    {
        int index = currentPlayers[conn.connectionId];

        GameObject _temp = (GameObject)GameObject.Instantiate(spawnPrefabs[index],
            startPositions[conn.connectionId].position,
            Quaternion.identity);

        NetworkServer.AddPlayerForConnection(conn, _temp, playerControllerId);

        return _temp;
    }
```

You'll see a reference to startPositions [http://docs.unity3d.com/ScriptReference/Networking.NetworkManager-startPositions.html] , it's just a list of spawn points in the loaded level, and I use the connection ID as the spawn point to use in the example, you're welcome to approach that however you need.

So let's run this and see how it does…

Perfect!! It works, that's exactly what I needed, and now I can get the spawns working with multiple character types.

```
Dictionary<int, int> currentPlayers;
public override GameObject OnLobbyServerCreateLobbyPlayer(NetworkConnection conn, short
playerControllerId)
    {
```

```
        if(!currentPlayers.ContainsKey(conn.connectionId))
            currentPlayers.Add(conn.connectionId, 0);

        return base.OnLobbyServerCreateLobbyPlayer(conn, playerControllerId);
    }

    public void SetPlayerTypeLobby(NetworkConnection conn, int _type)
    {
        if (currentPlayers.ContainsKey(conn.connectionId))
            currentPlayers[conn.connectionId] = _type;
    }

    public override GameObject OnLobbyServerCreateGamePlayer(NetworkConnection conn, short
playerControllerId)
    {
        int index = currentPlayers[conn.connectionId];

        GameObject _temp = (GameObject)GameObject.Instantiate(spawnPrefabs[index],
            startPositions[conn.connectionId].position,
            Quaternion.identity);

        NetworkServer.AddPlayerForConnection(conn, _temp, playerControllerId);

        return _temp;
    }
```

So this is how the final block of code looked like after, and it seems to be working well. So in the end I was able to get the solution I wanted working, with the little documentation that was provided, and some inductive investigation. This is relatively unobtrusive to the current code in the project, so it's perfect for small and efficient changes.

Though of course this is what worked best for me and my situation, I'm sure you or others have better solutions to this issue, and if you do I'd love to read them. Thanks for enjoying this adventure with me and I Hope this helps you out.

Posted 19th January 2016 by AlexBedardReid

Labels: Network Starter, NetworkConnection, NetworkManagerLobby, NetworkServer, OnLobbyServerCreateGamePlayer, OnLobbyServerCreateLobbyPlayer, OnServerAddPlayer, playerControllerId, spawnPrefabs, UNET, Unity

View comments