

Net Positive

Building Networked Games with Unity & Photon

What we'll be doing

We're going to make a ultra-simple MMO (modestly multiplayer online) dungeon crawler. Where they can run around with their friends, attacking, enemies, each other.

We will start from empty and incomplete scripts, and incrementally build up functionality. It's OK if you are new to coding in Unity. There will be links to live google docs for each relevant script in the project for you to paste into your project. (links to these can be found [here](#))

This document is contains a mix of instructions, and explanations. Steps for you to perform start with a (!), and are **highlighted in purple**.

The existing Photon documentation is quite good, and my tutorial is loosely based on their basic tutorial:

<https://doc.photonengine.com/en/pun/current/tutorials/pun-basics-tutorial/intro>

Overview

How to run and test networked games in Unity

Connecting to the Photon server

Networked Objects (PhotonViews), and ownership

Remote Procedure Calls (RPCS)

Syncing Variables over the network

First Steps

(!) **make sure you have Unity 2017 installed**

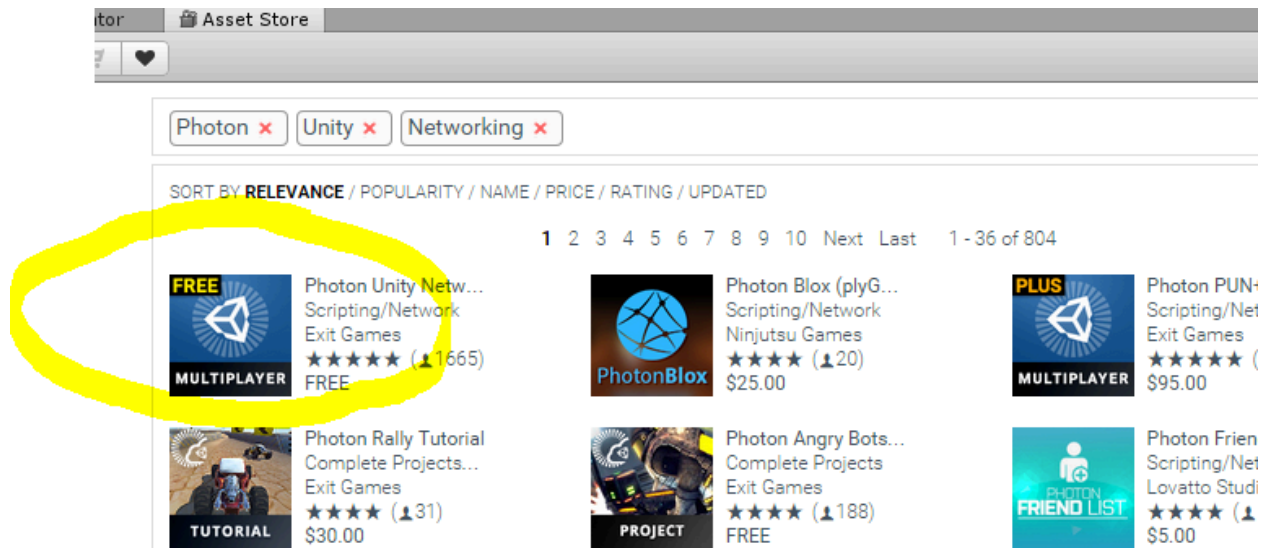
(!) [download](#) the template project

(!) unzip and open with Unity

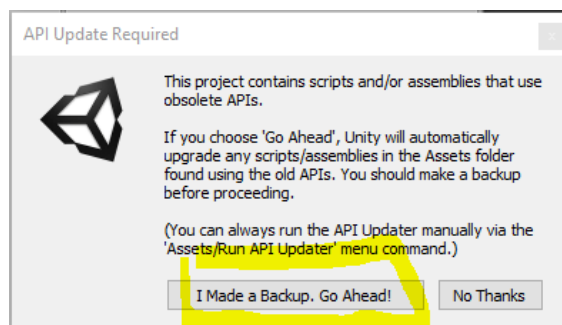
Setting Up the Project

There will be some errors, it's just because you need to install Photon.

(!) Download and import '**Photon Unity Networking**' from the Asset Store.



It's this free one



Hit this button if it appears

Create Photon Account

(!) Register for a Photon account at

<https://www.photonengine.com/en/Account/SignIn>

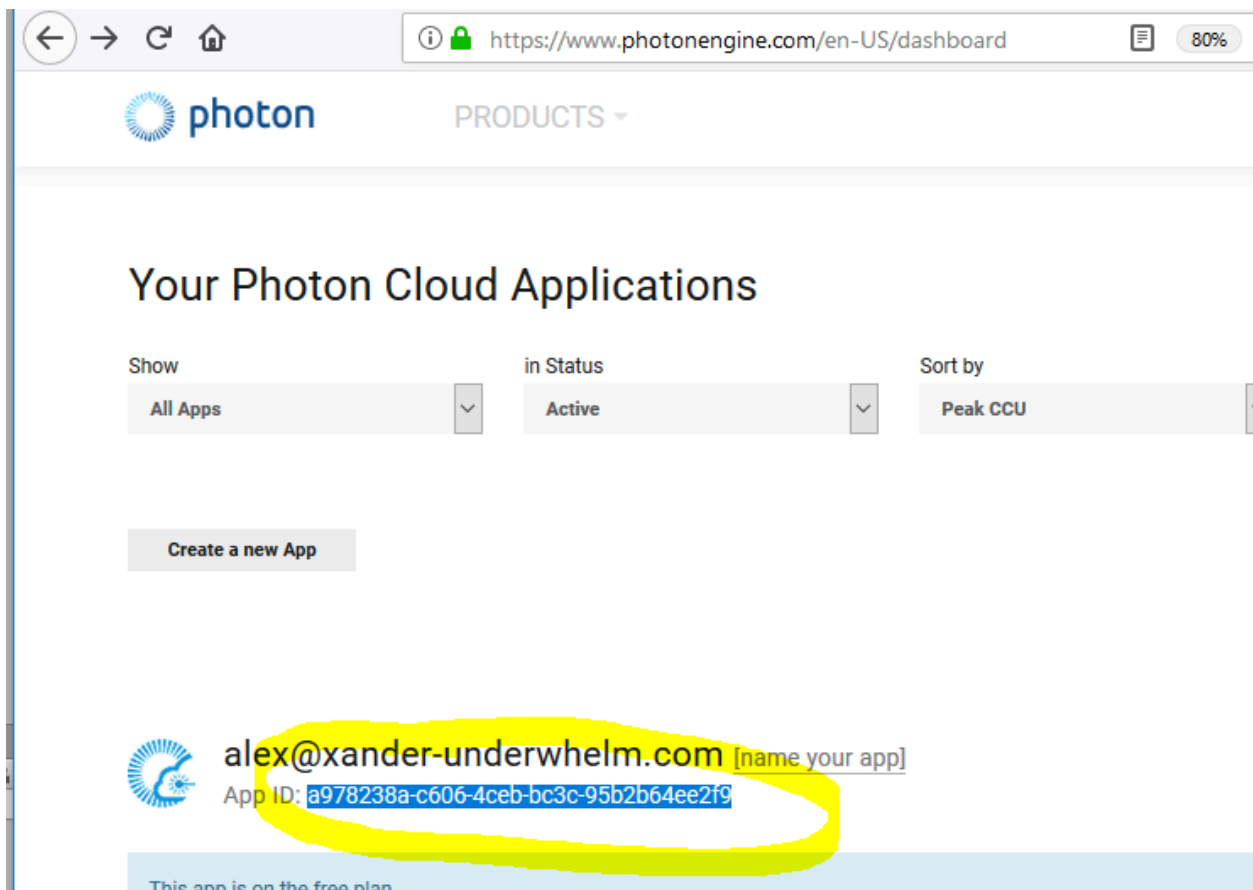
Link Project with your Photon Account

(!) Sign in to your photon account

<https://www.photonengine.com/en-US/Photon>

It should automatically take you to your dashboard

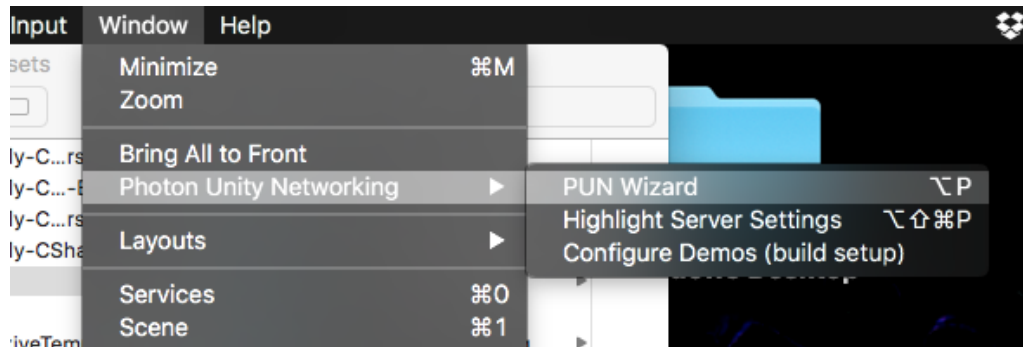
(!) Copy your **'App ID'** from the dashboard...



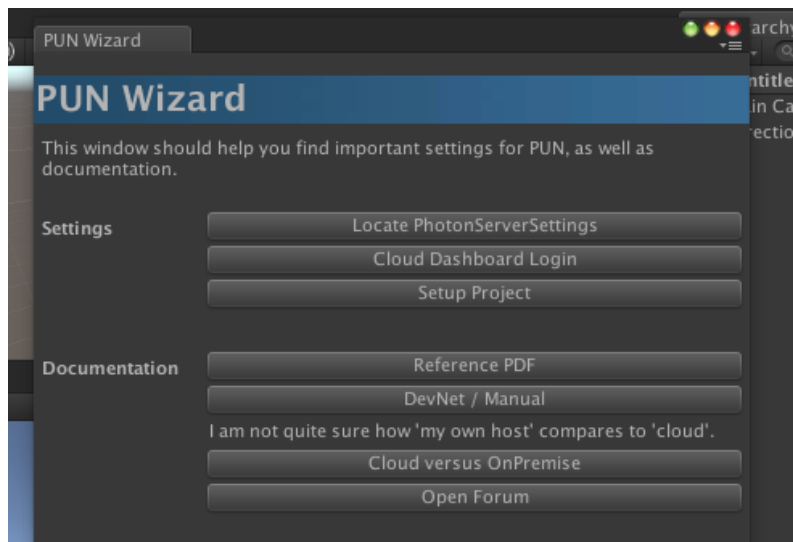
The screenshot shows the Photon Cloud Applications dashboard. At the top, there is a navigation bar with the Photon logo and a 'PRODUCTS' dropdown menu. Below this, the main heading is 'Your Photon Cloud Applications'. There are three filter sections: 'Show' with a dropdown set to 'All Apps', 'in Status' with a dropdown set to 'Active', and 'Sort by' with a dropdown set to 'Peak CCU'. A 'Create a new App' button is located below the filters. The main content area displays a list of applications. The first application is 'alex@xander-underwhelm.com' with an App ID of 'a978238a-c606-4ceb-bc3c-95b2b64ee2f9'. The App ID is circled in yellow. A blue banner at the bottom indicates 'This app is on the free plan.'

Here's the Dashboard, with the App ID circled

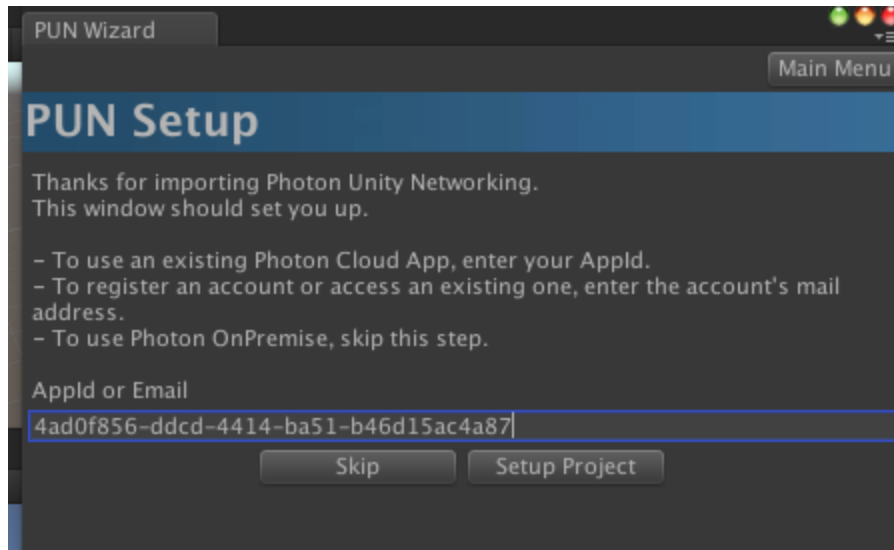
(!) Go to **Window->Photon Unity Networking->PUN Wizard**



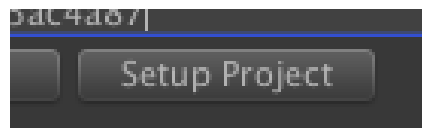
(!) Hit the **'Setup Project'** button



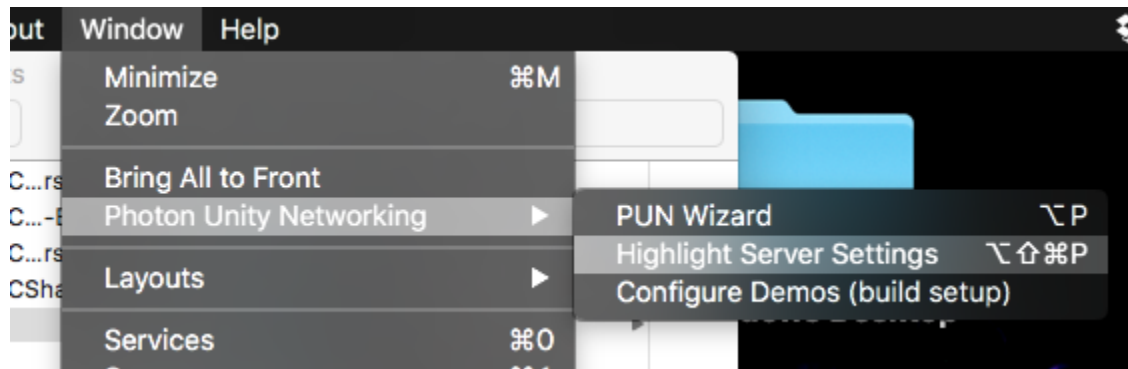
(!) Paste in your app-ID



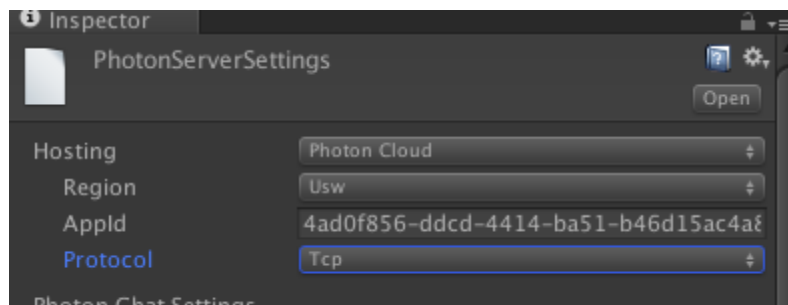
(!) Hit 'Setup Project'



(!) Go to **Window->Photon Unity Networking->Highlight Server Settings**

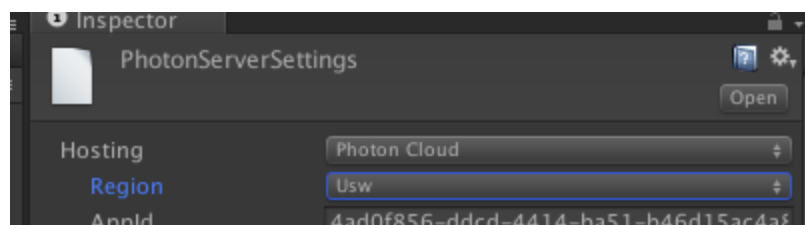


(!) **IMPORTANT!** In the Inspector, set '**Protocol**' to '**Tcp**'



It won't work if you miss this step!

(!) Also change **Region** to '**Usw**'.

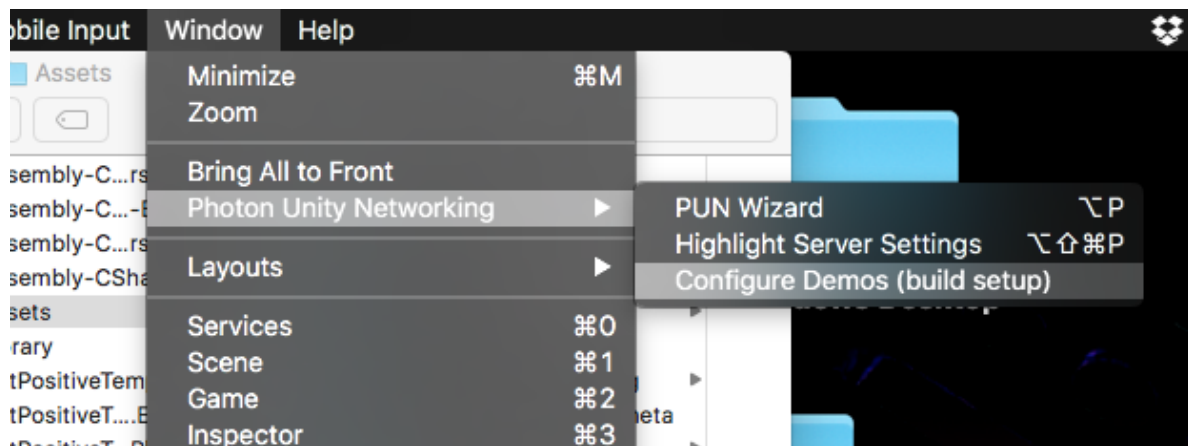


Go through a Photon server on the west coast instead of Europe

Running Networked Games

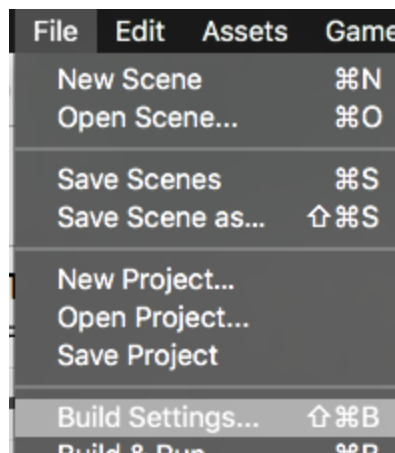
We're building a multiplayer game, so we need to run multiple copies of our game. Typically, we'll run one instance of the game in the editor for 1 player, while simultaneously running a project build (Exe etc..) for each additional player. Let's give it a try, using Photon's builtin demos.

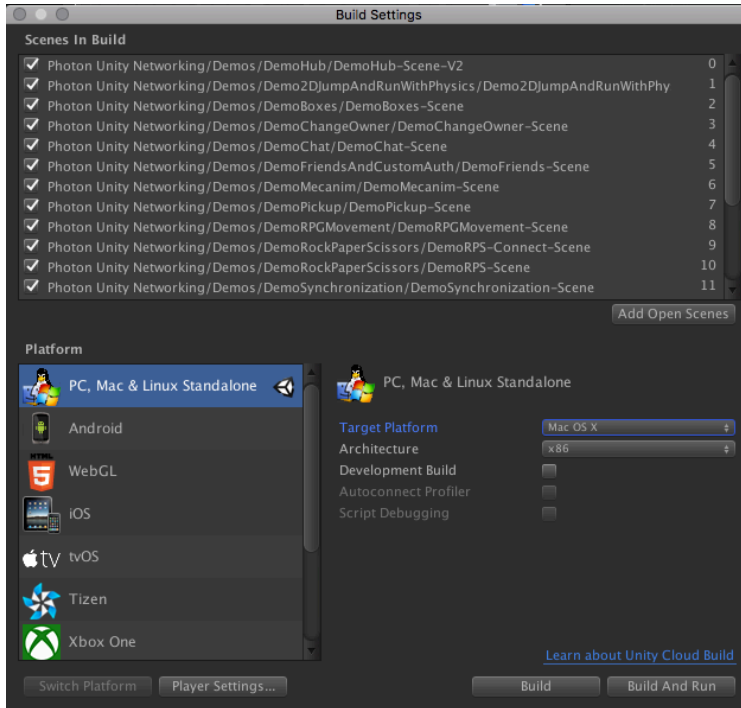
(!) Go to **Window->Photon Unity Networking->Configure Demos (build setup)**



Now we're ready to try Photon's built in demos, and start developing for ourselves

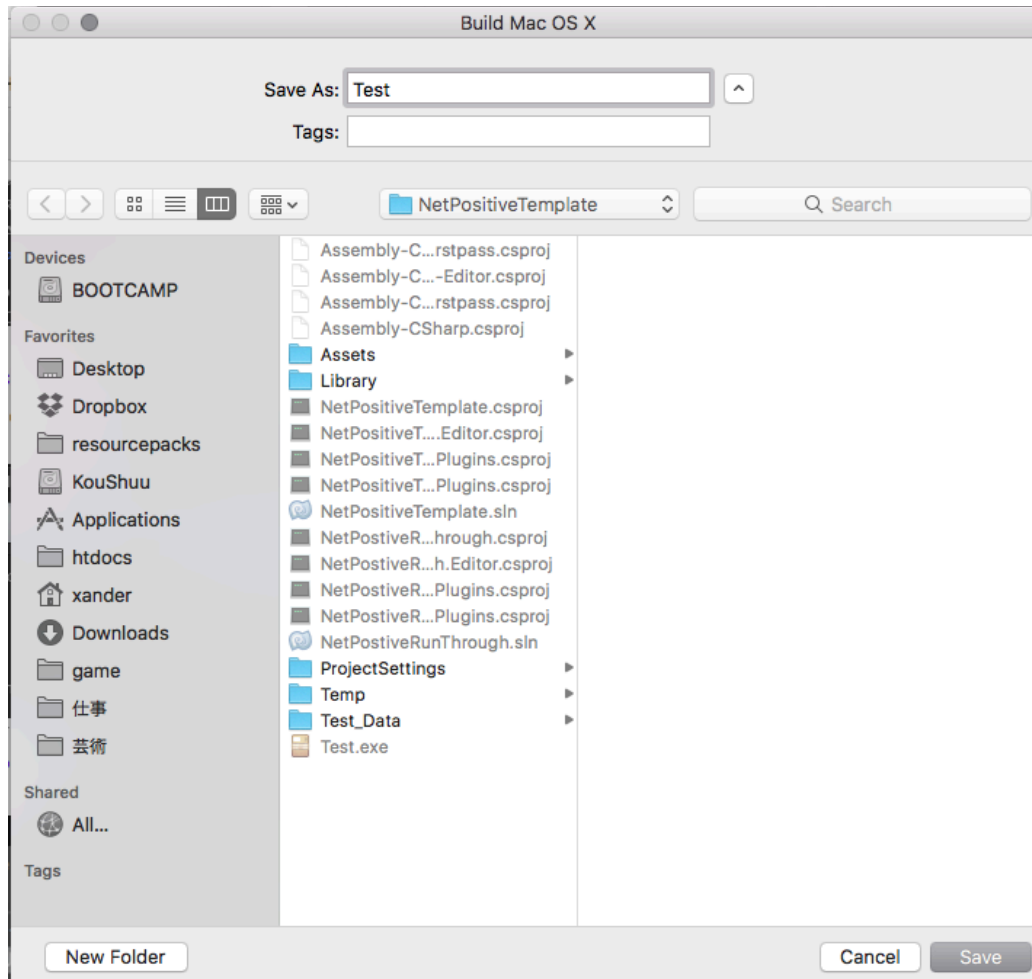
(!) Go to **'File->Build Settings...'**





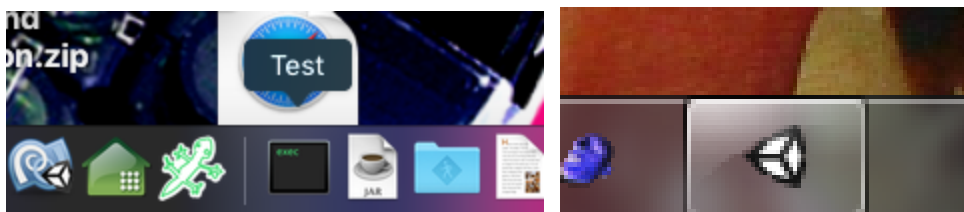
(it should look like this, with a bunch of scenes added)

(!) Build a Mac, or PC version Giving it the name “test”



I'm working on Mac here, so I'm making a Mac build.

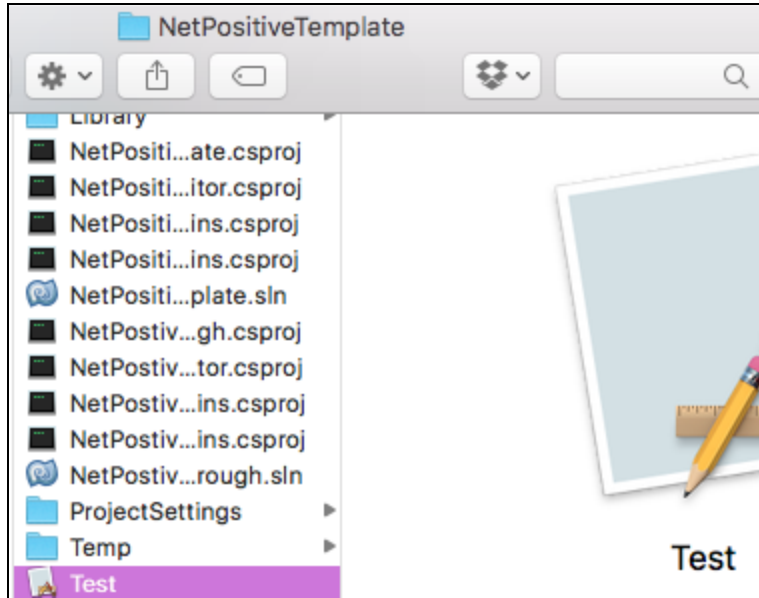
It's also a good idea to put the EXE in your taskbar (Windows), or on the dock (Mac). That way you can quickly start a second copy of the game.



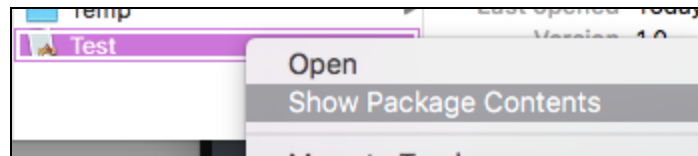
Here's the build on the OSX dock, and the Windows taskbar respectively

Important note for Mac Users:

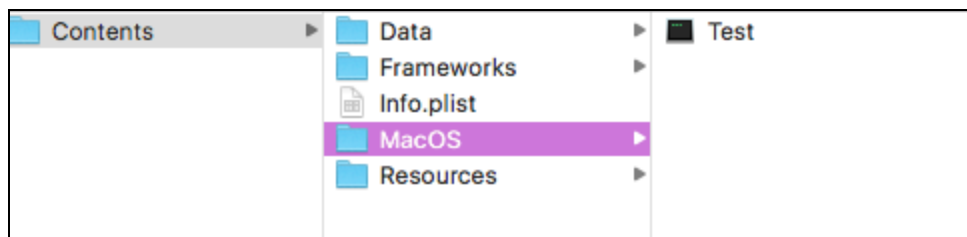
If you want to run multiple copies of your built game (to test more than 2 players), you need to go into the build's package contents below and use a different file:



Once your build is finished



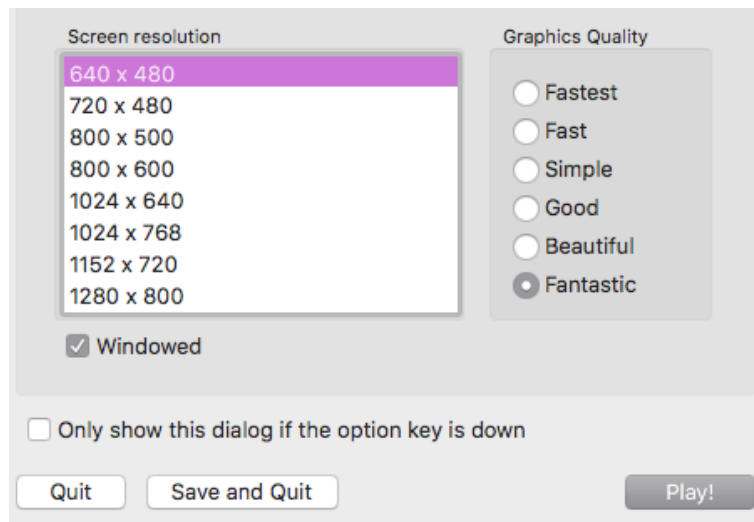
2-finger click on the build and choose 'Show Package Contents'



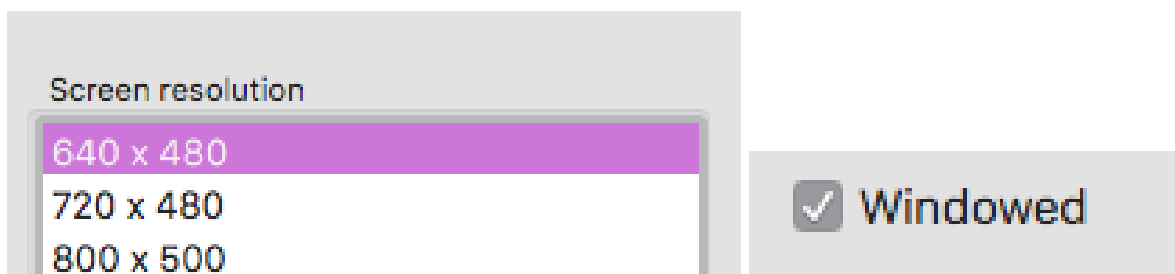
*Make an Alias of the file with the Black Icon in **Contents/MacOS**, or put it on the right side of your dock:*



(!) Once it's finished building, run the built game.

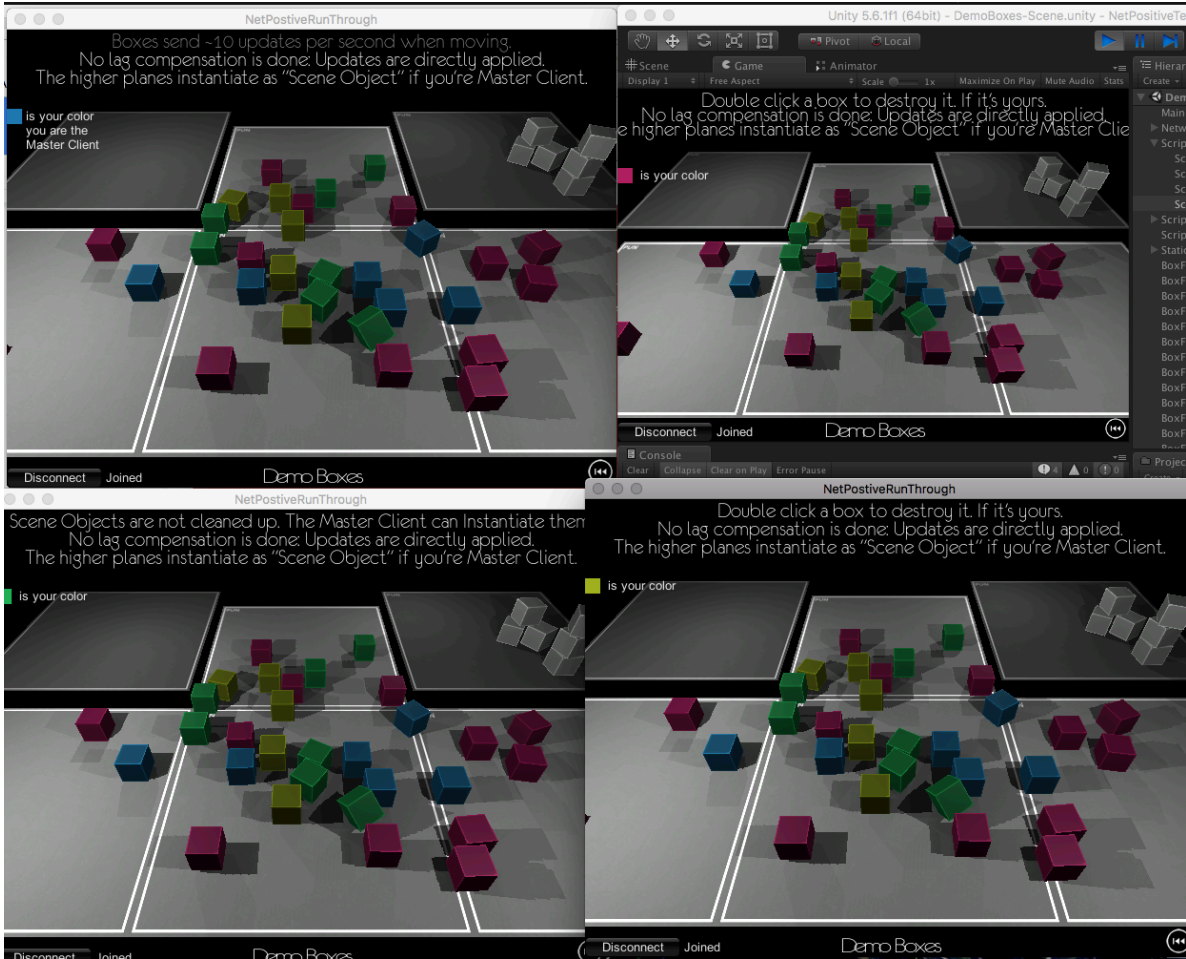


(!) In the resolution dialog, check “**windowed**”, and choose a small resolution. (like, 640x480)

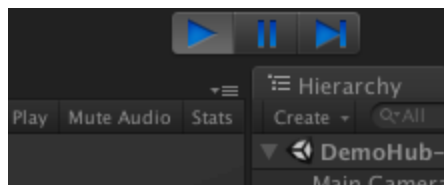


DON'T run the game full screen, or at a high resolution. We must run one copy of the game for each player to test the networking (one copy of the game can be the editor), and we really need to be able to see all the players' game screens.

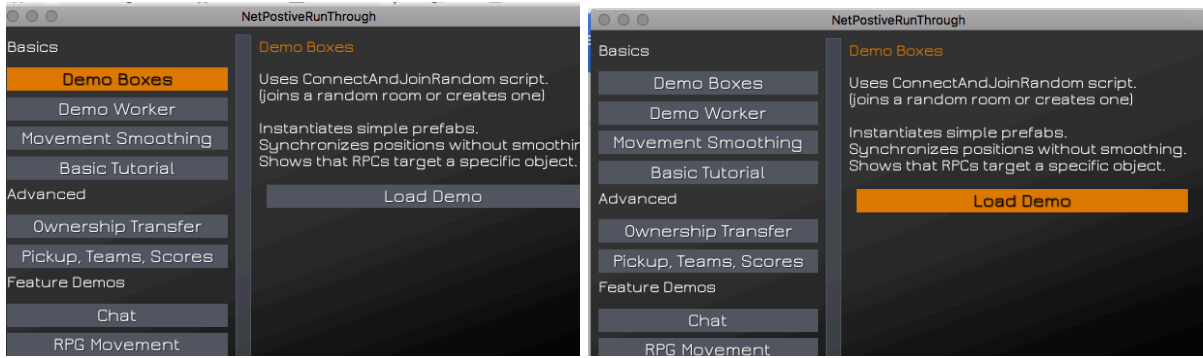
E.g. Below, we're running 3 copies of the demo, and 1 copy in the editor



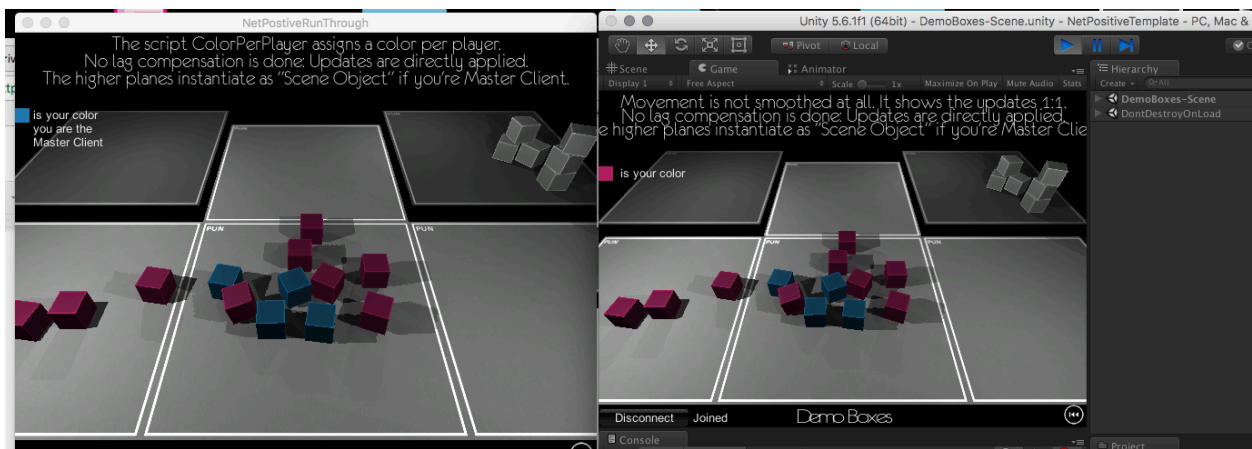
(!) Once the build is running, also run the scene in the editor.



(!) Start, the 'Demo Boxes' in both the build, and the editor



Choose demo Boxes, then Load Demo in both the build, and the editor

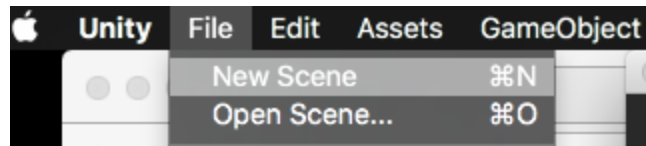


You should be able to click and see boxes appear in both windows

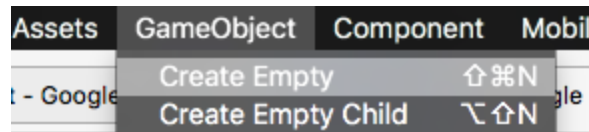
Making a new game, from scratch

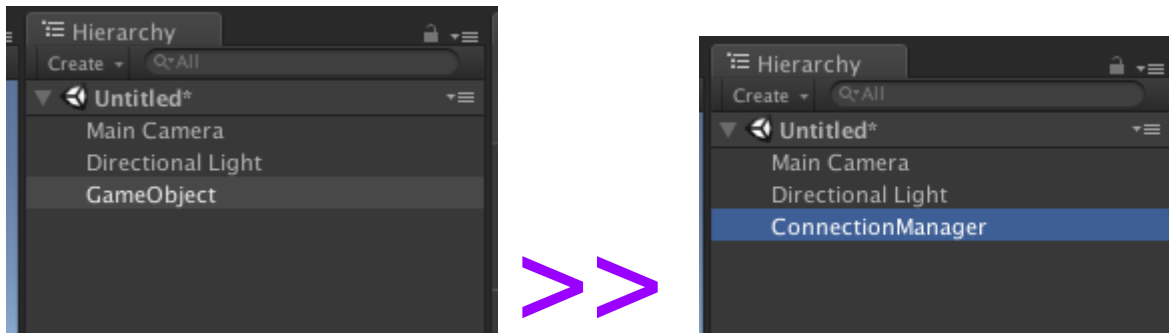
We will be making a super simple action game. Up to 20 players will automatically connect to single networked level. move a character around, be able to swing a sword, to kill an enemy (and optionally other players), and collect different wearable masks.

(!) Create a new scene

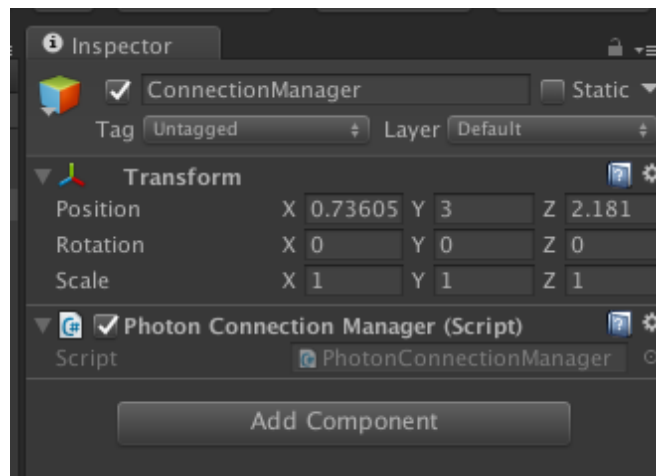


(!) Create an empty gameobject, and name it **ConnectionManager**

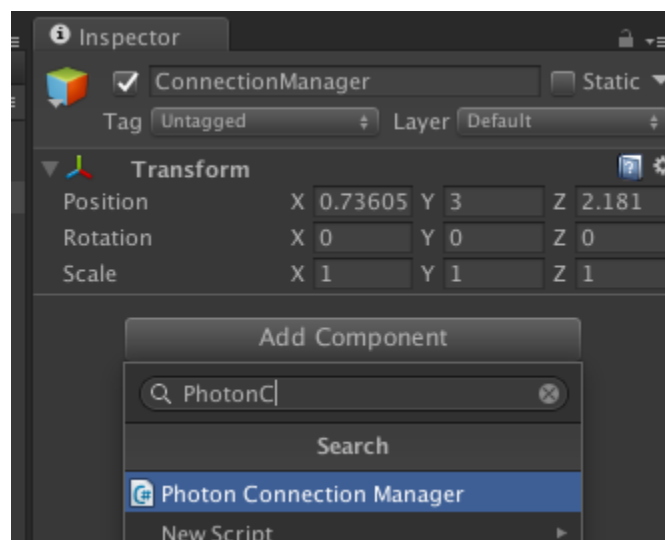




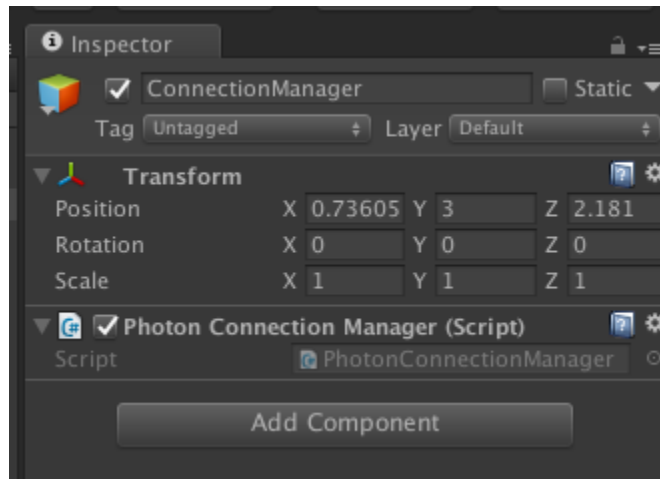
(!) Add a [ConnectionManagerIncomplete](#) script to the *ConnectionManager* object



...

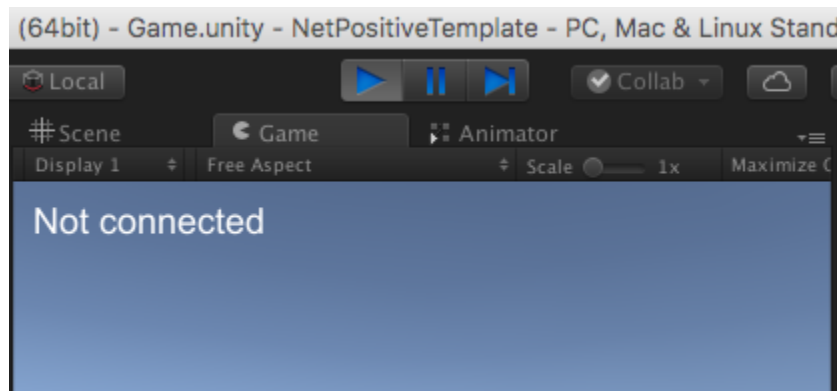


...



...

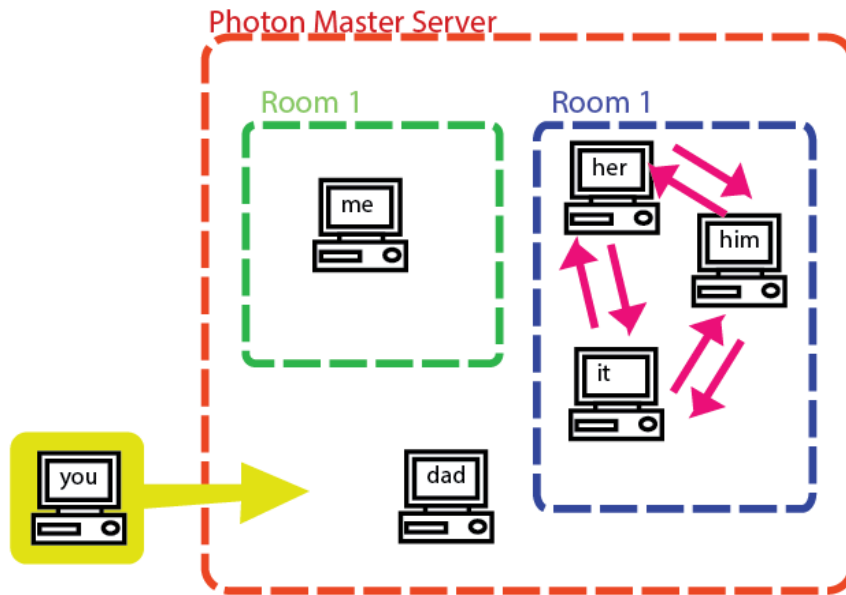
(!) Run the game in the editor



I gave you an incomplete script! we'll fix it below.

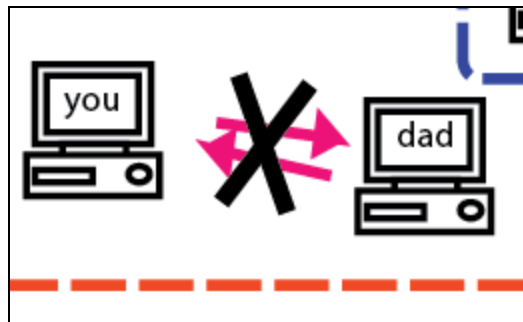
Connecting to Photon

To participate in a photon game, first must do 2 connection steps. a new player must first connect to the Photon *MASTER SERVER*.

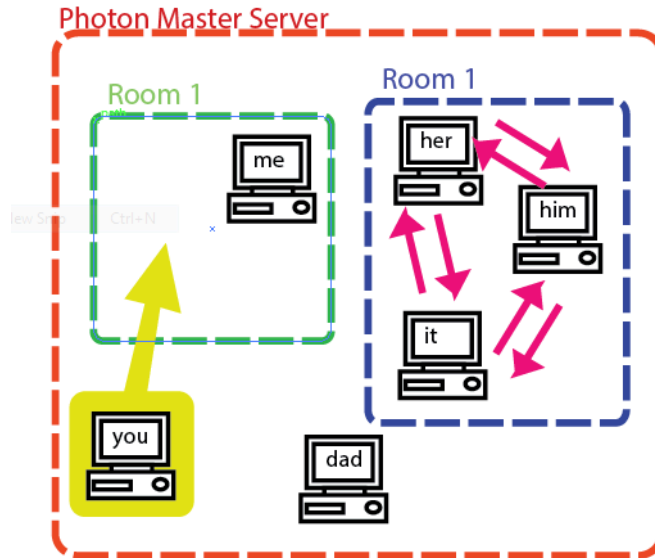


1st, you must connect to the Photon Master Server

Once connected to the master server, a player must join a *ROOM*.
 In our project, there will just be 1 room that all players join.

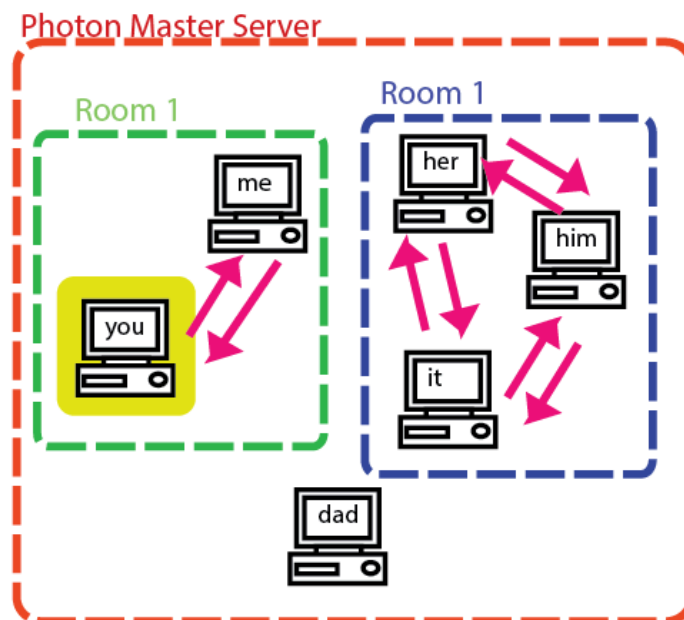


Can't talk to dad unless we're both in the same room (Both you and dad are not in a room yet!)



Once connect to the master server, if you want to actually communicate/play a game with other players, you must enter a room

ROOMS are like individual matches of a game, and players can only send and receive messages to other players in the same ROOM**. Player can freely create, join, and leave rooms. Photon also provides functionality to restrict who can enter a room, and when they can enter, or to find a random available room to join. (Think, random matchmaking). In an MMO, there might be individual rooms individual dungeons or towns.



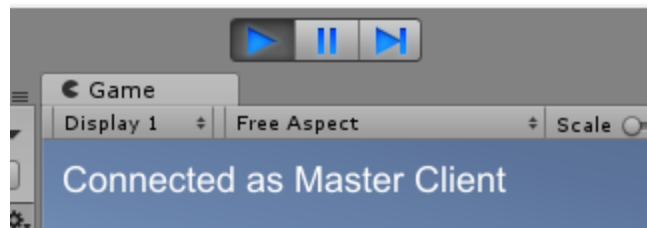
NOTE: Photon's own tutorial has more info about auto-joining a free room & matchmaking:
<https://doc.photonengine.com/en/pun/current/tutorials/pun-basics-tutorial/intro>

(!) **CODE-ALONG** : Update your script to match [PhotonConnectionManagerIncomplete](#)

*Warning! If you're skipping ahead, the above won't be ready yet.
As a hint, you need to connect to the master server, then join/create a room*

Here's more information about callbacks like `OnConnectedToMaster()`, and `OnJoinRoom()`
https://doc-api.photonengine.com/en/pun/current/interface_i_pun_callbacks.html

(!) Once the above code-along is done, Run the scene, and verify that the game view shows you are **'Connected as Master Client'**

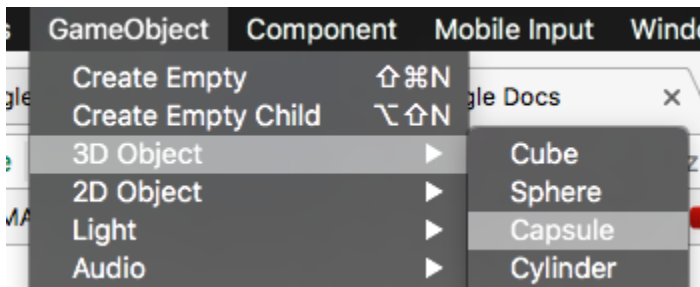


You should see this,

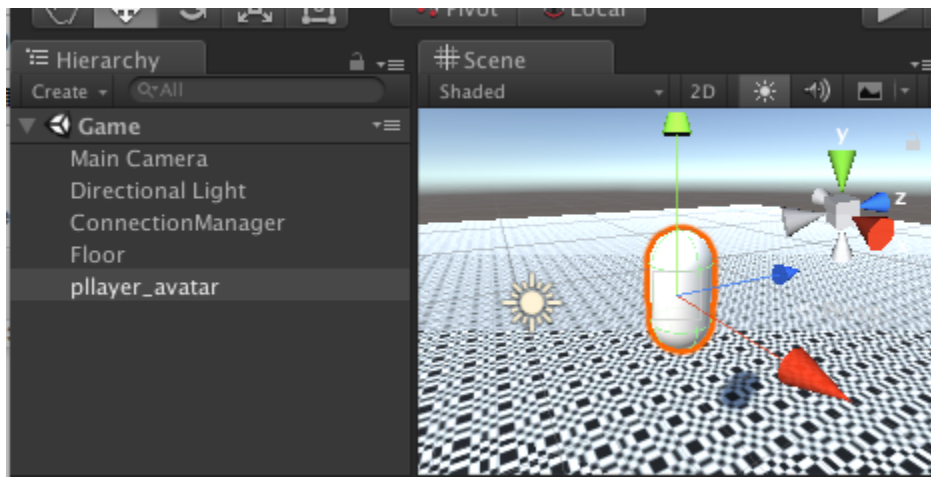
Creating a networked object

To start we'll create a networked capsules whose positions and rotation will be synced across the network. We'll eventually turn this object into a prefab for player's avatar.

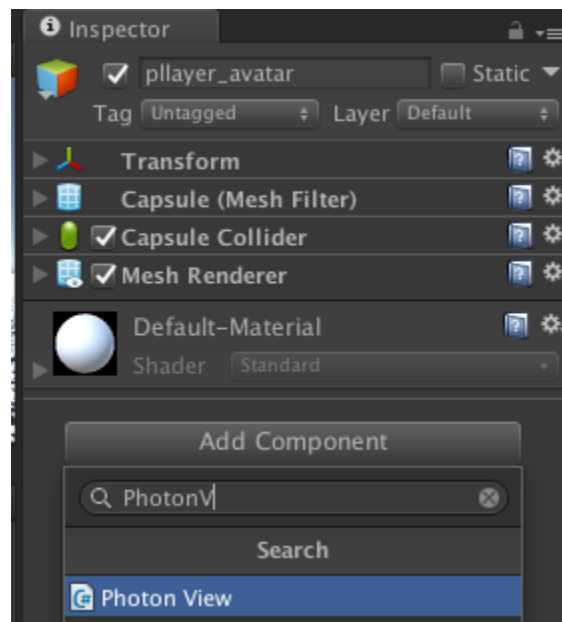
(!) Create a capsule, (it will eventually be the player's avatar)



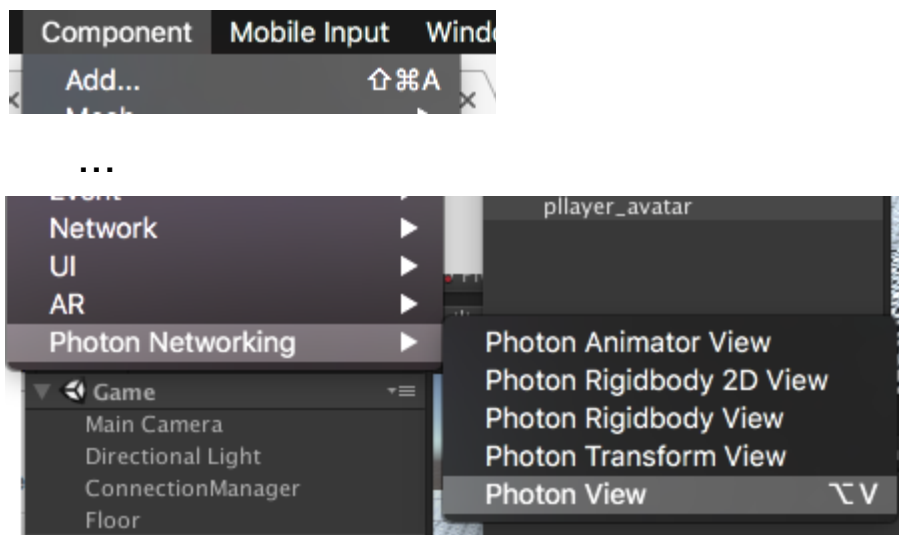
(!) Name it “player_avatar”



(!) Add a **PhotonView** component to “player_avatar”



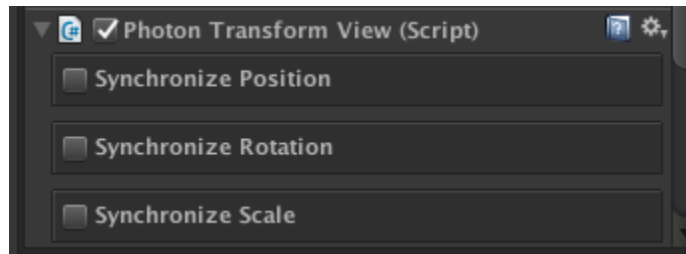
You can use the **'Add Component' button** in the inspector...
OR



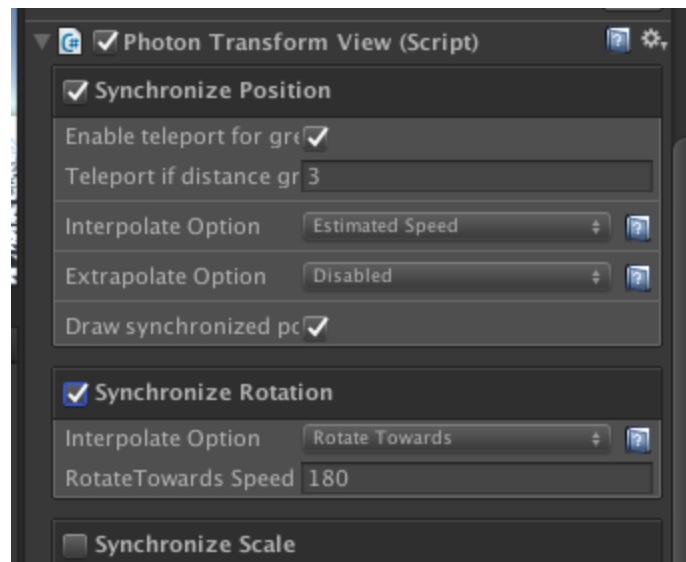
Use the Menu **'Component->Photon Networking->Photon View'**

(!) Also add a **PhotonTransformView** component

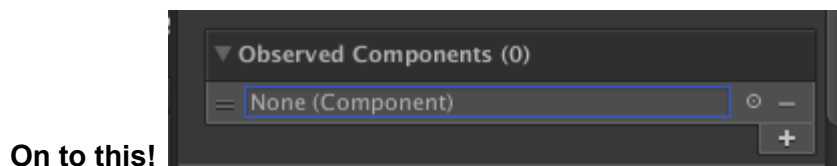
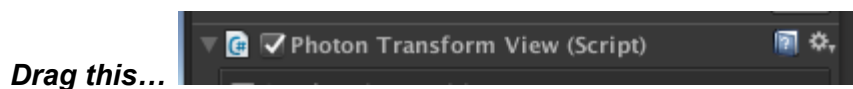
You can use the same menu, or inspector button

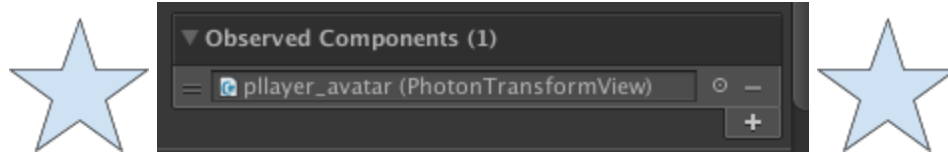


(!) Check '*Synchronize Position*' & '*Synchronize Rotation*' on the *PhotonTransformView*



(!) Drag the *PhotonTransformView* component onto the *Observed Components* slot on the *PhotonView*.

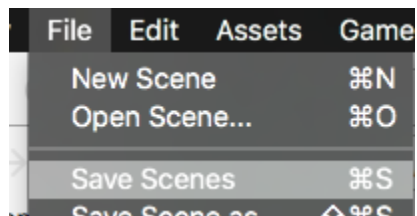




Should look like this once dragged

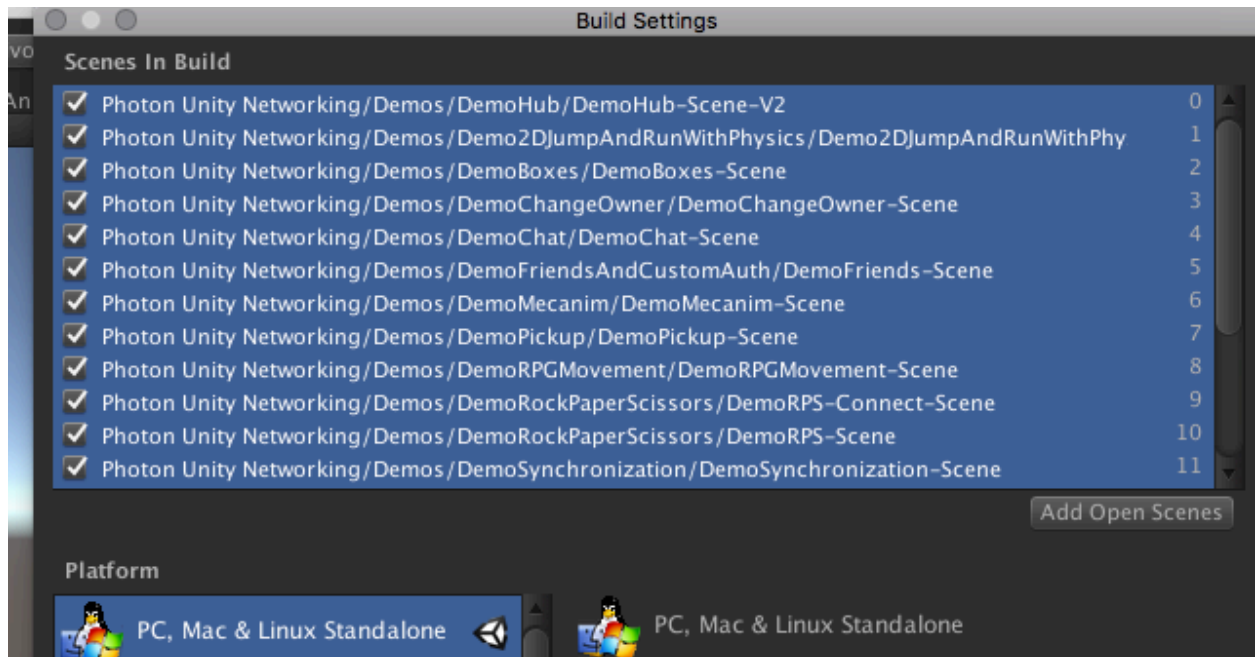
Moving a networked Object

(!) Save scene and call **“Game”**.



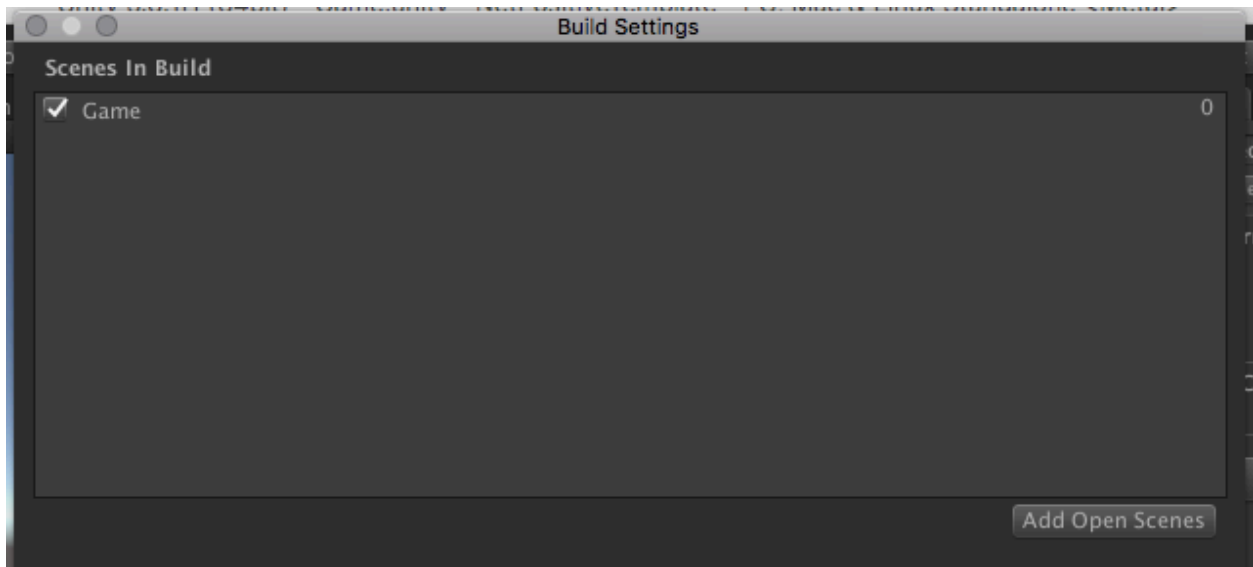
(!) Open **File->Build Settings...**

(!) Select and delete all the scenes in build



Select and delete these

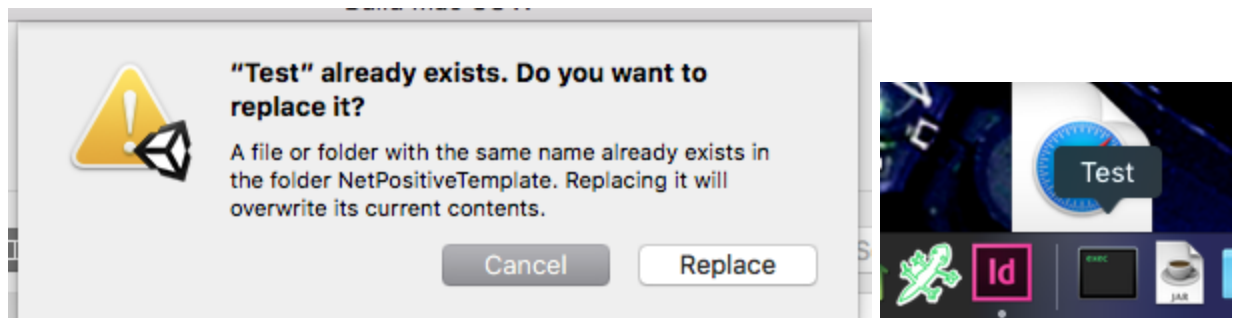
(!) Click the 'Add Open Scenes' button to add "Game" scene to the build.



You should now just have 1 scene in the build

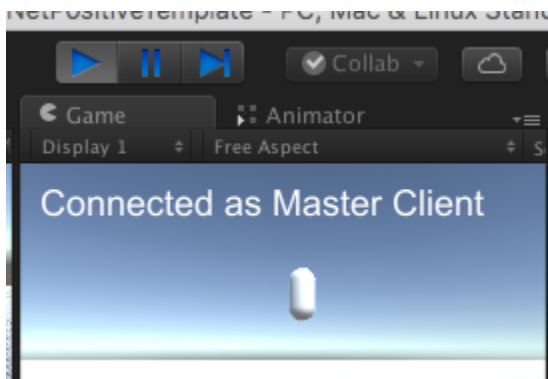
(!) Build your game. again

Don't change the name or location in the file dialog! This way, you can just use your taskbar/dock shortcut again.



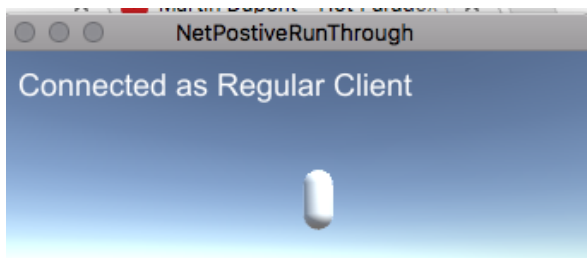
Just replace the old build! It will save you time

(!) **First** Run the game in the **EDITOR**.



Build should say, connected as Master Client

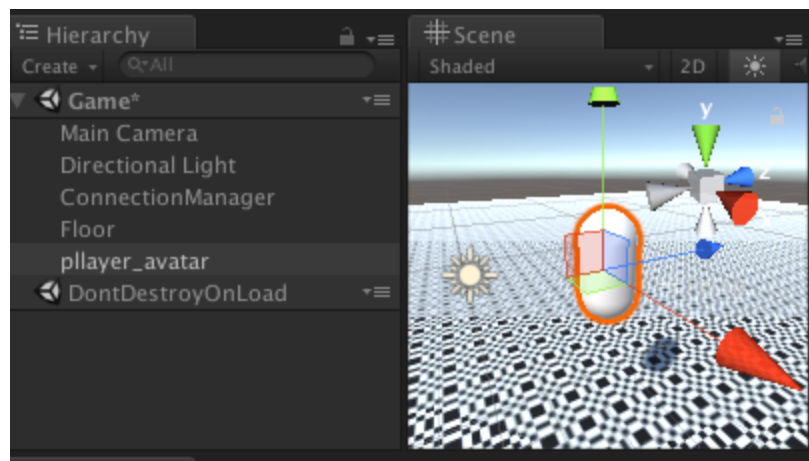
(!) **Second**, run the build you just made.



Build should say, connected as Regular Client

(!) Try moving "*player_avatar*" in the scene view of the editor with the move tool.

You should see it move on the build window too when you do!

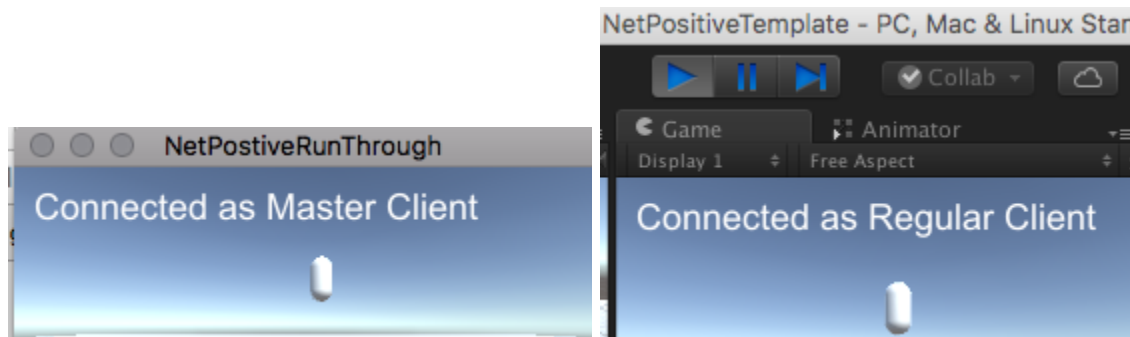


Try moving 'player_avatar' in the scene view

Now the other way around

(!) Try running the build first, THEN the editor. & try moving the player in the Editor this way.

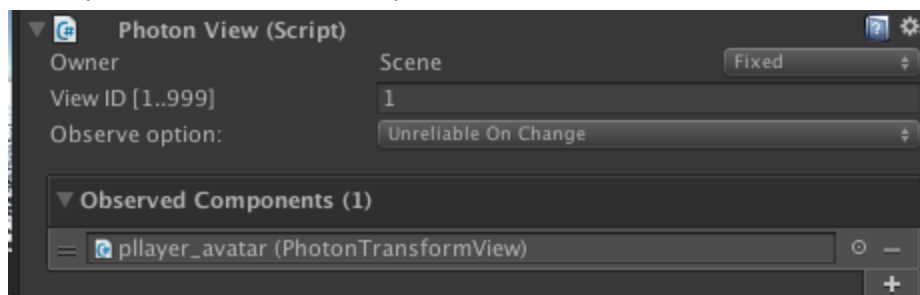
.....It won't work!



Build should be master, editor should be regular

About the Photon View

A **Photon View component** designates a game object whose state needs be synchronized over the network. Unfortunately, it doesn't just magically sync everything about an object, synced information must be contained in special scripts. **PhotonTransformView** is one such script that does a good job syncing position/rotation/scale over the network. Later we will write our own script to sync our own custom player state over the network.

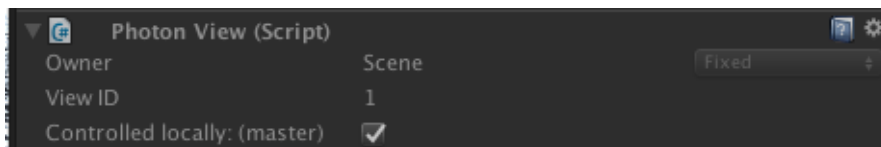


Why could we sometimes move the object, and other times not?

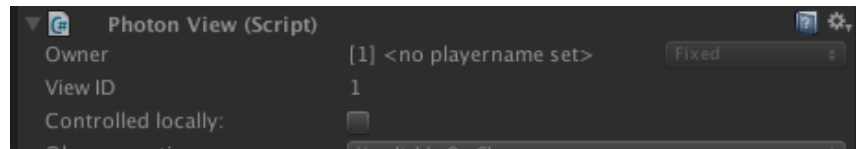
Remember our pong discussion? In networked games, game objects, (like this player) have an **owner**. I.E. which of the player owns, and is responsible for an object. The player that *owns* the object is the authority on its various state (like, position, rotation, etc...) and all other players are updating their *local* copy of the object to match the owner's version of the object.



Objects in the scene, as you would expect are given to the first player who joins the room. This first player is called the **master client**. In our “broken” case, the build-game **owns** the player-capsule, and the editor-game is constantly updating the capsules position to reflect the build’s version (The Photon Transform View is the culprit!, helped out by the Photon View). Our attempts to move this object that doesn’t belong to us is being quickly overwritten by this background process that makes its position match the own. If you pay attention to the **Photon View** component in both situations, you’ll see that a checkbox will appear “Controlled Locally” signifying whether or not the editor is the owner of a particular object.

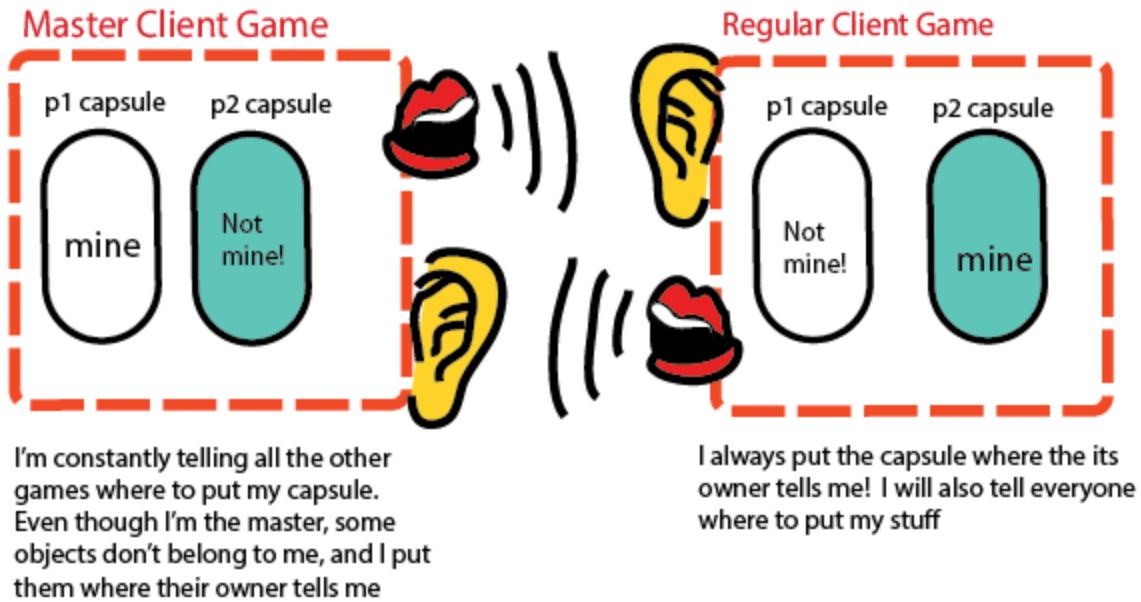


If the editor owns the object...



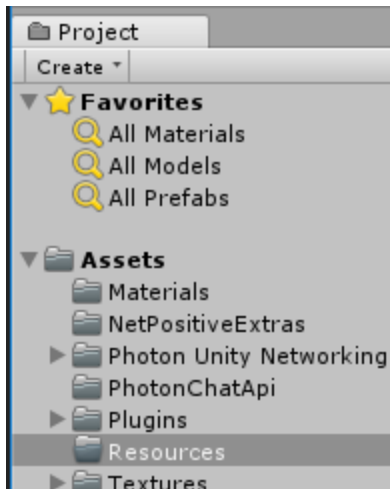
If the editor DOESN'T own the object...

What we really want, is create an avatar for *each* connected player, and have each player be the owner of their avatar.



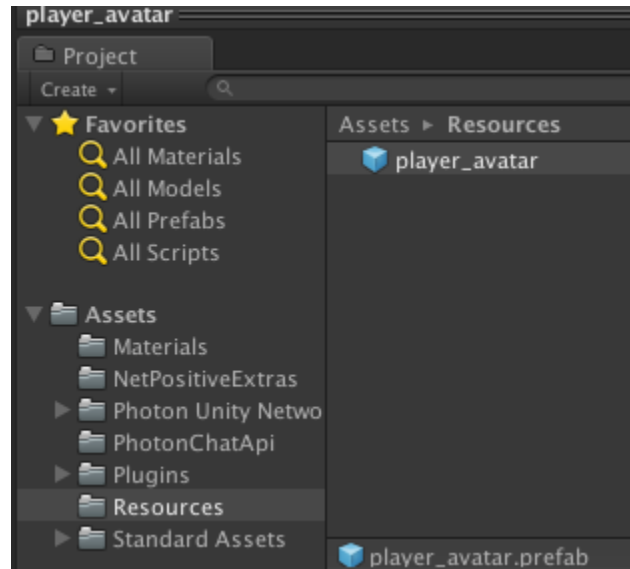
Creating our own 'Player GameObject'

(!) Create a new Folder called 'Resources'



As many of you may know already, the contents of folder named 'Resources' are available to dynamic loading with 'Resources.Load()'

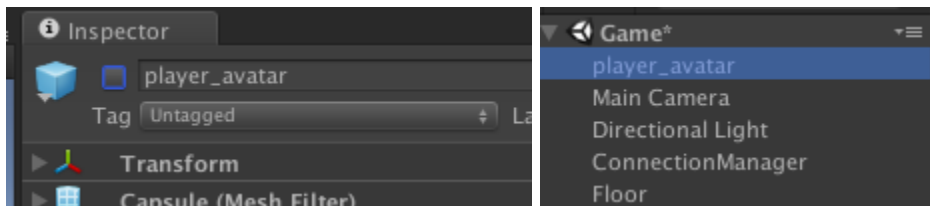
(!) Make the *player_avatar* a prefab by dragging it from the hierarchy into the **Resources** folder in the *Project* tab.



It ***MUST*** go in the '**Resources**' folder

(!) Disable the *player_avatar* object in the hierarchy

We will be modifying this prefab, so it will be helpful to keep it around in the scene.



(!)Code along : [PhotonConnectionManagerIncomplete](#),

we're going to add the following function

```
void OnJoinedRoom()
{
    GameObject localPlayer = PhotonNetwork.Instantiate("player_avatar", Vector3.zero,
    Quaternion.identity, 0);
    localPlayer.name = "local avatar";
}
```

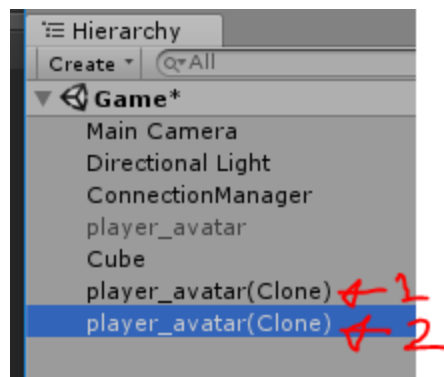
Explanation:

OnJoinedRoom() is automatically called when we join a room. Now when a player first joins a room, they'll instantiate their own avatar object. When a player creates an object with PhotonNetwork.Instantiate, that player owns the object.

(!) Build your game again

(!) Run the build, and the Editor..

you'll see two new objects in the hierarchy, and you'll be able to correctly move the one labeled "local avatar (Clone)".

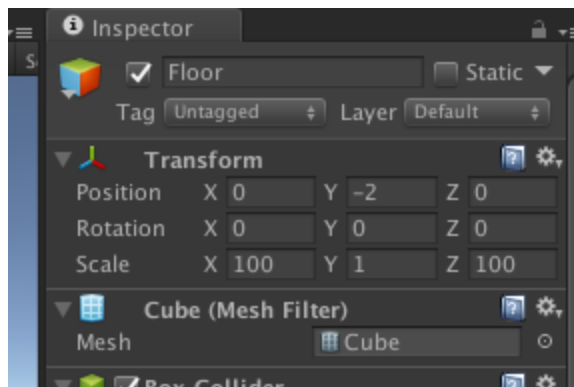
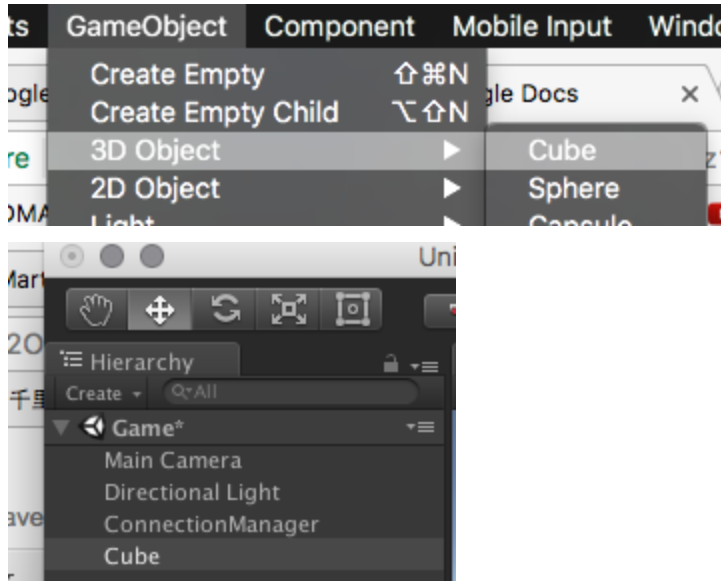


Experiment: What will happen if you try to move the these in the editor??

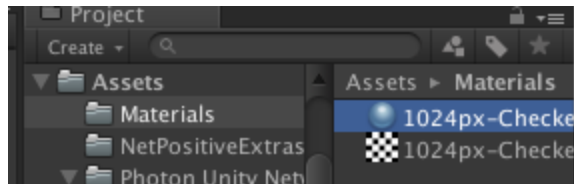
Quickly Add a floor

Our character will be affected by gravity, so we'll need some kind of ground.

(!) Create a big cube for the floor, put on a checkerboard. Call it "floor".



I Recommend you make its scale (100,1,100) and its position it at (0,-2,0)



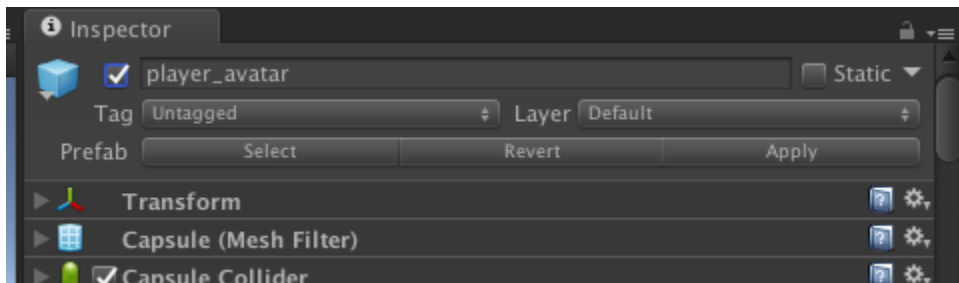
I also recommend you put a texture on it

Make your player keyboard controlled

Warning: Make sure you stop the Editor, and your build before moving on to this step! Otherwise you'll lose your changes and have to do it all over again!

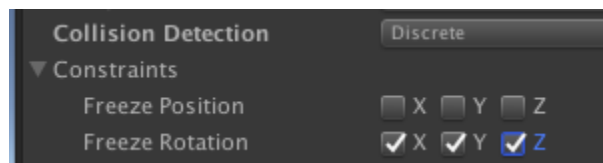
Moving the objects in the editor is alright for testing... but let's make it so we can control each object with the keyboard. First, we need to update our 'player_avatar' prefab.

(!) unhide the player_avatar object



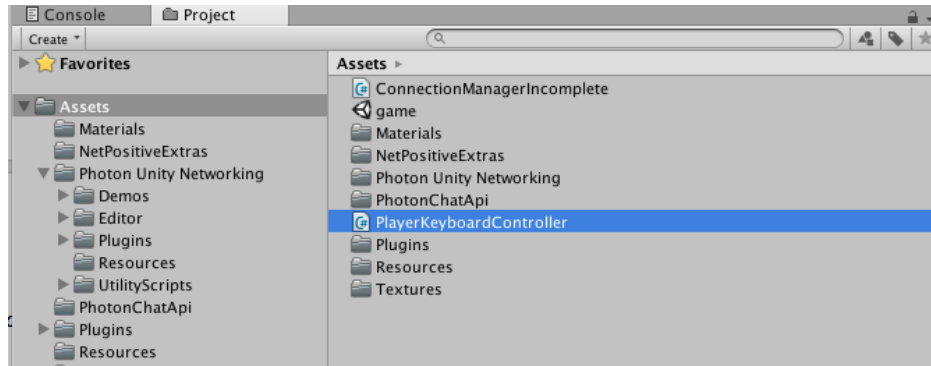
(!) Add a rigid body to the player_avatar object

(!) on the rigidbody, in **Constraints**, Check **Freeze Rotation** for X, Y & Z



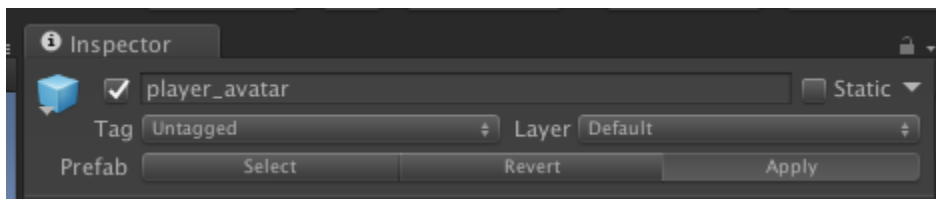
Freeze rotation for x,y, and z

(!) Add a **PlayerKeyboardController** script to the 'player_avatar' prefab.



This is a very simple premade script I'm providing.

(!) Hit '**Apply**' at the top of the inspector apply these changes to the **player_avatar** prefab

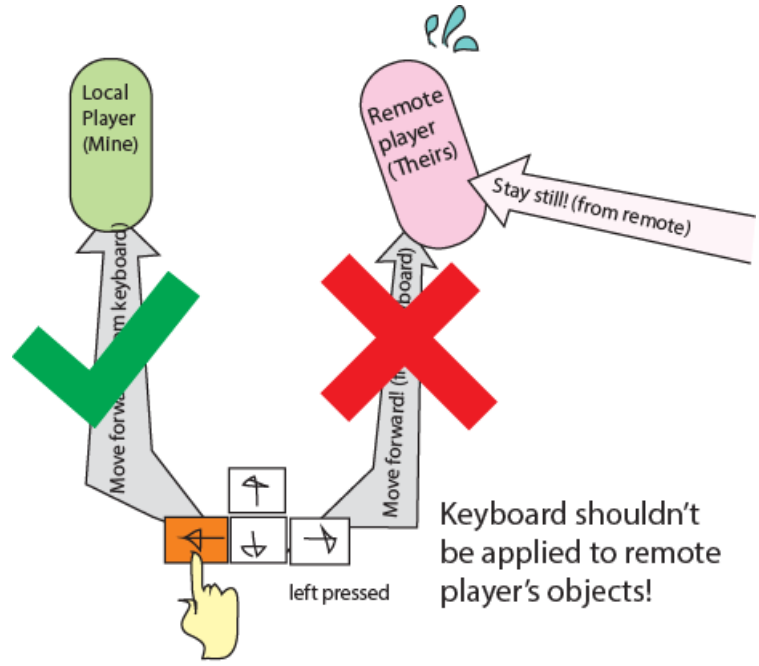
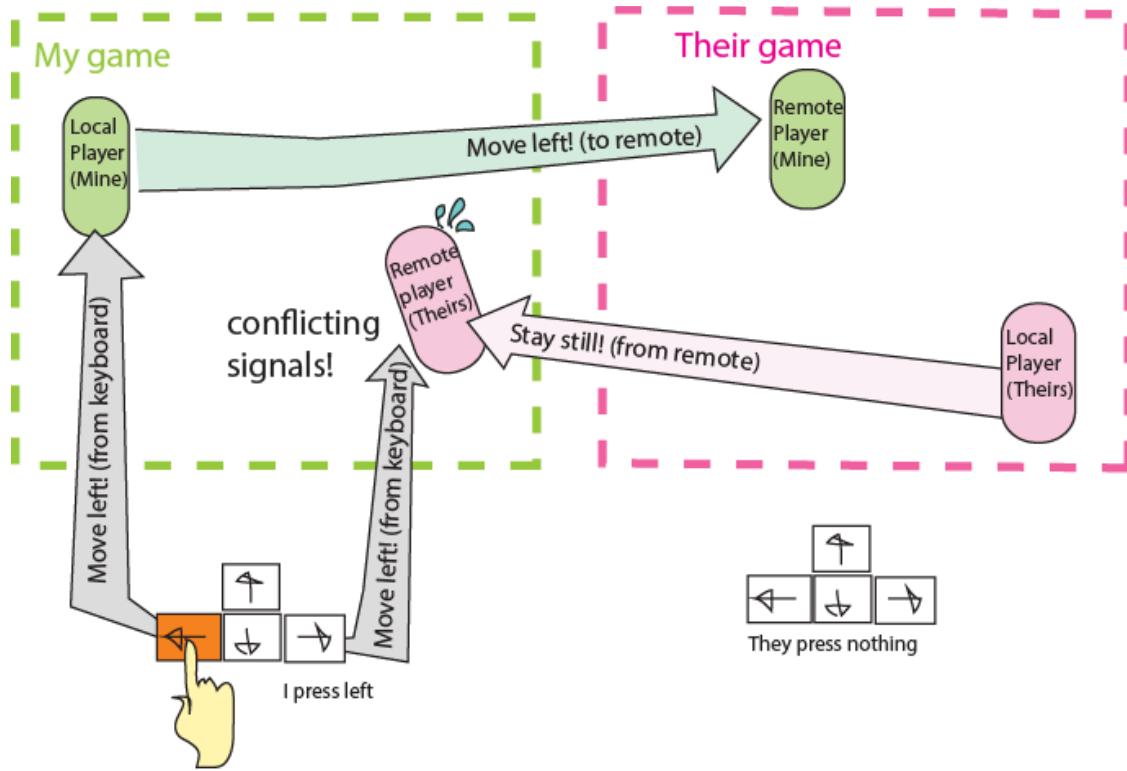


(!) hide the **player_avatar** object again

Try it out, and you can see that it kind of works, but that both players move when you hit the keys.

What's wrong?????

Both our local player, and the remote player have Keyboard.cs attached. My keyboard movements are applied to both my own player object, and the remote player object. On the remote player object shouldn't be affected by my keystrokes, and these competing with the remote player's "true" position in the build.



Turn off PlayerKeyboardController.cs for any player we don't own

(!) Code along : open **PlayerKeyboardController**, and change its Start() function to the below:

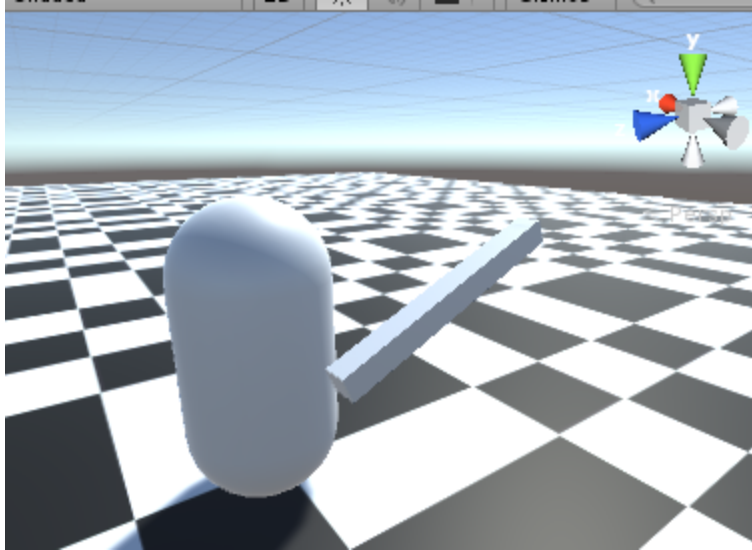
```
void Start ()
{
    if (!this.GetComponent<PhotonView>().isMine)
    {
        this.GetComponent<Rigidbody>().isKinematic = true;
        this.enabled = false;
    }
}
```

After you save, and play again, the player should move separately.

The line: `this.GetComponent<PhotonView>().isMine` is **true** if we own this object, and changes we make (e.g. position) will be sent to other participants in the game. If it's **false** it's owned by a player on another computer, and we can only listen for their changes. We only want to move our own local object with keyboard, so we should disable the *PlayerKeyboardController* for any player object that isn't ours. Similarly, we should also disable the rigid body (done by setting `isKinematic=true`). Remember, the owner of an object is the authority on its state (in this case its position/orientation). Simulating physics on an object changes its position/orientation! If we run physics on an object we don't own, our local physics system will set a position for the object in competition with the "true" position sent from the remote player.

Making a Weapon

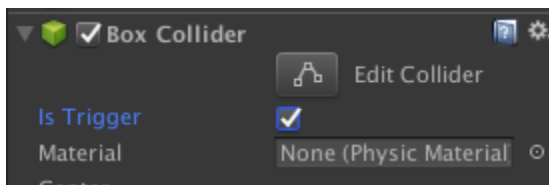
Let's make a "sword" our players can swing by hitting space.



(!) Make a cube and stretch into a tall, thin stick.

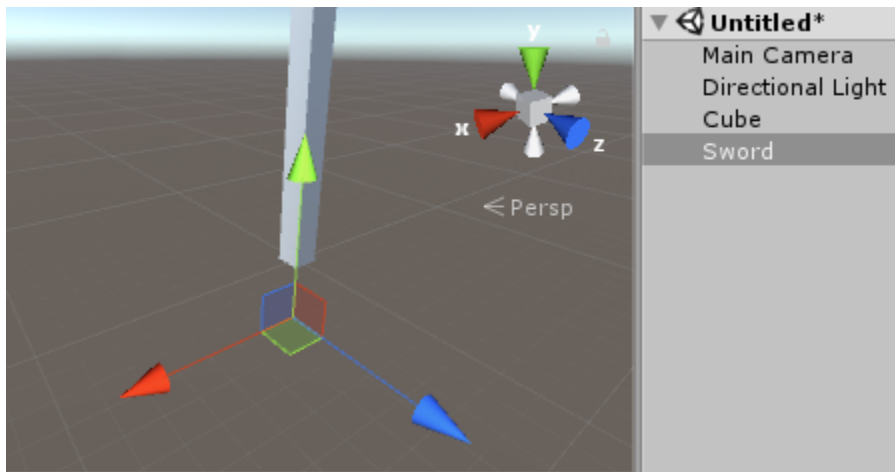
I recommend a scale of (0.1, 2, .1)

(!) Check “isTrigger” on the stick’s collider



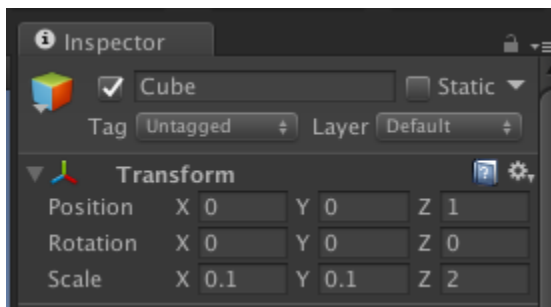
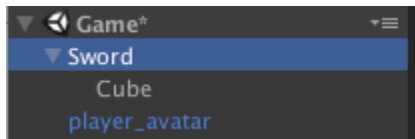
(!) Make an empty game object, and line it up with the bottom of your stick (a little bit past is good)

(!) Rename the empty game object “sword”



The empty ‘sword’ object should be just below the stick (lined up with the green y-handle)

(!) Child the stick to this empty “sword” object

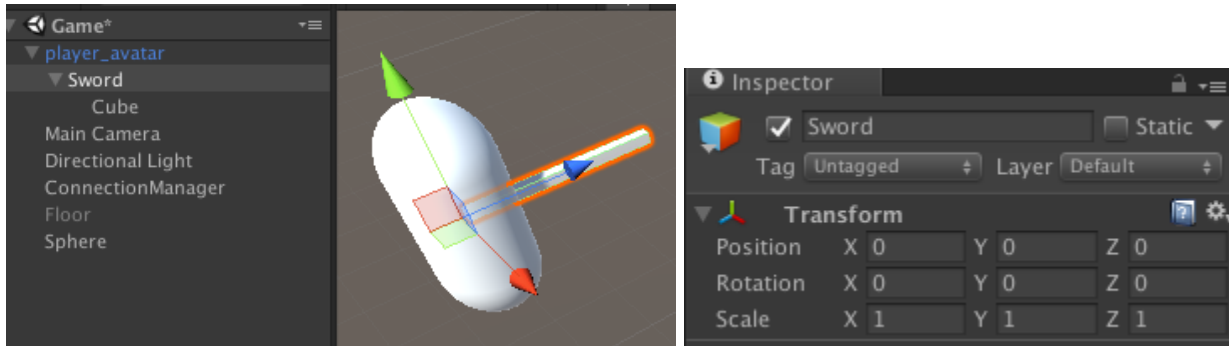


(A good position for the stick once it's a child of 'sword' (0,1,0) if you also used my recommended scale)

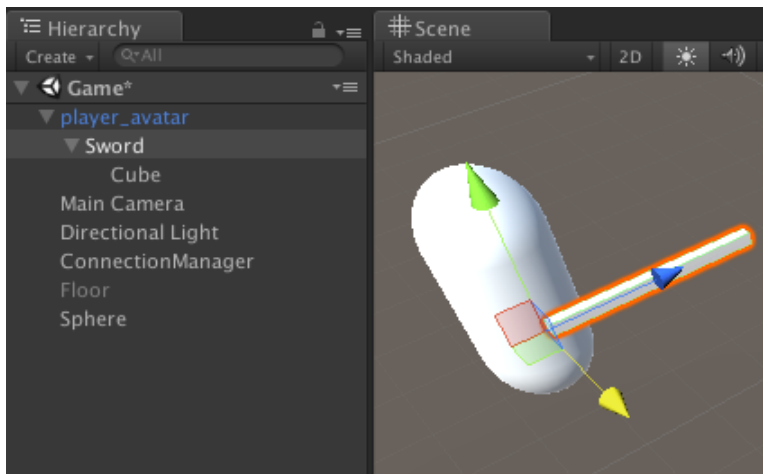
(!) Unhide *player_avatar* and child your sword to *player_avatar*

(!) Select the “sword” in the hierarchy

(!) Zero out the “sword” object’s position, in the inspector and optionally move it just to the right of the player.

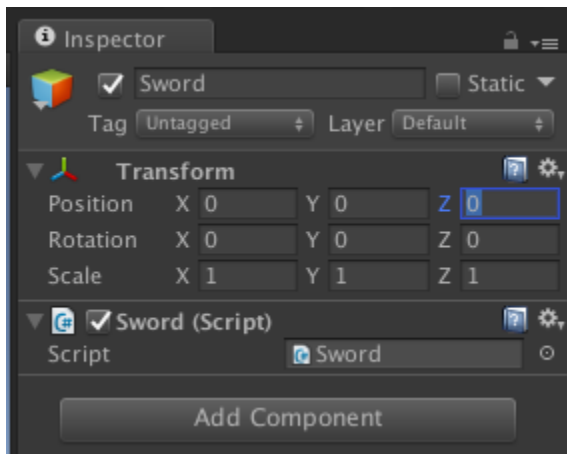


Sword at (0,0,0) position



Pull the red move handle, so the sword is at the player's side

(!) Add a new Script to your sword object, calling the script 'Sword'



(!) code along : Add the following function to 'Sword' script (it can be called to swing the sword)

```
public void swingSword()
{
    //make the sword appear
    this.gameObject.SetActive(true);

    //This function, varyWithT() performs a 1-time animation specified in code
    //It takes 2 arguments:
    //1st: an animation function
    //2nd: the duration of the animation
}
```

```

this.varyWithT(

//Animation function, called over repeatedly the 'animation duration'
(float t) =>
{
    //t is is the 'normalized animation time'
    // t = '0' at beginning of animation
    // t = '.5' halfway through
    // t = '1' at end of animation

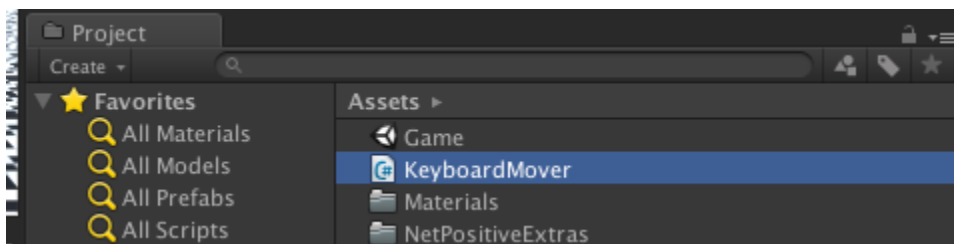
    //Move the Sword from pointing up, rotation = (-90,0,0) ...
    // ... to pointing forward, rotation = (0,0,0)
    this.transform.localEulerAngles =
        Vector3.Lerp(new Vector3(0, 0, 0), new Vector3(90, 0, 0), t);

    //at the end of the animation, make the sword disappear
    if (t == 1)
    {
        this.gameObject.SetActive(false);
    }
},

//Animation duration
.2f
);
}

```

(!) Select *player_avatar* and apply the changes to the prefab.



(!) Disable *player_avatar* in the hierarchy again.

Actually swing the sword by calling this function

(!) Create a new Script called `CharacterSheet`, and add it to `player_avatar`

(!) In `CharacterSheet`, add the following function

```
void useWeapon()
{
    this.GetComponentInChildren<Sword>(true).swingSword();
    //the 'true' above has GetComponentInChildren also check in inactive children
    //which our sword often is!
}
```

(!) Also, in `PlayerKeyboardController`, change `Update()` to match the following:

```
void Update()
{
    moveCharacterViaKeyboard();

    if (Input.GetKeyDown(KeyCode.Space))
    {
        this.GetComponent<CharacterSheet>().useWeapon();
    }
}
```

Test it out: can you swing your sword?

If you try it out now, when you hit space, your player's sword is working OK, but you can't see the other players swinging their swords! The function `useWeapon()` is only being called *locally*, on our single instance of the game. We want it to be called on everyone's else game too

RPC (Remote Procedure Call)

To call a function over the network, we use something called a *Remote Procedure Call* (RPC) <https://doc.photonengine.com/en-us/pun/current/manuals-and-demos/rpcsandraiseevent>

First, we must designate the function as an *RPC*

(!) In *CharacterSheet* add the line `[PunRPC]` just above `swingWeapon()`, like so:

```
[PunRPC] //marks a function so that it can called over the network
void useWeapon()
{
    this.GetComponentInChildren<Sword>(true).swingSword();
}
```

Next, we use the method `RPC()` of our object's *PhotonView* component to call that object:

(!) Back in *PlayerKeyboardController* replace the `useWeapon()` line in `Update` to match the following:

```
...
if (Input.GetKeyDown(KeyCode.Space))
{
    this.GetComponent<PhotonView>().RPC("useWeapon", PhotonTargets.All);
}
...
```

Try it out, and you should see the other player swinging their sword.

Breaking RPC line down...

```
this.GetComponent<PhotonView>().RPC("useWeapon", PhotonTargets.All);
```

`PhotonView.RPC` takes 2 arguments, first, **name of the function** you want to call, and 2nd, the **target players**, on the network you want receive the function. Here, we're calling it on every connected player, including ourselves. It's also possible to call a function on a specific connected player, or subset of players, but we won't get into any of those cases today

More info about *PhotonTargets*:

https://doc-api.photonengine.com/en/pun/current/group__public_api.html#gab84b274b6aa3b3a3d7810361da16170f

An Aside: Why don't we also sync the swords transformation info, like we did for the player?

Interfaces (making things respond to sword hits)

Let's make it so things respond to being hit with sword.

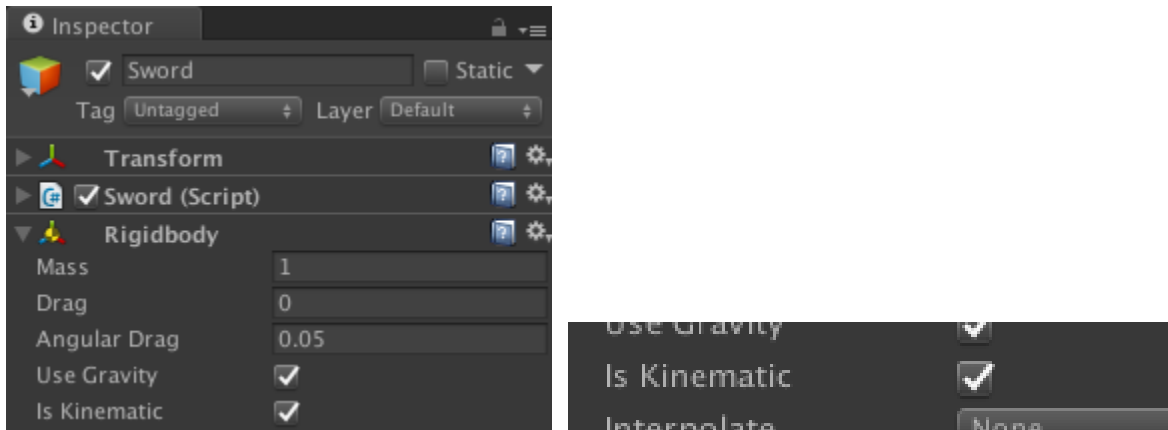
First, let's just print to the console when our sword hits something.

(!) Open [Sword](#) script, and add the following

```
public void OnTriggerEnter(Collider other)
{
    Debug.Log("Sword hit " + other.name, other.gameObject);
}
```

(!) We should also add rigidbody to the sword, and check is kinematic, so our sword will recognize objects without rigidbodies.

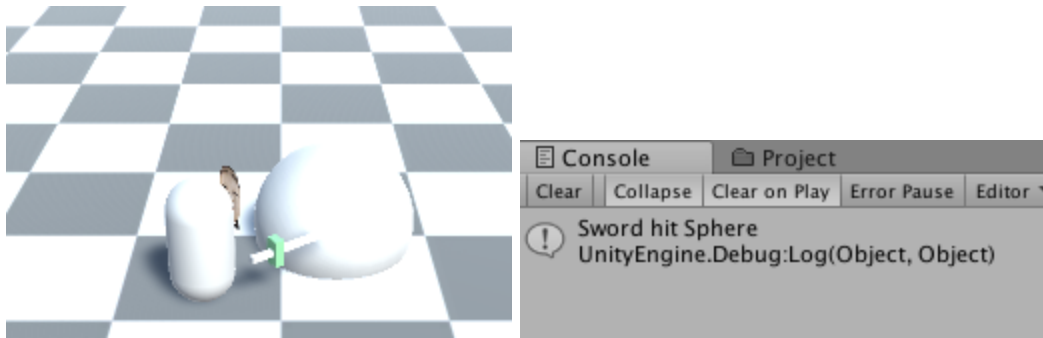
(FYI, at least one object in trigger collision needs a rigid body)



Select the sword, add rigid body, check '**Is Kinematic**'

(!) Add a sphere or cube to your game, and run, (just the editor is fine), checking the console as you hit space to swing the sword to see what your sword is hitting

We might be hitting our own player! The google doc [Sword](#) will be updated accordingly to fix this if necessary.



You should see something like this

Making 'Hittable' things

What we really want, is for the sword to act on a wide range of “hittable” objects, which will respond to being hit in their own way. Enter, the *interface*.

(!) Make a new script, [IHittable](#)

(!) Open, delete everything and paste in below

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public interface IHittable
{
    void takeHitLocally(int damage);
}
```

This script is called an interface, and won't be attached to anything directly. Instead, it defines a new *type* of object, which always has the function, `takeHitLocally(int damage)`. Thus, in sword, we can check for any objects of type *IHittable*.

(!) Add the below to [Sword](#)

```
public void OnTriggerEnter(Collider other)
{
    Debug.Log("Sword hit " + other.name, other.gameObject);

    IHittable otherThingHittableScript = other.GetComponent<IHittable>();

    if (otherThingHittableScript != null)
    {
        otherThingHittableScript.takeHitLocally(1);
    }
}
```

In this new code, the sword checks if the the thing colliding *implements* contains any *IHittable* scripts, in which case we can call its *takeHitLocally()* function.

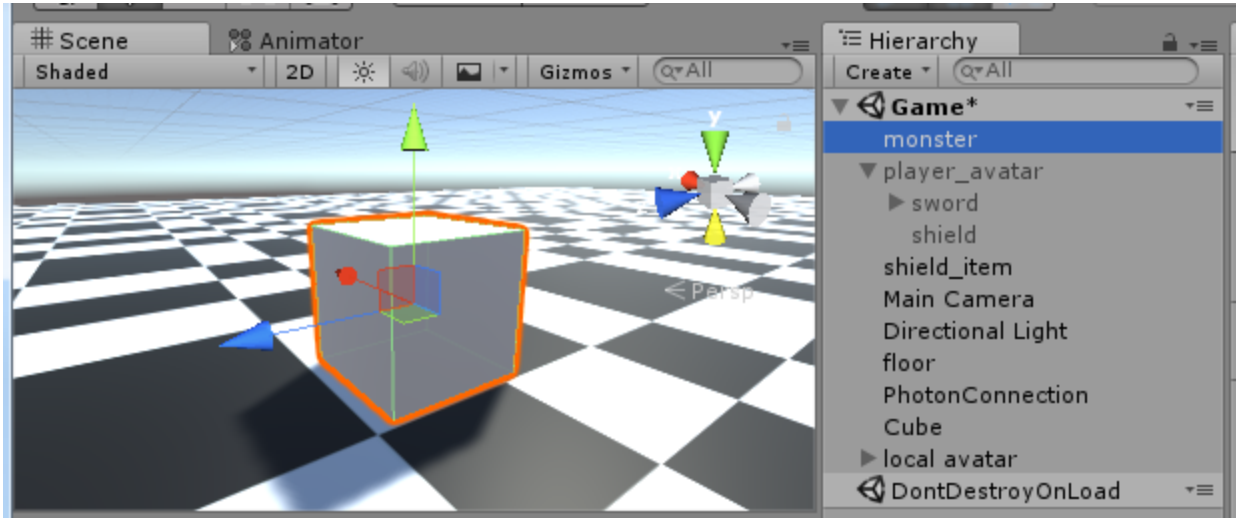
Implementing Hittable, to give things HP

There are no *IHittables* at the moment, but fortunately, *any* script can become an *IHittable* by doing 2 things. 1: Declaring itself as *implementing IHittable*, and 2: including public versions of all the functions in the interface.

Let's make a hittable script to make an object disappear if hit by a sword, a specified number of times. First we'll use it on a killable monster/destroyable wall.

(!) Make a cube

(!) Name it *monster*



(!) Add a **RigidBody** component

(!) Make and attach new Script, *HittableAndDisappear*

(!) Alter the following “, IHittable” after “MonoBehaviour” in *HittableAndDisappear*

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class HittableAndDisappear: MonoBehaviour, IHittable
{
    ...
}
```

This designates [HittableAndDisappear](#) as implementing 'IHittable'. But we're getting a compiler error!

(!) Add the function `takeHitLocally(int Damage)` to **HittableAndDisappear** to finish implementing the 'IHittable' interface.

```

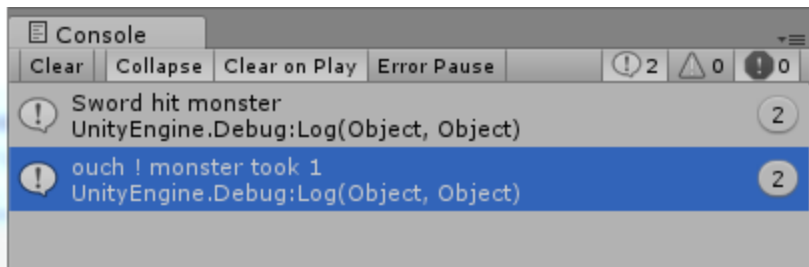
public void takeHitLocally(int damage)
{
    print("ouch! " + this.name + " took " + damage + " damage");

    //==== flash the object red =====
    //turn the object red...
    this.GetComponent<Renderer>().material.color = Color.red;

    //then turn it back white .25 seconds later
    this.delayedFunction(
        () => { this.GetComponent<Renderer>().material.color = Color.white; }
        ,
        .25f);
    //=====
}

```

(!) Run the game, (just in the editor is fine), a check that hitting the monster generates an 'ouch' message in the console.



(!) Add a public variable for hp.

```

public class HittableAndDisappear: MonoBehaviour, IHittable, IPunObservable
{
    public int hp = 100; //how much sword damage object can take before
    disappearing
    ...

```

(!) Update takeHitLocally to include subtract 'damage', and destroy the object

```

public void takeHitLocally(int damage)
{
    print("ouch! " + this.name + " took " + damage + " damage");

    //==== flash the object red =====
    //turn the object red...
    this.GetComponent<Renderer>().material.color = Color.red;

    //then turn it back white .25 seconds later
    this.delayedFunction(
        () => { this.GetComponent<Renderer>().material.color = Color.white; }
        ,
        .25f);
    //=====

    //--- apply the damage, and destroy the object if hp goes to 0 ----
    this.hp -= damage;
    if (hp <= 0)
    {
        Destroy(this.gameObject);
    }
}

```

(!) add an OnGUI function to display the hp.

...

```

//paste before the last closing curly brace '}' in the file
void OnGUI()
{
    //Display the HP
    AlexUtil.DrawTextAtWorldPosition(
        this.transform.position,
        "HP : " + this.hp,
        24,
        Color.magenta,
        new Vector2(0,35)
    );
}

```

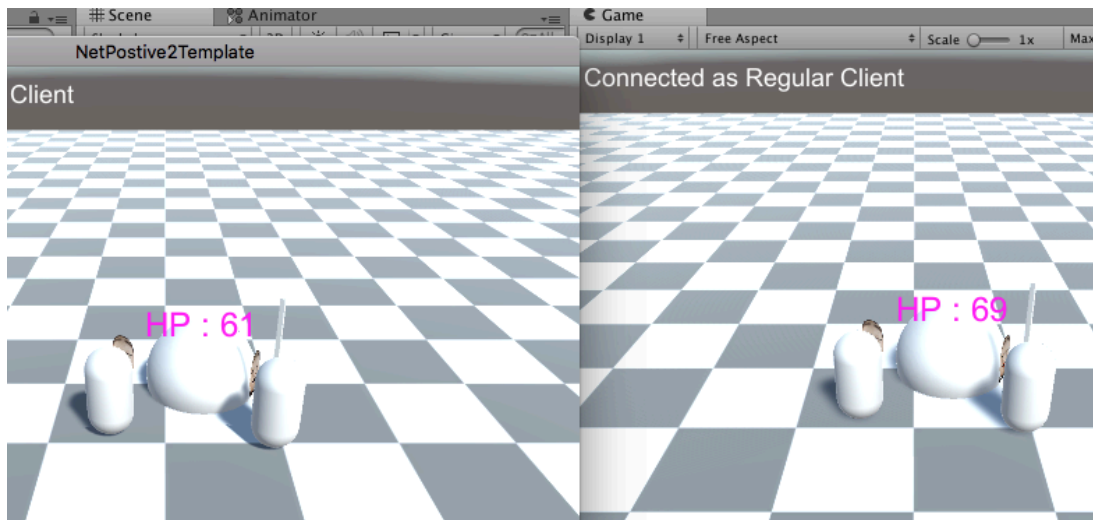
(bonus) Making a HittableDoor

Add this script, [HittableDoor](#) to an object and it will lift up when hit with a sword. You'll need to add Photon view, and a PhotonTransformView.

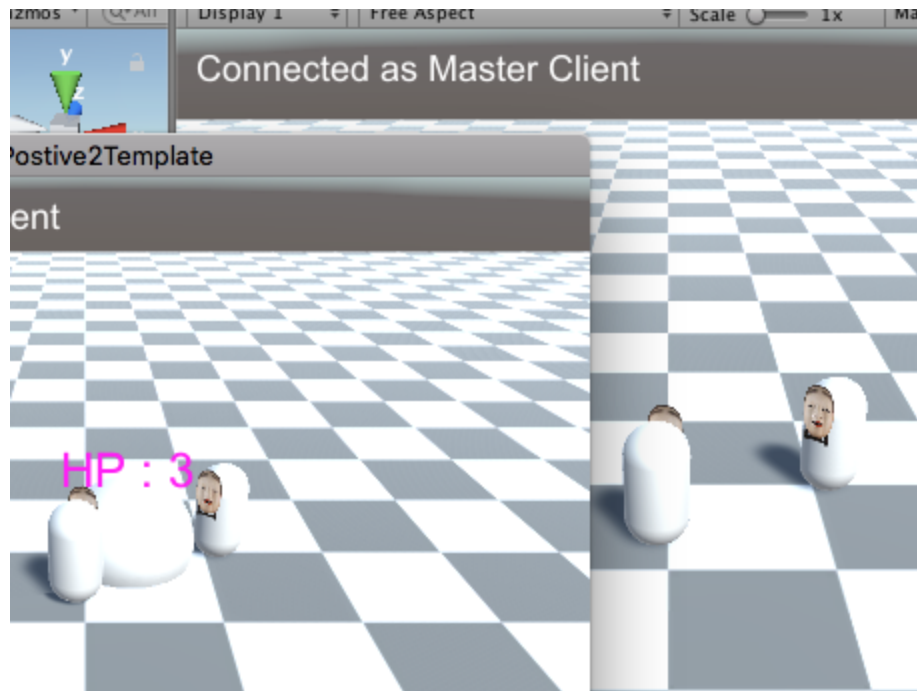
HP, Out of Sync

If you run your game with 2 players, give the monster 100 hp, and hit it a bunch, you'll probably see the *hp* goes out of sync between the 2 games. What's happening?

Also, when one side reaches zero, disappears, it only disappears there



61 in one game, 69 in the other??



It disappeared on the master client side, but not on the regular client side!

Syncing Object Destruction

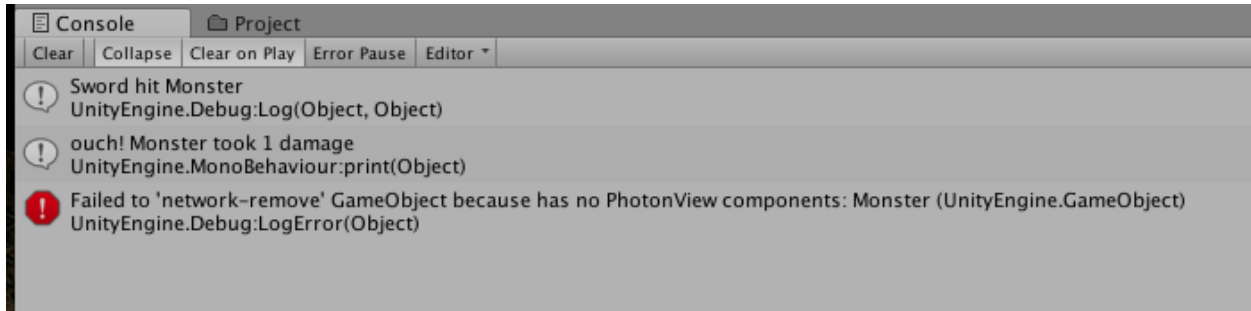
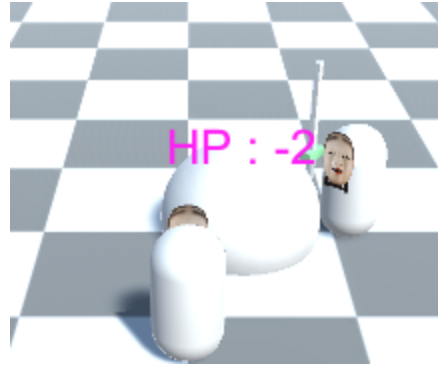
We can fix the inconsistent disappearance by using **PhotonNetwork.Destroy()** instead of just **Destroy()**

(!) Change **Destroy(...)** to **PhotonNetwork.Destroy(...)** in **HittableAndDisappear**

...

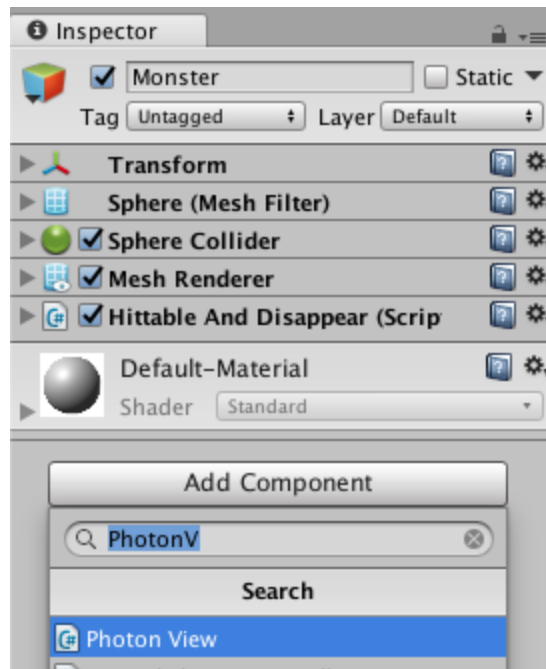
```
this.hp -= damage;
if (hp <= 0)
{
    //Destroy the object...
    //If you want to Destroy a networked object
    //You need to use PhotonNetwork.Destroy, otherwise
    //the object will only be destroyed locally
    PhotonNetwork.Destroy(this.gameObject);
}
```

If you try to run now, you'll get an error when Monster's HP goes to zero.
PhotonNetwork.Destroy requires a PhotonView component!



Will fail to destroy is monster has no PhotonView component

(!) Add a **PhotonView** component to **Monster**



Apply Damage over the Network with an RPC

Maybe, if we applied our damage across the network, that'd fix things.

(!) Add to **HittableAndDisappear**, a new function **applyDamage()**, we'll call as an RPC

```
...
public void takeHitLocally(int damage)
{
    this.GetComponent<PhotonView>().RPC("applyDamage", PhotonTargets.All, damage);
}

[PunRPC]
public void applyDamage(int damage)
{
    hp -= damage;
    if (hp <= 0)
    {
        PhotonNetwork.Destroy(this.gameObject);
    }
}
...
```

Broken, in a new way

If we try now... the enemy is taking damage twice! What's going on?

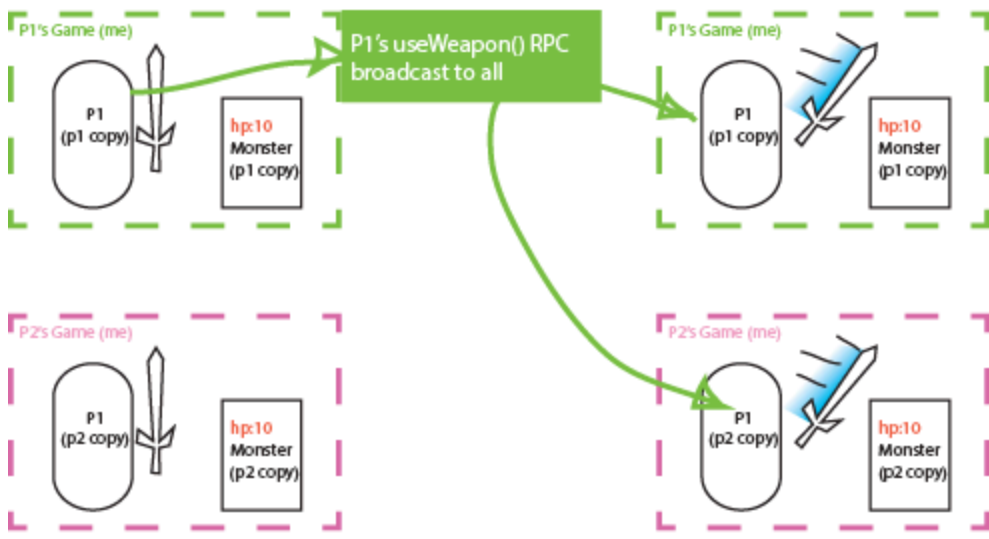
Player1's sword exists in 2 places...

When player1 swings the sword, OnTriggerEnter is being checked both locally, and remotely

Thus... both the local sword sends out an applyDamage RPC to everyone on the network. meanwhile, the remote sword is also getting and OnTriggerEnter(), and sending applyDamage() as an RPC to everyone as well. If we added a 3rd player, it would actually get sent a 3rd time!

time 1

time 2

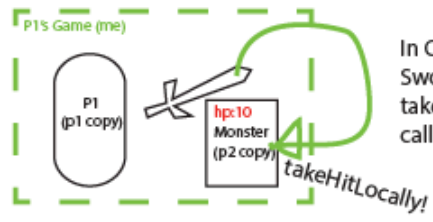
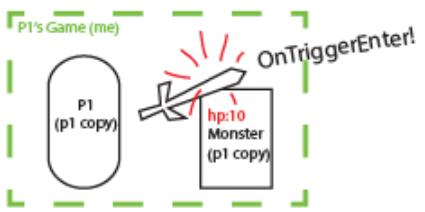


P1 pressed attack button

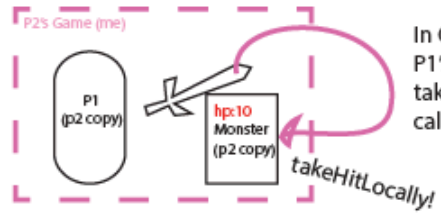
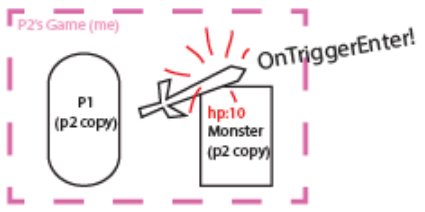
Sword moves in both games

time 3

time 4

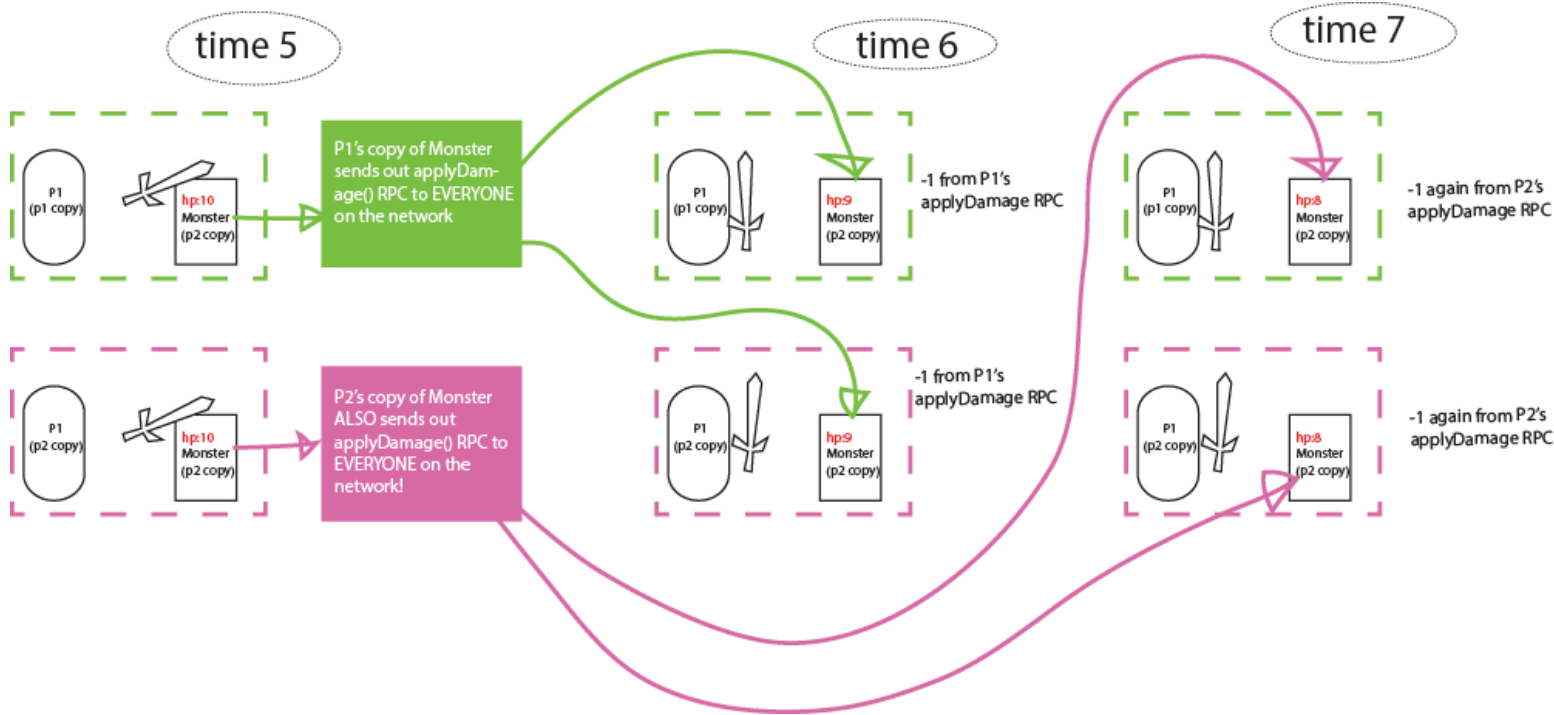


In OnTrigger, P1's local Sword calls takeHitLocally() called on monster



In OnTrigger, P2's copy of P1's Sword also calls takeHitLocally() called on monster

Both swords get OnTriggerEnter() in their copy of the game!



Moreover, the conditions of `useWeapon`'s animation (gets triggered at slightly different times because of latency), `Framerates` might also be different across different games, and so the conditions leading up to `OnTriggerEnter` are not going to be exactly the same for each of our games, so some hits might get lost, or doubled, and our HP's can go out sync.

An Aside: Potentially, this isn't that big of a deal! Small differences like this, (the HP of an enemy being slightly different for each player). Sometimes, it's worth living with little inconsistencies like this, since it can save you bandwidth.

Check for `OnTriggerEnter` only locally

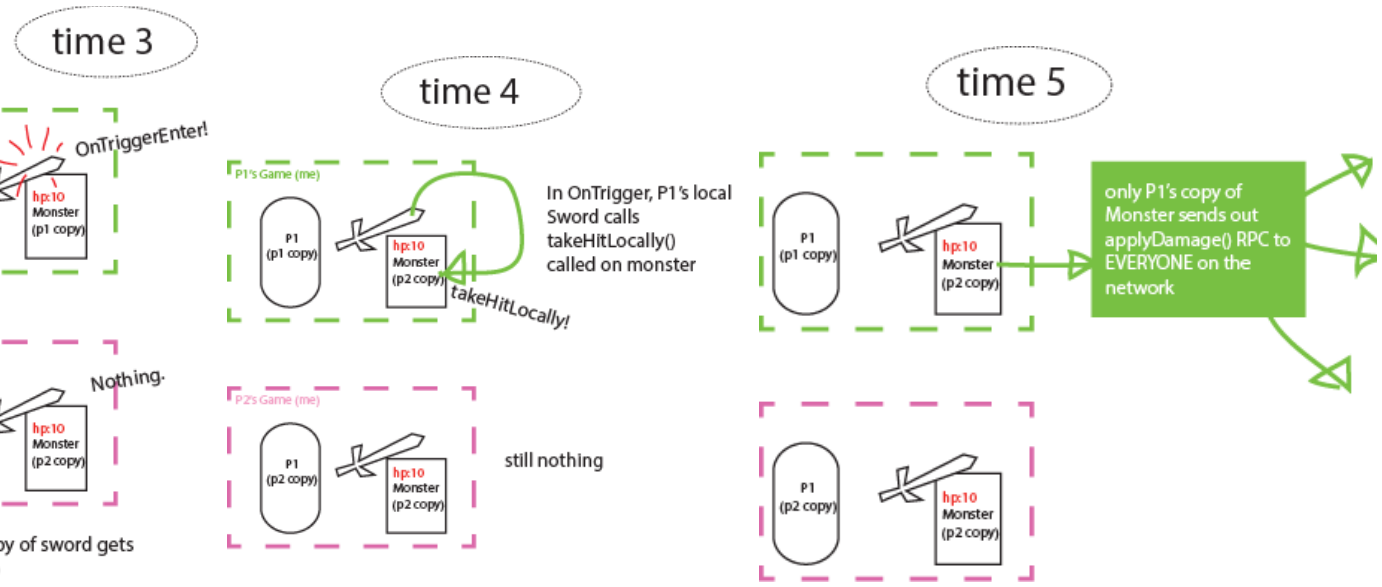
We named the function `takeHitLocally()` for a reason. It makes sense for only the *local* version of a sword to check if hits something, and then for the `applyDamage()` RPC to propagate from just the player who swung the sword.

(!) update `OnTriggerEnter` in **Sword** to the following

```

...
public void OnTriggerEnter(Collider other)
{
    bool isLocalSword = this.GetComponentInParent<PhotonView>().isMine;
    if (isLocalSword && other.GetComponent<IHittable>() != null)
    {
        other.GetComponent<IHittable>().takeHitLocally();
    }
}
...

```

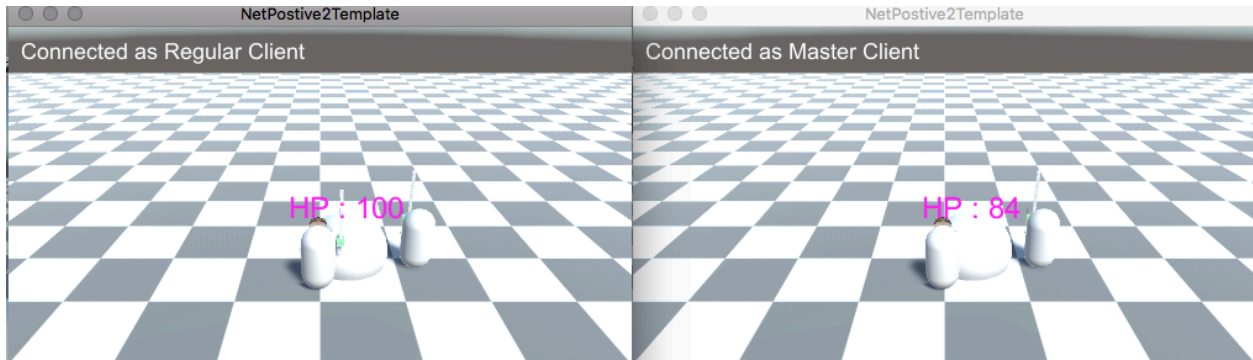


Now, our local sword swings are purely decorative on the remote side, and we decide if we should apply damage in just one place, and HP won't drift.

Still out of sync in another case though...

If you start a game by yourself, and deal some damage, and then another player joins...

The new player sees an enemy with full HP!



Regular client showed up late, and doesn't have the correct HP

Syncing HP (and other vars) over the network

What we really want, is for there to be just one, true value of our monsters HP, and this one true value is applied all across the network.

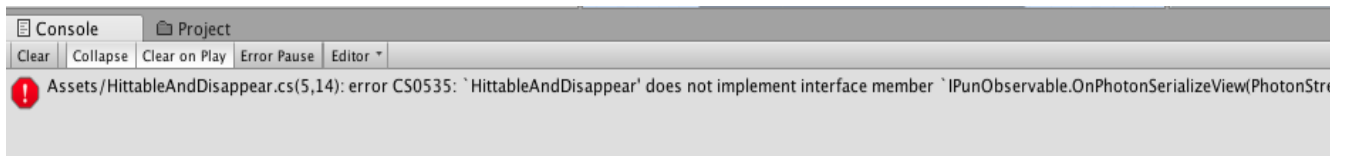
This is actually what was happening with *PhotonTransformView* from the beginning. The *owner* of the object, the master client had the one true position, and all the other clients receive the true value from the owner. We can do this with other variables too!

(!) Update the top of `HittableAndDisappear`

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class HittableAndDisappear : MonoBehaviour, IHittable, IPunObservable {
...
}
```

We're actually implementing the interface `IPunObservable`, so will get an error until we add the function required by `IPunObservable`.



(!) Also to `HittableAndDisappear`, add the required function, `OnPhotonSerializeView(...)`


```

public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
    if (stream.isWriting)
    {
        // We own the object, and this variable, thus send it to the other
        players
        stream.SendNext(HP);
    }
    else
    {
        // we do *not* own this variable, and here we receive it's value
        //from the owner
        HP = (int) stream.ReceiveNext();
    }
}

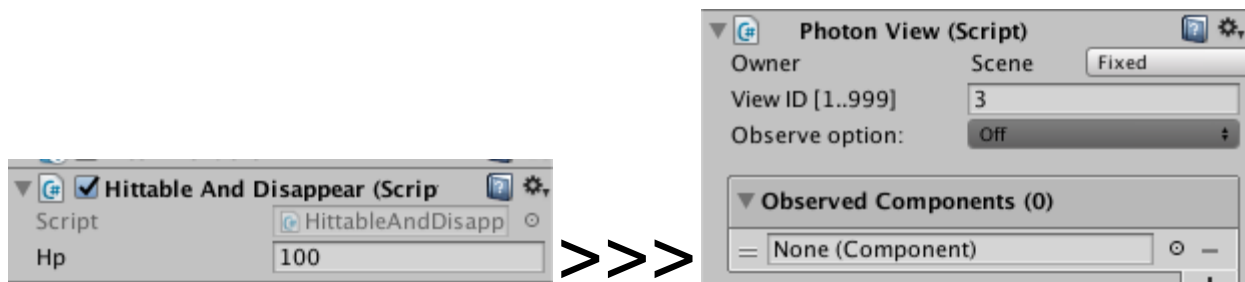
```

All `OnPhotonSerializeView` functions will be of this form: In the 'stream.isWritingBranch' 'stream.SendNext()' call with each variable we want to receive, and in the 'else' branch, reading those values out in the same order. The '(bool)' part is called a cast. `stream.ReceiveNext()` returns the variables as a generic 'object', and this cast is used to convert it back to its proper specific type (this case, an int).

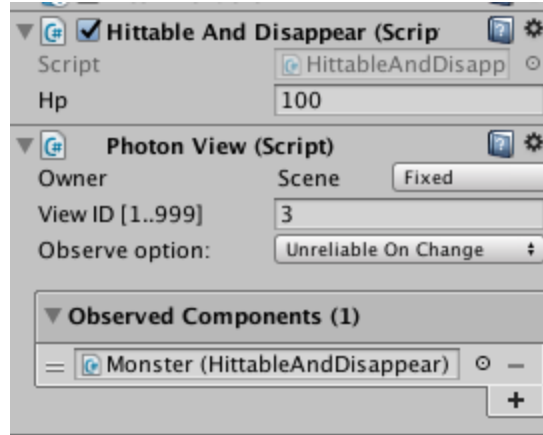
More info on `OnPhotonSerializeView()`

https://doc.photonengine.com/en-us/pun/current/getting-started/feature-overview#_observembo

(!) In the Inspector, Drag and drop **Monster's HittableAndDisappear** component onto the empty **Observed Component** slot on the *PhotonView*



like we did for the PhotonTransformView at the start, drag onto the ObservedComponent slot



It should look like this

Now if a player joins the game late, the enemies HP will still be right!

A small tweak.

(!) Update HittableAndDisappear to the below...

```
[PunRPC]
public void applyDamage()
{
    //Play a little animation (grow in size) when hit
    this.varyWithT((float t) => {
        this.transform.localScale = (1 + .1f * Mathf.PingPong(2 * t, 1)) *
Vector3.one;
    }, .1f);

    //but actually only the owner to variable should be changing it!
    if (this.GetComponent<PhotonView>().isMine)
    {
        hp--;
        if (hp <= 0)
        {
            PhotonNetwork.Destroy(this.gameObject);
        }
    }
}
```

<make clearer>

(Optional) Make the monster wander around

(!) Add [RandomNPCMover](#) script

(!) Add a **PhotonTransformView** component,

(!) Drag the **PhotonTransformView** onto *Observed Component* slot of the PhotonView Component

(!) Check '*Synchronize Position*' & '*Synchronize Rotation*' on the **PhotonTransformView**

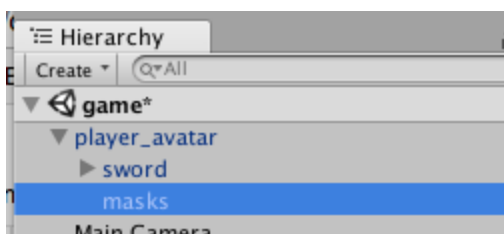
Putting it all together to make Items/Equipment

Let's go back to the player, and create pieces of equipment you can collect around the dungeon. Let's start with some masks.

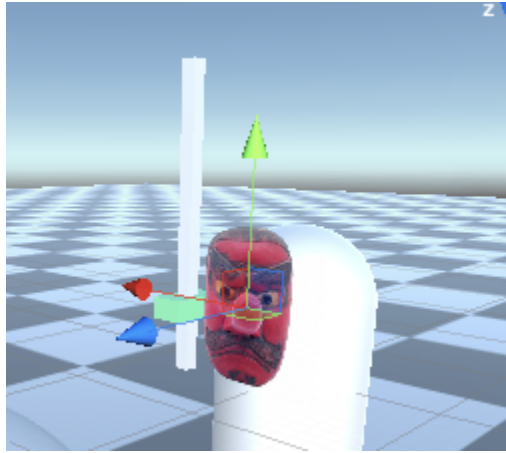
First, we're going to cheat a little, and bake in all the possible items you can collect into the player object.

(!) Select `player_avatar`, and unhide

(!) Create an empty child on it, and name it 'masks'



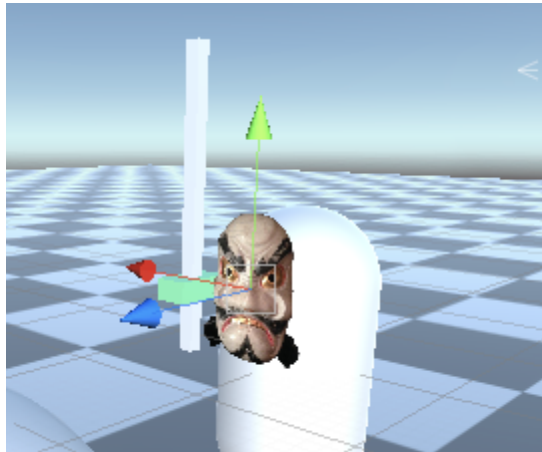
(!) drag and drop a sprite from Assets/Textures/NohMasks onto your mask object, and position them at the front of of the player avatar. Position and resize as you feel led.



(!) give the mask a name.

(!) hide the mask, and repeat the above steps for a couple more masks.





*Position so that they face the positive z-axis (blue handle)
Only 1 mask will be enable at a time.*

(!) select 'player_avatar', create and add a new script called EquipmentGroup

(!) Open EquipmentGroup and add in the below

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EquipmentGroup : MonoBehaviour, IPunObservable
{
    public Transform allItemContainer;
    public string currentlyEquippedItemName;

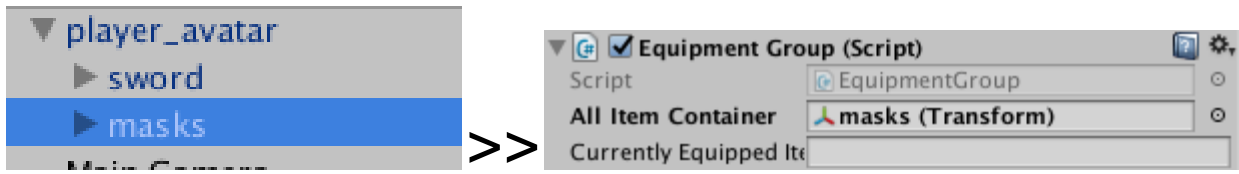
    // Update is called once per frame
    void Update () {
        foreach(Transform item in this.allItemContainer)
        {
            if (item.name == currentlyEquippedItemName)
            {
                item.gameObject.SetActive(true);
            }
            else
            {
                item.gameObject.SetActive(false);
            }
        }
    }

    public bool hasItem(string itemName)
    {
        foreach (Transform item in this.allItemContainer)
        {
            if(item.name == itemName)
            {
                return true;
            }
        }
        return false;
    }

    public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if (stream.isWriting)
        {
            stream.SendNext(currentlyEquippedItemName);
        }
    }
}
```

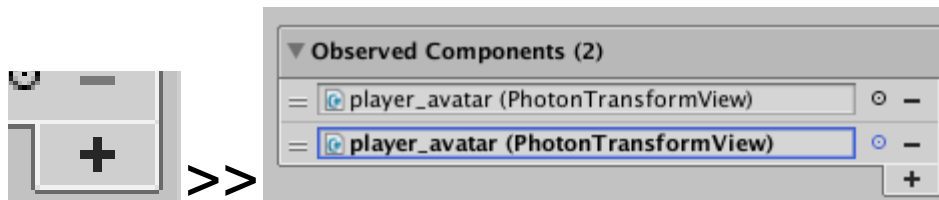
```
else
{
    this.currentlyEquippedItemName = (string)stream.ReceiveNext();
}
}
}
```

(!) drag and drop the 'masks' object with all the mask-children onto the 'All Item Container' slot on Equipment Group.



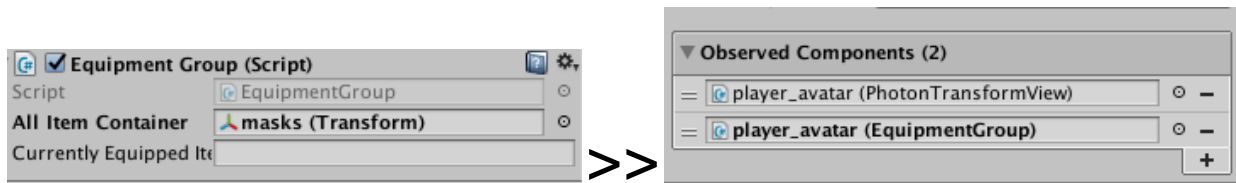
'Masks' onto 'All item Container'

(!) In player_avatar's **PhotonView**, hit the small '+' icon on 'Observed Components' to create a new slot

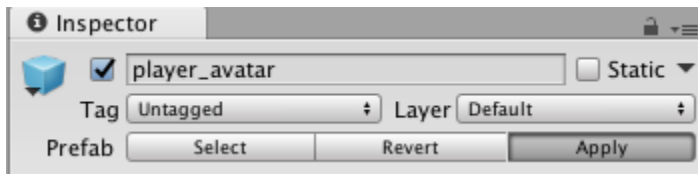


Click the plus to add a second slot!

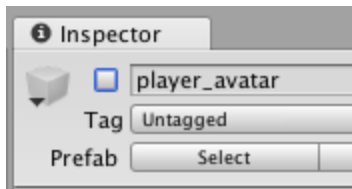
(!) drag and drop **Equipment Group** onto the new slot in the **PhotonView**



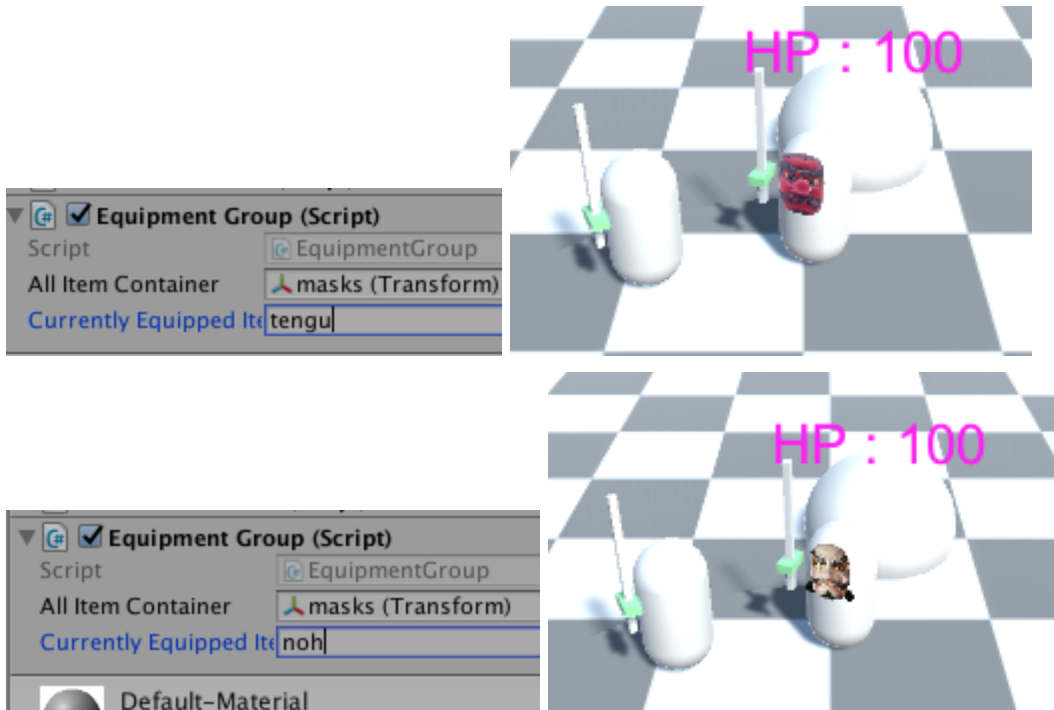
(!) hit 'Apply' on the *player_avatar* prefab



(!) hide 'player_avatar'

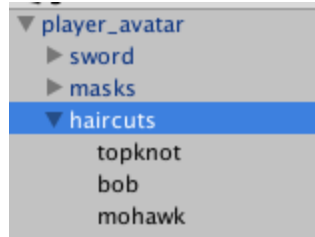


Now try running the game, selecting the your *player_avatar*(clone) object, and setting the value of 'Currently Equipped' on the EquipmentGroup component, and you should see the mask changing, and appearing over the network.

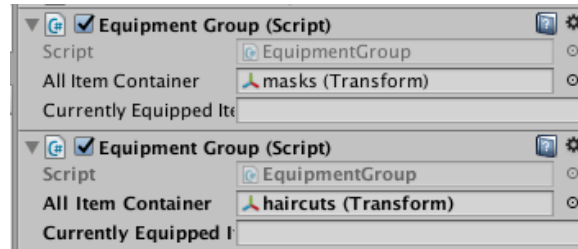


(Optional) You can actually reuse EquipmentGroup for multiple kind of equipment (shields, haircuts). Just add an additional EquipmentGroup component, and an additional child container of possible items and do the same steps as above

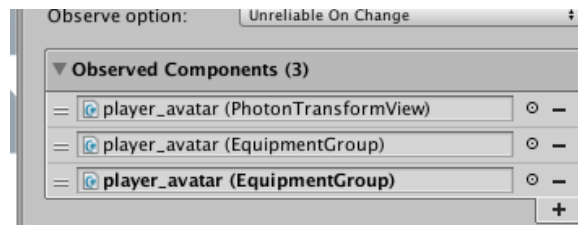
If you were going to also add haircuts...



You'd add another set of children like masks...



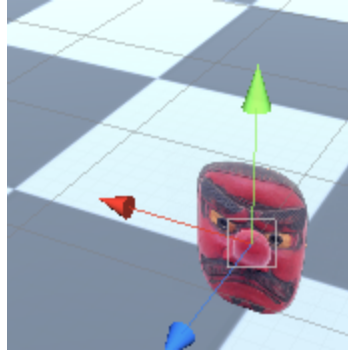
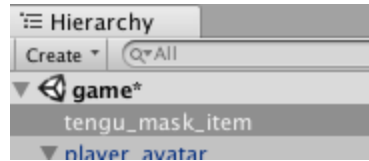
*A second **EquipmentGroup** component...*



*And add the second **EquipmentGroup** to observed components (yes, a little weird/redundant looking)*

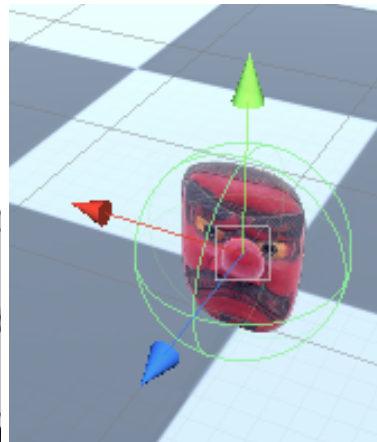
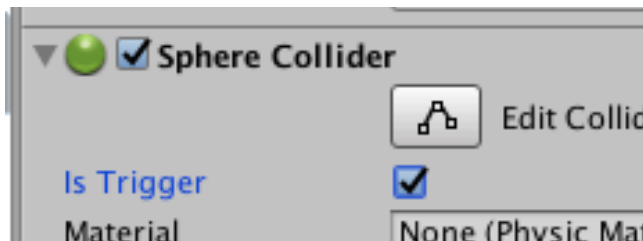
Making Collectible Equipment Items in the World

(!) Drag and drop one of your masks into the world.



I called mine tengu_mask_item

(!) Add a sphere collider and check isTrigger



(!) Add a new script called 'CollectibleItem'

(!) In this script, and fill as below

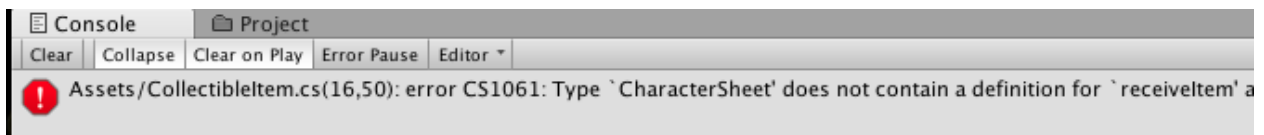
```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CollectibleItem : MonoBehaviour
{
    public string itemName; //the item the collecting player will receive

    private void OnTriggerEnter(Collider other)
    {
        //if a player touches us...
        if (other.GetComponent<CharacterSheet>() != null &&
            other.GetComponent<PhotonView>().isMine)
        {
            //give it the item
            other.GetComponent<CharacterSheet>().receiveItem(this.itemName);
            Destroy(this.gameObject);
        }
    }
}

```



The function receiveItem, in characterSheet doesn't exist yet, let's add it.

(!) Open CharacterSheet and add a new function receiveItem(string itemName)

```

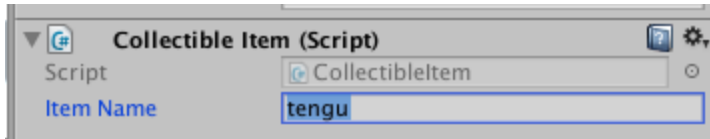
...
public void receiveItem(string itemName)
{
    foreach(EquipmentGroup equipmentGroup in
this.GetComponents<EquipmentGroup>())
    {
        if (equipmentGroup.hasItem(itemName))
        {
            equipmentGroup.currentlyEquippedItemName = itemName;
        }
    }
}

```

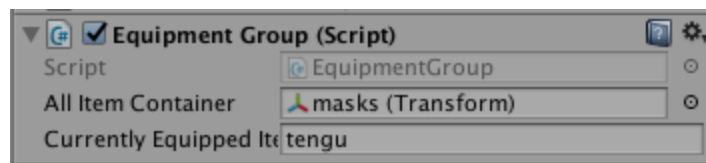
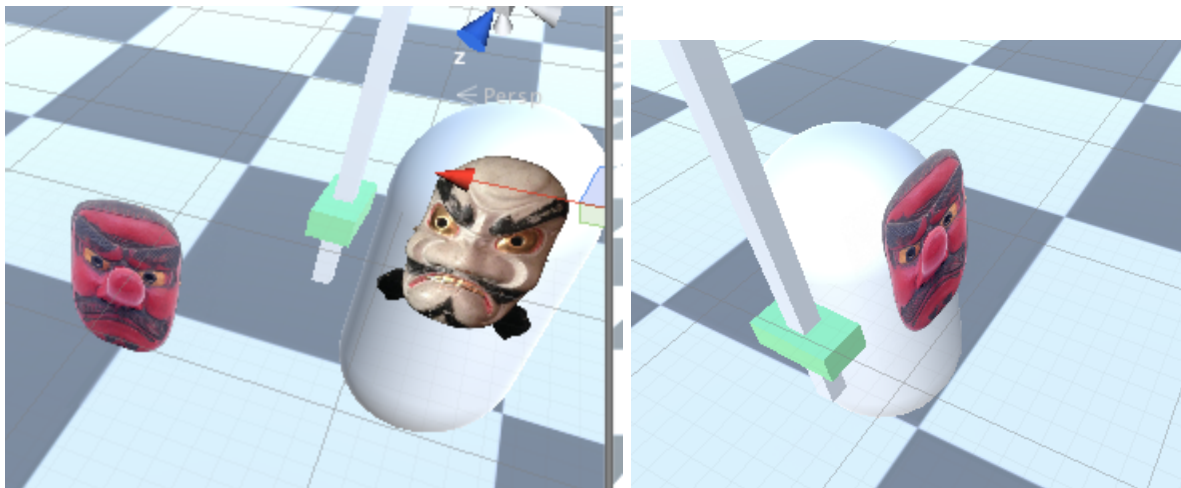
...

This should fix the error caused when you created CollectibleItem

(!) On the collectible object you created... change the value of 'Item Name' to the name of one of your masks.



Now, if a player hits an item, their equipped item in **EquipmentGroup** changes to the item specified by **CollectibleItem**, and the **CollectibleItem** object disappears.



Bonus: Here's a [link](#) to a fancier version of this project (a little messy)