DOCUMENTATION

Rio 3D
STUDIOS

a guide to create your own Multiplayer Online

learn step by step to develop your mmo!

SUMMARY

QUICK START

HOW TO DEVELOP

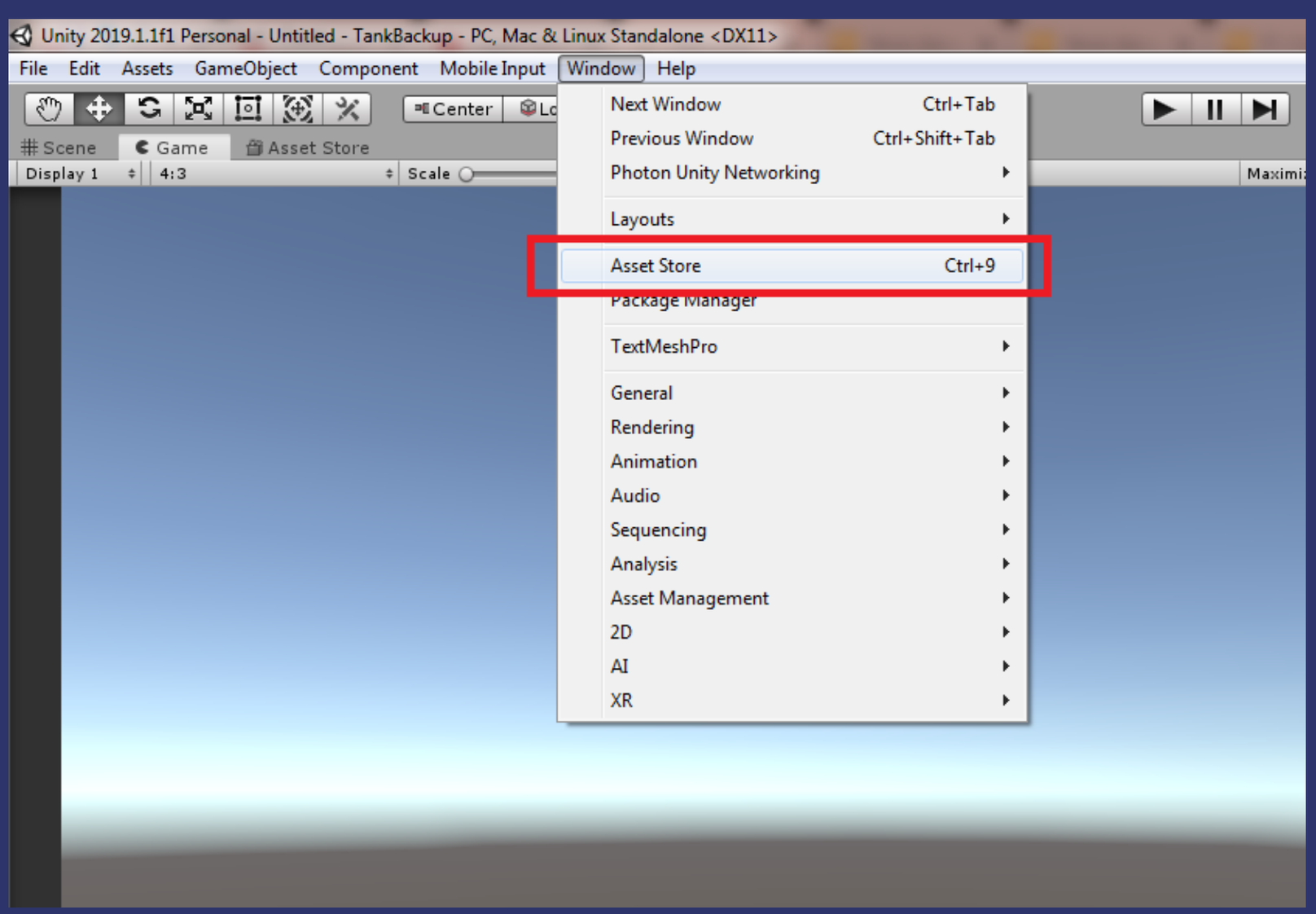
CONTACT & SUPPORT

QUICK START

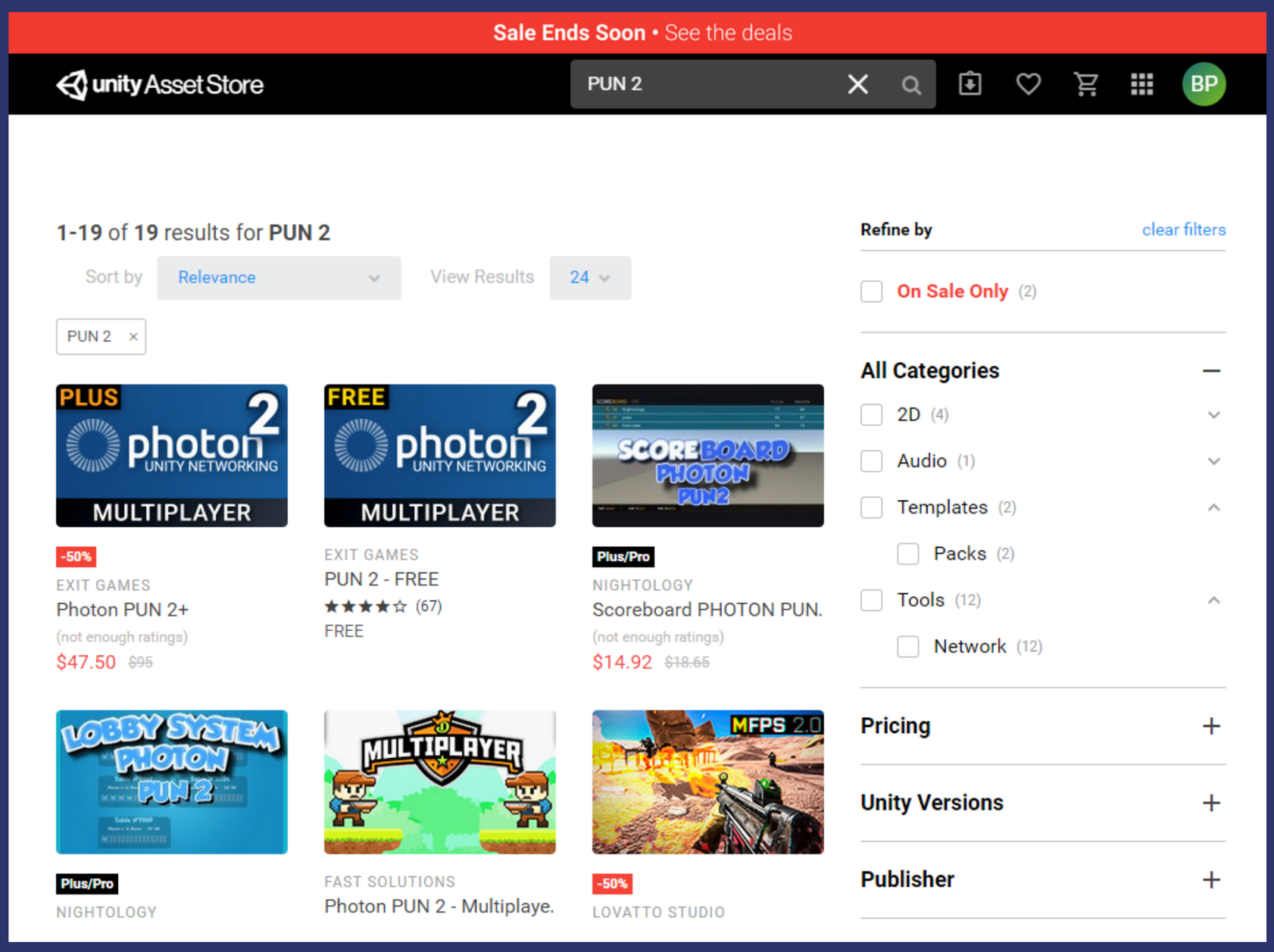
Download and Import PUN 2

To download Photon Unity Networking, follow the steps below:

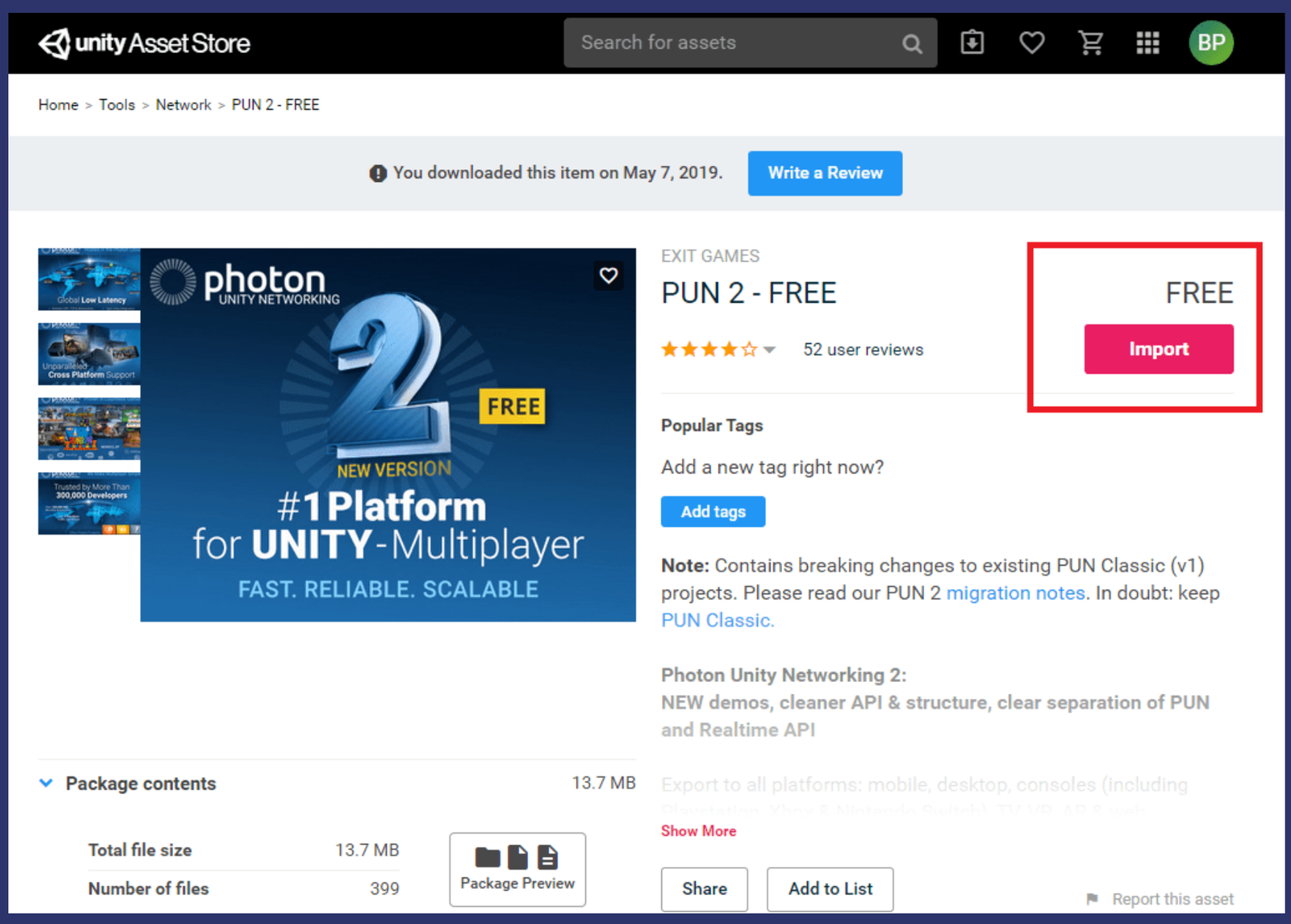
In your Unity Editor, open up Asset Store by clicking on Window tab and click Asset Store or you can do a shortcut by pressing Ctrl + 9.



Next, click on the search bar and type **PUN 2**.
Choose PUN 2 -FREE



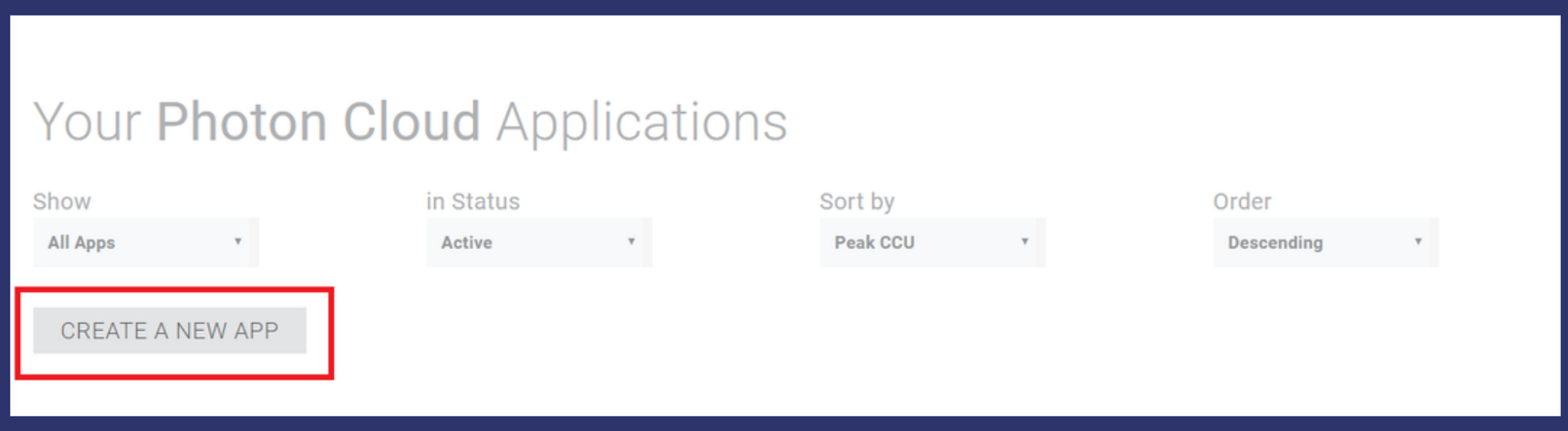
Once opened, click download and wait until you can import



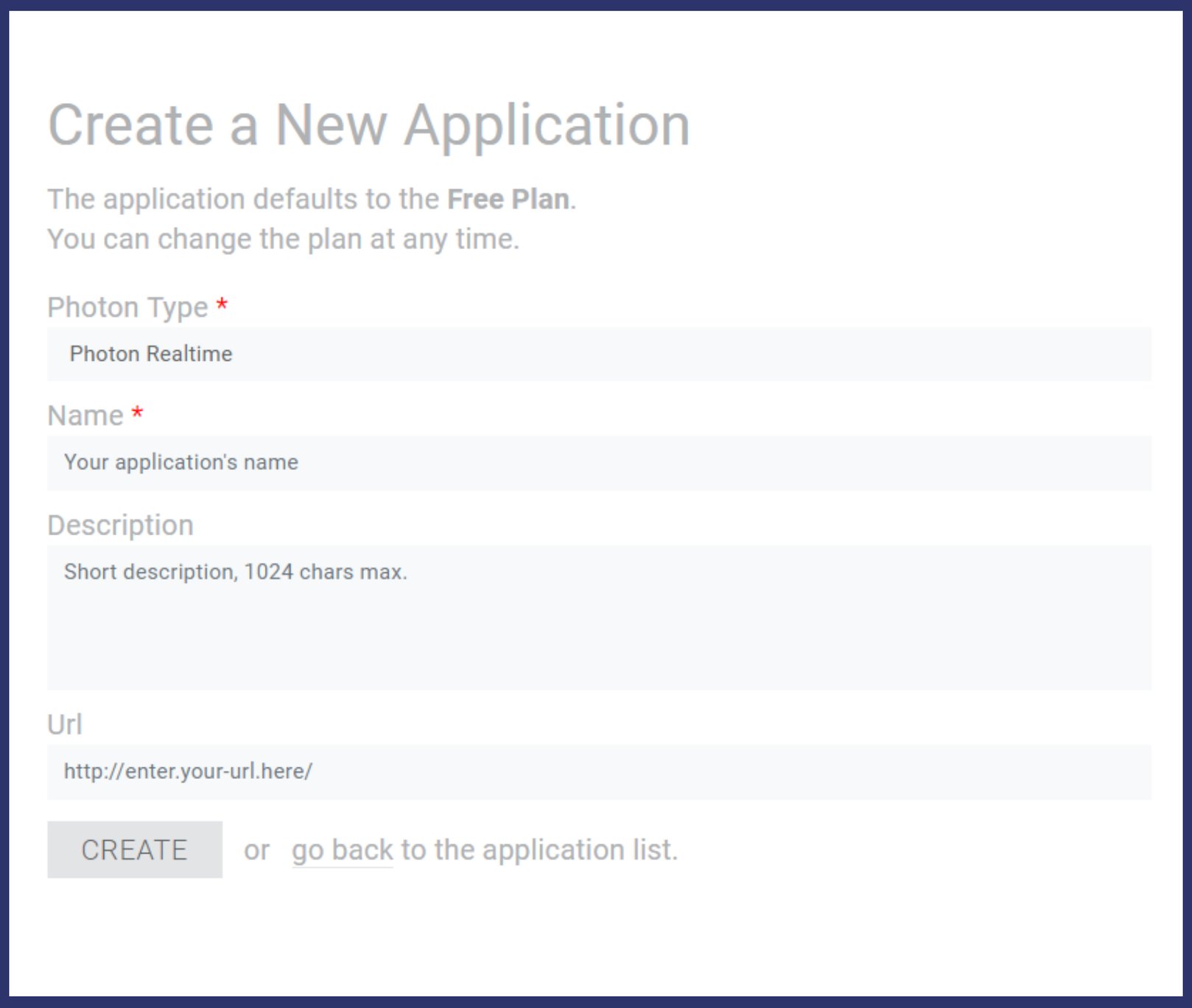
Now before we proceed, we have to create first a Photon App ID. You can do this by going to their official website at photonengine.com.

Once arrived, log in with your account or if you haven't registered yet, do so by clicking on Sign up.

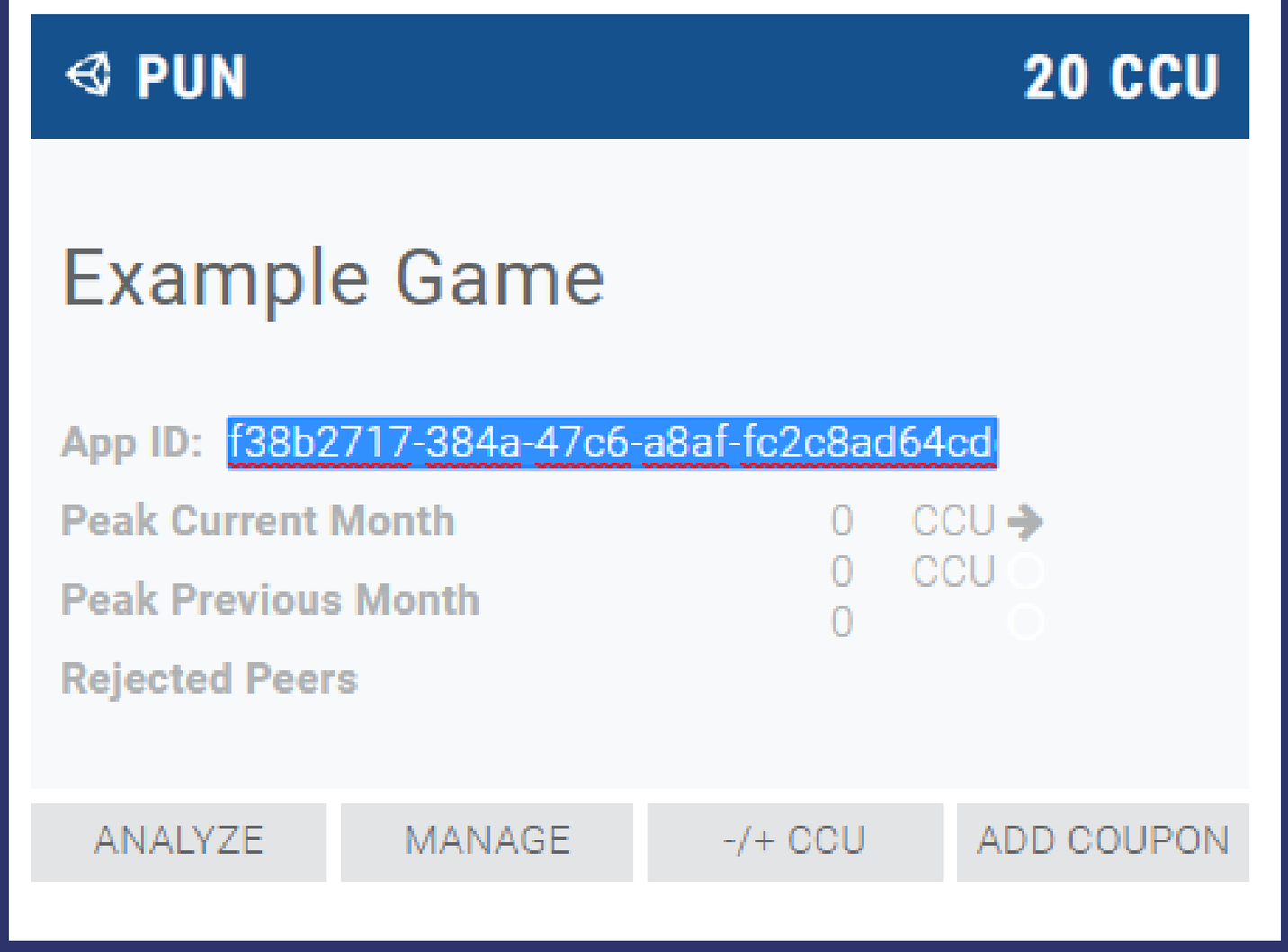
Next, once you're logged in and arrived at the dashboard. Click Create a New App like what's seen in the image below.



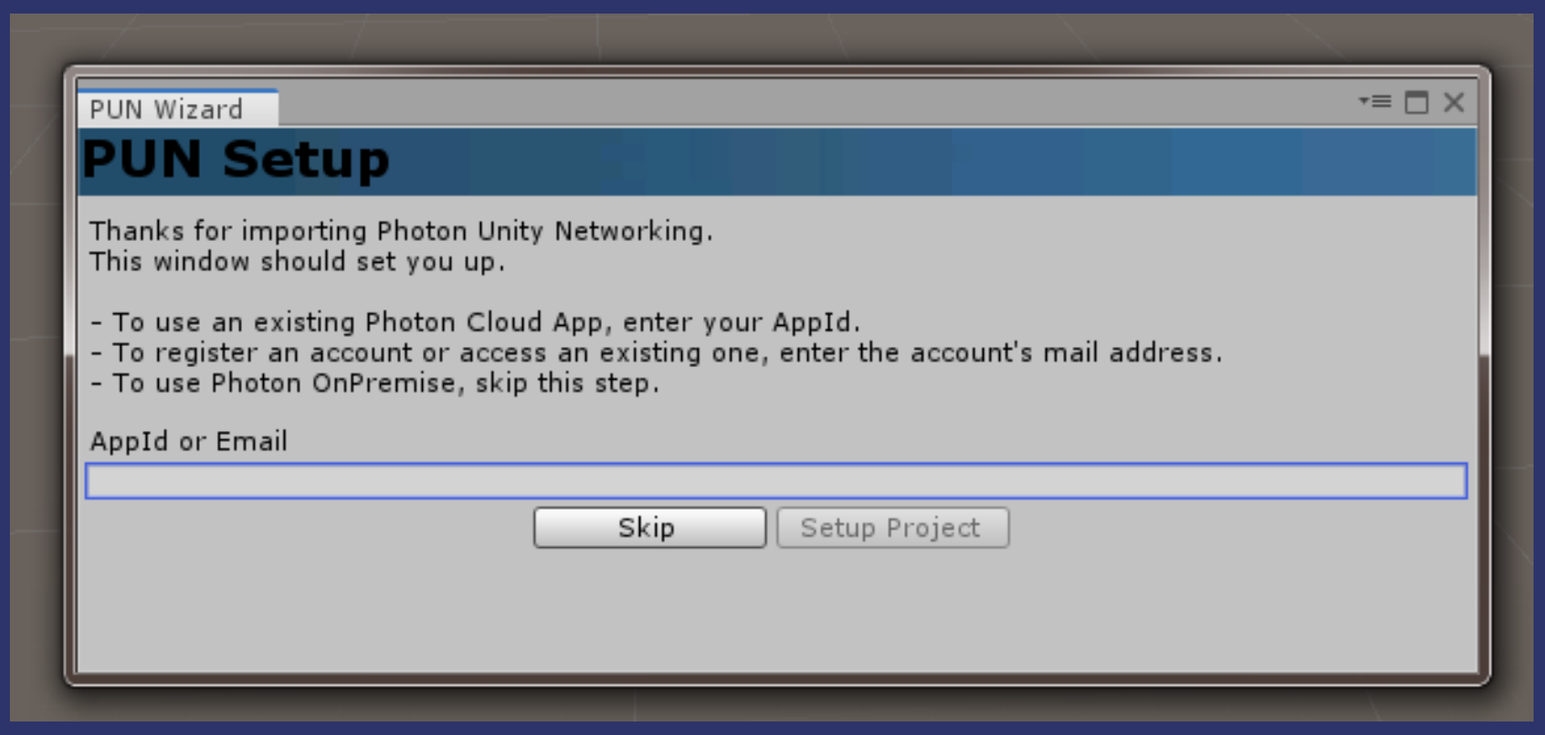
After clicking the button, fill in the form like what you're seeing in the image below:
For Photon Type: Choose Photon PUN or Photon RealtimeName: Name of your ProjectDescription: Any description you want or you can leave this emptyURL: leave this blank
Once done, click create.



Look for the panel like below and copy the App ID. Make sure it has the same name as the one you have typed in the form.

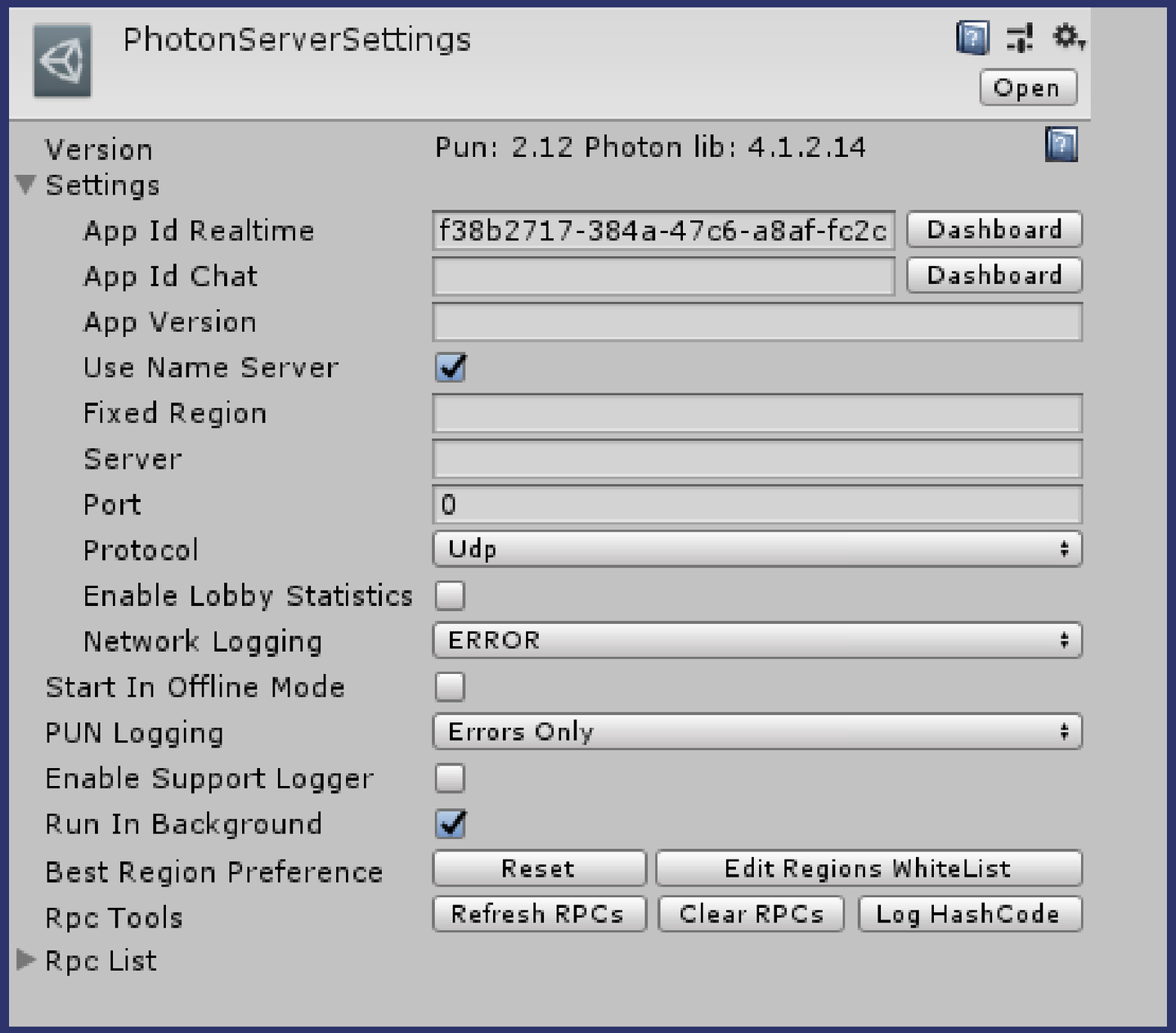


Next, go back to your Unity Editor and Paste the App Id in the window like what you're seeing in the image below:

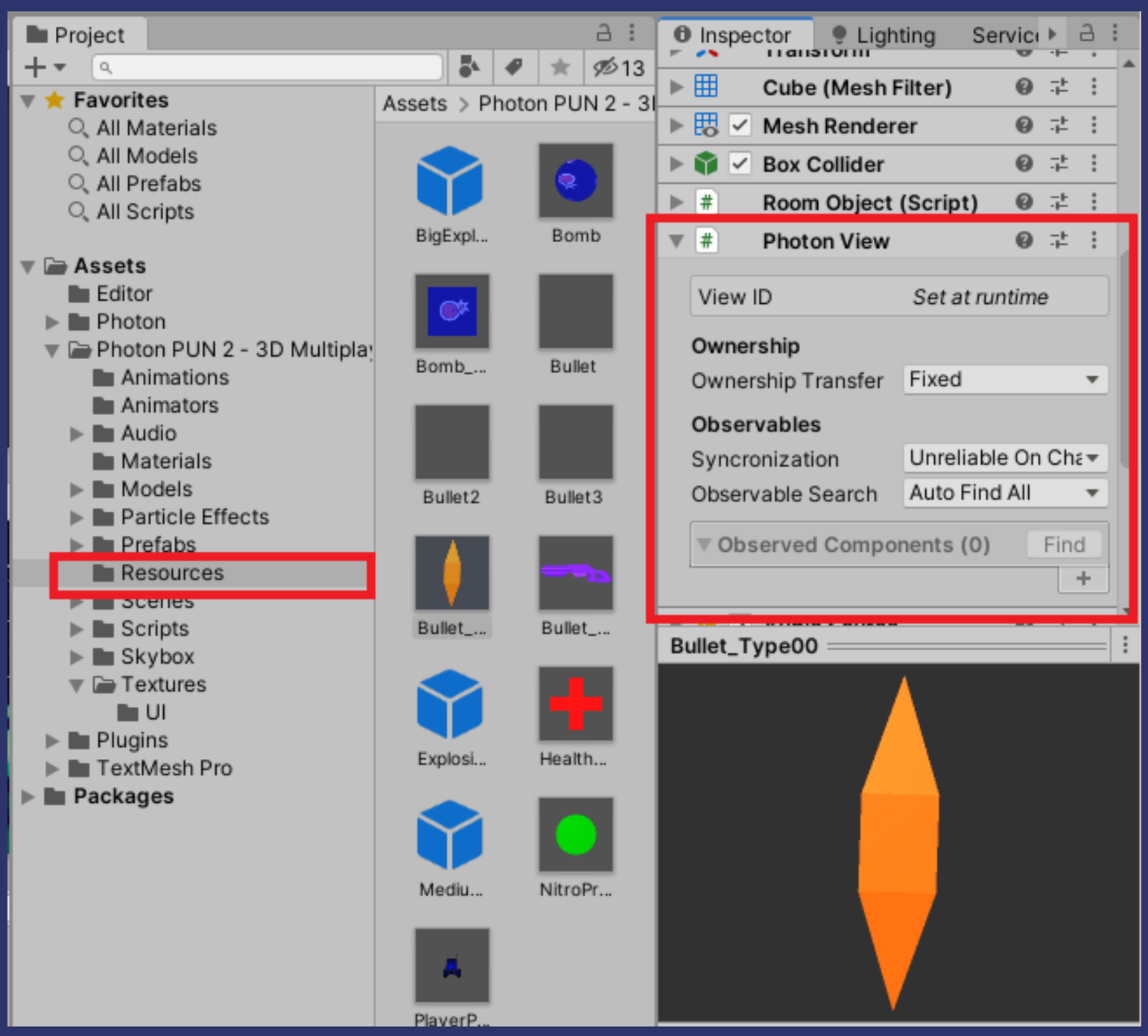


Remember, you'll only see this wizard above after you imported PUN 2 from the asset store. If somehow, you don't see this or you have accidentally skipped the wizard, you can find the server settings in folders path:

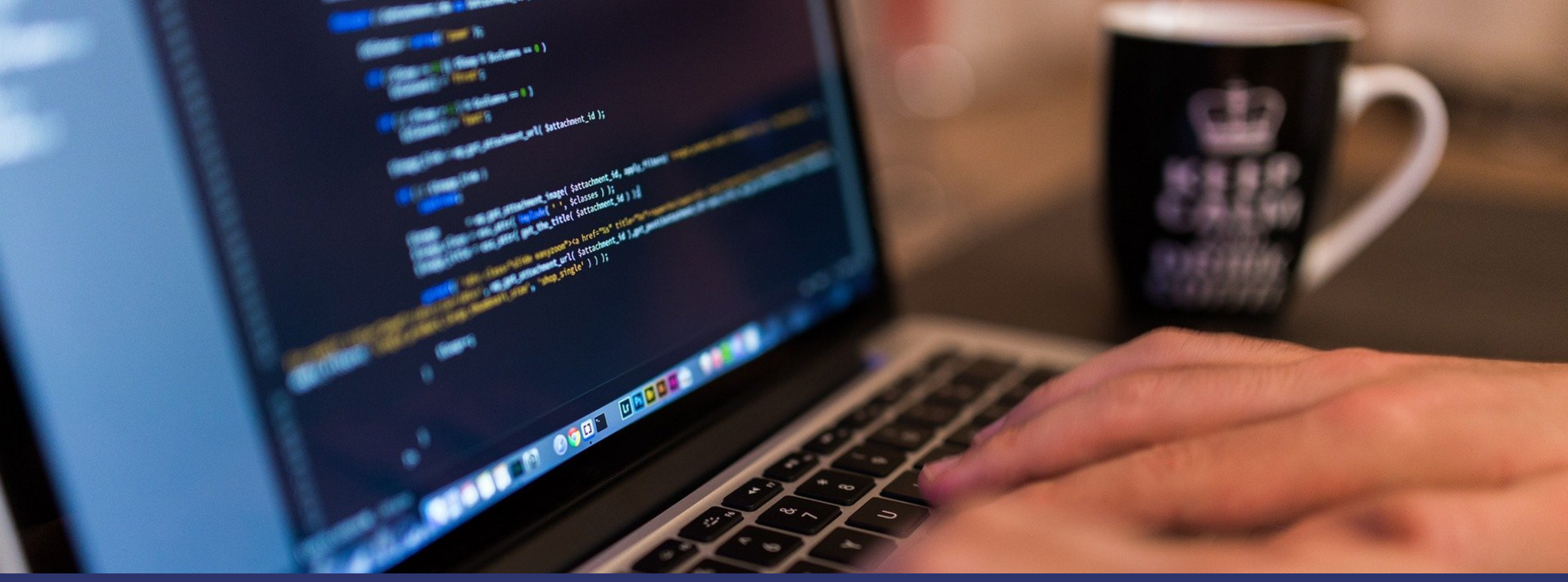
Assets > **Photon** > **PhotonUnityNetworking** > **Resources** and click PhotonServerSettings and just paste the App ID in the App Id Realtime input box like below.



Now we shoud check the **Photon View** component in all prefabs in the Resources folder:



We will update the position and rotation through State Synchronization.
That’s it! Now you have successfully connected your game with Photon Network.



How to develop

Class configuration

to use some photon callbacks we must change the class inheritance from **MonoBehaviour** to **MonoBehaviourPunCallbacks**

```
public class GameManager : MonoBehaviour {

TO

public class GameManager :  MonoBehaviourPunCallbacks
{
```

Connecting to Photon servers

```
// Use this for initialization
void Start () {


    if (!PhotonNetwork.IsConnected)
    {
        // Connect to the photon master-server.
        // We use the settings saved in PhotonServerSettings (a .asset file in this project)
        PhotonNetwork.ConnectUsingSettings();

        status = Status.Connecting;

    }


}
```

Creating room

to create a room we use the **CreateRoom** method present in the **LobbyManager** class

```
/// <summary>
///  CREATE_ROOM.
/// </summary>
public void CreateRoom()
{

    roomName = "Room " + UnityEngine.Random.Range(1000, 10000);

    RoomOptions roomOptions = new RoomOptions {MaxPlayers = maxPlayers};

    roomOptions.CustomRoomPropertiesForLobby = new string[3] { "OwnerName","Map", "isPrivateRoom"};

    roomOptions.CustomRoomProperties = new ExitGames.Client.Photon.Hashtable();
    int max_players = (int) maxPlayers;
    roomOptions.CustomRoomProperties.Add("MaxPlayers", max_players.ToString());
    roomOptions.CustomRoomProperties.Add("OwnerName", inputLogin.text);
    roomOptions.CustomRoomProperties.Add("Map", currentMap);
    roomOptions.CustomRoomProperties.Add("isPrivateRoom",isPrivateRoom.ToString());

    PhotonNetwork.CreateRoom(roomName, roomOptions, null);

    status = Status.Creating;

}
```

Join room

for this template we created a method called **JoinRoom** in **LobbyManager** class, which is called when the user when the user selects a room in **RoomListPanel**.

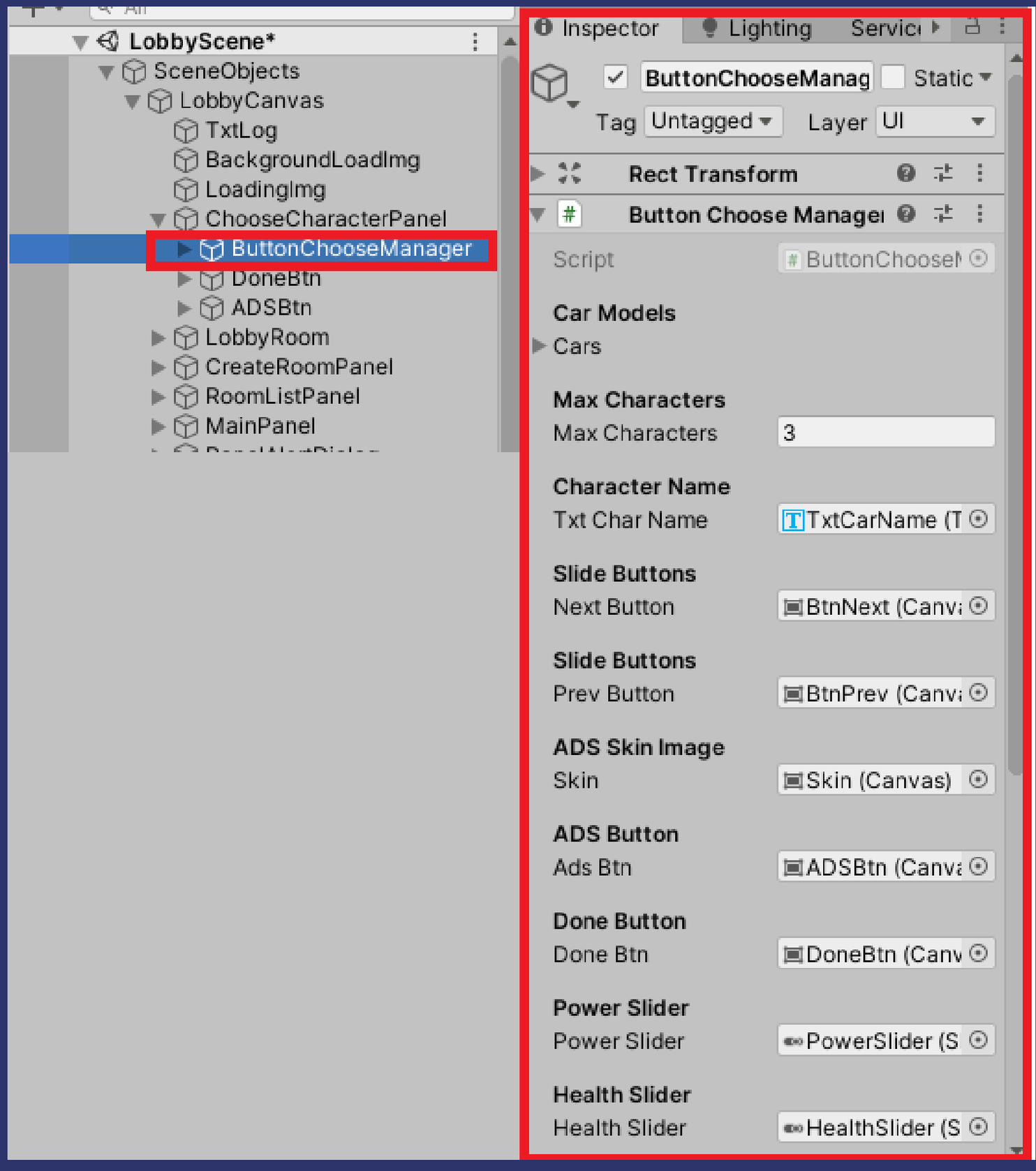
This method will attempt to connect the player to one created by a pre-existing user, as soon as the player enters the room, the **OnJoinedRoom ()** method is triggered

when the player enters the room, the **OnJoinedRoom** callback is automatically triggered, and in **Matchmaking** RPC method, the player is placed waiting for the other players in the room

```
/// </summary>
 public void JoinRoom(string _roomName){

    PhotonNetwork.JoinRoom(_roomName);

}

/// <summary>
/// method called automatically after the player has joined the room.
/// </summary>
public override void OnJoinedRoom(){

    status = Status.Rooming;

    Debug.Log ("\n joining ...\n");

    Debug.Log("RoomJoined " + PhotonNetwork.CurrentRoom.Name);

    PhotonNetwork.LocalPlayer.NickName = inputLogin.text;


    OpenScreen("lobby_room");

    SetUpRoom( PhotonNetwork.CurrentRoom.Name,PhotonNetwork.CurrentRoom.PlayerCount.ToString(),
                                        PhotonNetwork.CurrentRoom.MaxPlayers.ToString(), (string)PhotonNetw
    view.RPC("Matchmaking", RpcTarget.AllBuffered, PhotonNetwork.LocalPlayer);


}
```

Car Selection Function

In the Lobby scene, we have the **Button Choose Manager** object inside **Lobby Canvas** Game Object,. Attached to the object we have the ButtonChooseManager script which is responsible for managing the choice of cars and storing the car chosen by the user.



through the slide buttons the user can choose the desired car. Exceptionally for the mobile version of the game there is a lock screen, where the user can only choose the desired car if they watch the ADS video. An interesting way of monetizing the game.

Spawn Player Function

In the **Game scene**, we instantiate the object according to the car chosen in the previous **Lobby Scene**.

To instantiate the player avatar across the network, after joining a room, we use **PhotonNetwork.Instantiate** in **Game Manager** class.

```
/// <summary>
/// Spawns the player.
/// </summary>
public void SpawnPlayer()
{
    //makes the draw of a point for the player to be spawn
    int index = UnityEngine.Random.Range (0, map.GetComponent<MapManager>().spawnPoints.Length);

    // take a look in PlayerManager.cs script
    PlayerManager newPlayer;

    // newPlayer = GameObject.Instantiate( local player avatar or model, spawn position, spawn rotation)
    newPlayer = PhotonNetwork.Instantiate(playersPrefab.name,
    map.GetComponent<MapManager>().spawnPoints[index].position,
        Quaternion.identity).GetComponent<PlayerManager> ();


    networkPlayers [newPlayer.view.ViewID.ToString()] = newPlayer;

    Debug.Log("player instantiated");

    ExitGames.Client.Photon.Hashtable PlayerProperties = new ExitGames.Client.Photon.Hashtable();
    PlayerProperties.Add("Id", newPlayer.view.ViewID.ToString());
    PhotonNetwork.LocalPlayer.SetCustomProperties(PlayerProperties);


    newPlayer.view.RPC("SetUpPlayer", RpcTarget.All,newPlayer.view.ViewID.ToString(),
        PhotonNetwork.LocalPlayer.NickName,(int)PhotonNetwork.LocalPlayer.CustomProperties["currentCar"]);
```

Then in **Player Manager** class we use **RPC** (Remote Procedure Calls) to setup car atributes

```
/// <summary>
///  set the new player for the players who are already in the room
/// </summary>
/// <param name="_id">player's id</param>
/// <param name="_name"> player's name</param>
/// <param name="_car">car index</param>
[PunRPC]  public void SetUpPlayer(string _id, string _name, int _car)
{
    if(view.ViewID.ToString().Equals(_id)  )
    {

    GameManager gameManager = GameObject.Find("GameManager").GetComponent<GameManager>();
    id = _id;
    name = _name;
    car_index = _car;
    SetUpCar( _car);
    Set3DName(name);
    //puts the new player on the list
    gameManager.networkPlayers [view.ViewID.ToString()] = this;

    }

}
```

PhotonView Component

Okay, now that your players can create and access the rooms, it's time to see each other making their moves. For this to be possible, it is necessary to add the PhotonView component. The PhotonView component has the following editable properties:

- **View ID**: It is the unique identifier for this component. By default it already assigns an ID, so to avoid conflicts, it is better to leave the default value;

- **Observer**: Transform that can be synchronized;

- **Observe option**: The type of synchronization that will be used by the component. There are currently 4 types of synchronization. Off Observe is not synchronized, Reliable Delta Compressed synchronizes only when there are changes. Unreliable is always synchronized, and finally, Unreliable On Changem, which uses a little of the previous mode. It sends a last update to everyone when there are no more changes, ensuring that everyone receives it;

State Synchronization

There are two methods for network communication. The first is State Synchronization and the other is Remote Procedure Calls(RPC's).

State Synchronization constantly updates data via the network. It is ideal for updating data such as moving the player.

With **RPC**, you can mark your methods to be callable by any client in a room. RPCs that are most useful for data that doesn't constantly change. in this asset, we have implemented RPC calls in several cases, among them is the DisplayEffects method, used to synchronize all effects from player.

```
//display effects throuw network
GetComponentInParent<PlayerManager>().view.RPC("DisplayEffects", RpcTarget.All);
```

```
/// <summary>
/// method called by SpawnBullet method on gun class
/// </summary>
[PunRPC] public void DisplayEffects()
{

    StartCoroutine ("ShowEffects");

}
```

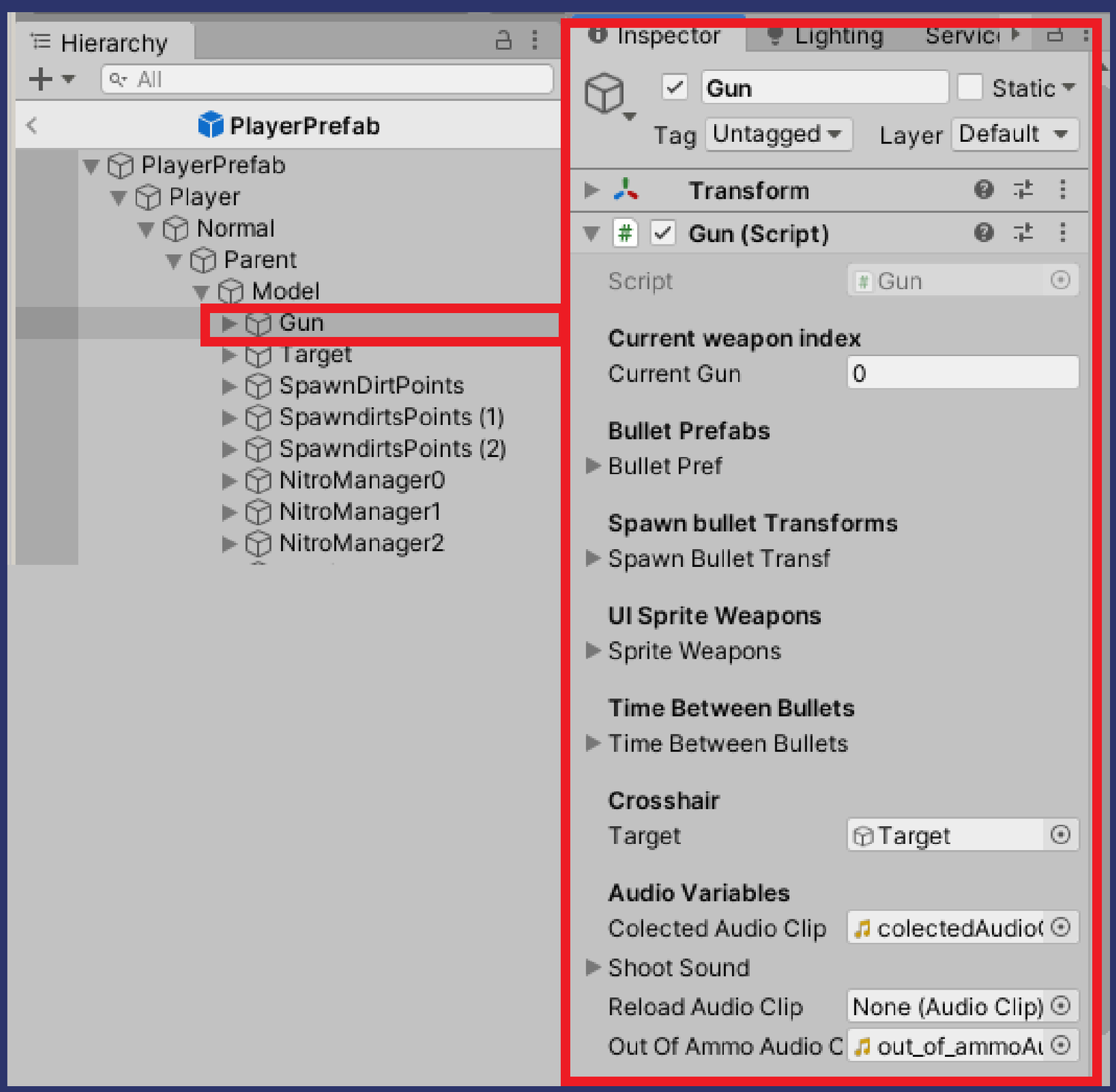
RPC call to activate the car's nitro on all online clones of the current player:

```
// if the player presses the "B" key on the keyboard, it activates nitro if there is a load
if (Input.GetKeyUp (KeyCode.B) || enableNitro)
{

    view.RPC("ActvateNitro", RpcTarget.All);

}
```

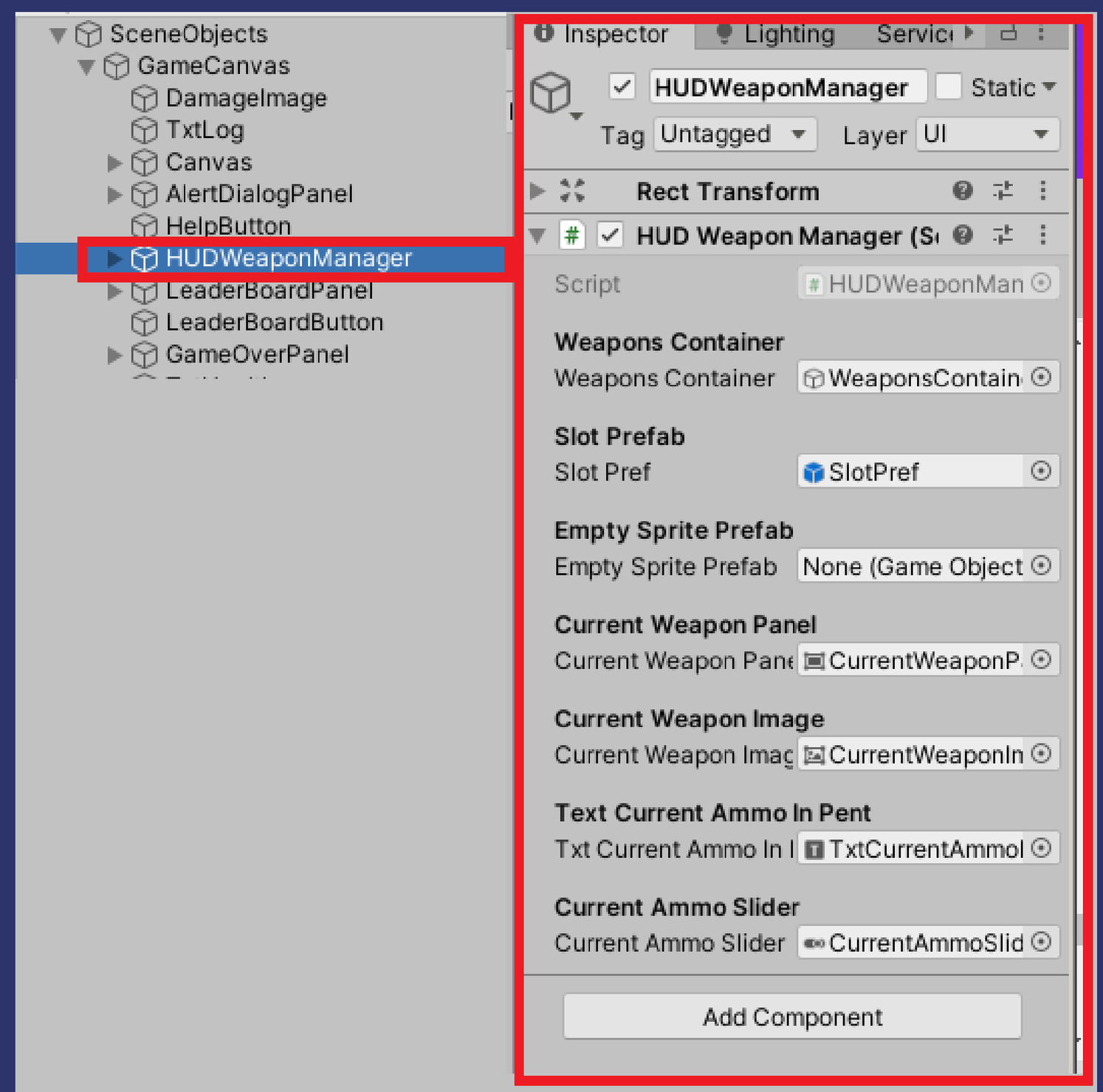
```
/// <summary>
/// method for activate player nitro
/// </summary>
[PunRPC] public void ActvateNitro()
{
  SpawnNitro();
}
```

Player Shooter Function

to control the player's weapon, we have the Gun object and attached to this object we have the **Gun** script. the Gun object can be found inside the Player Prefab Object

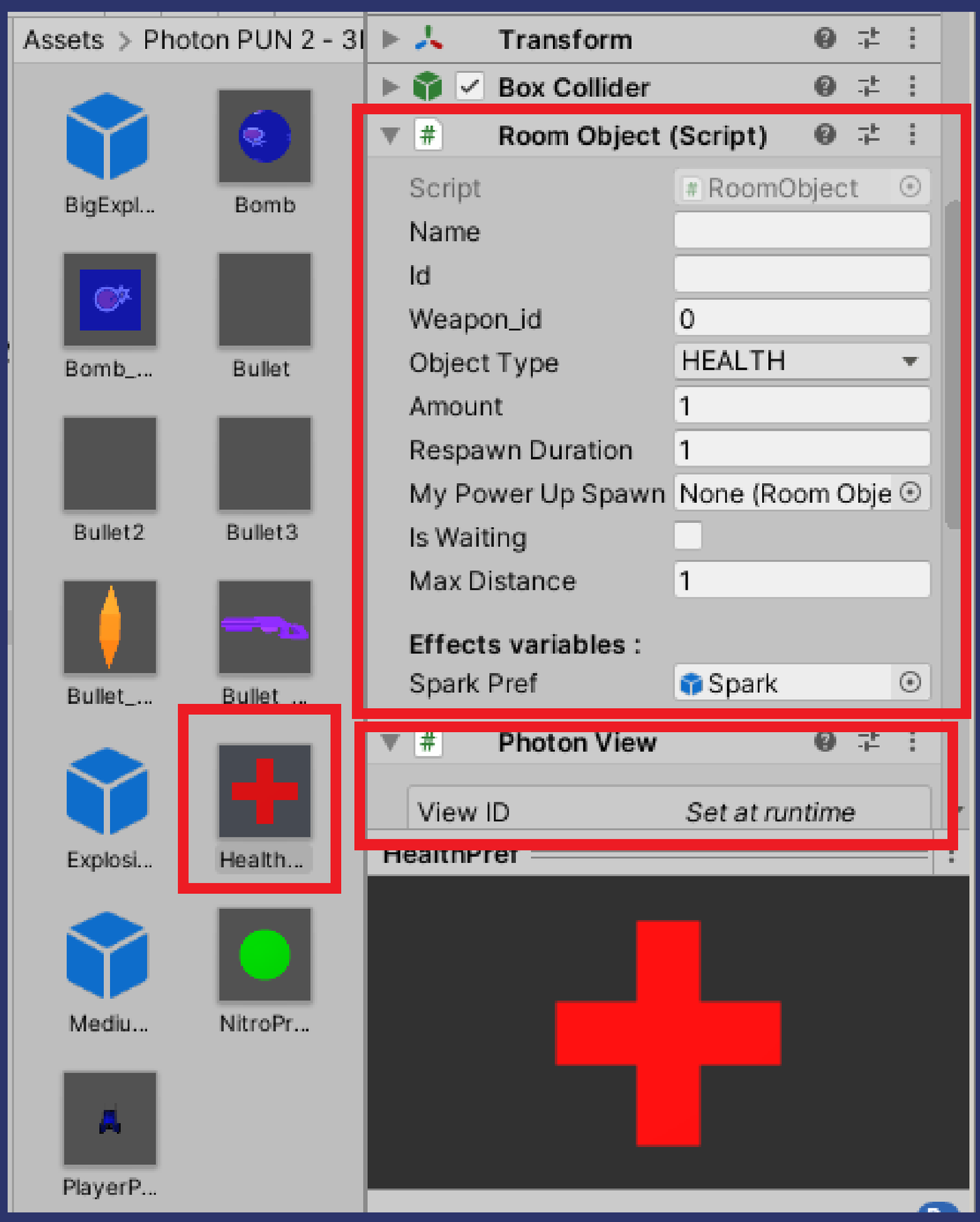


the exchange of weapons and control of the weapons UI present on the screen is performed by the **HUDWeaponManager** script, attached to the object of the same name.

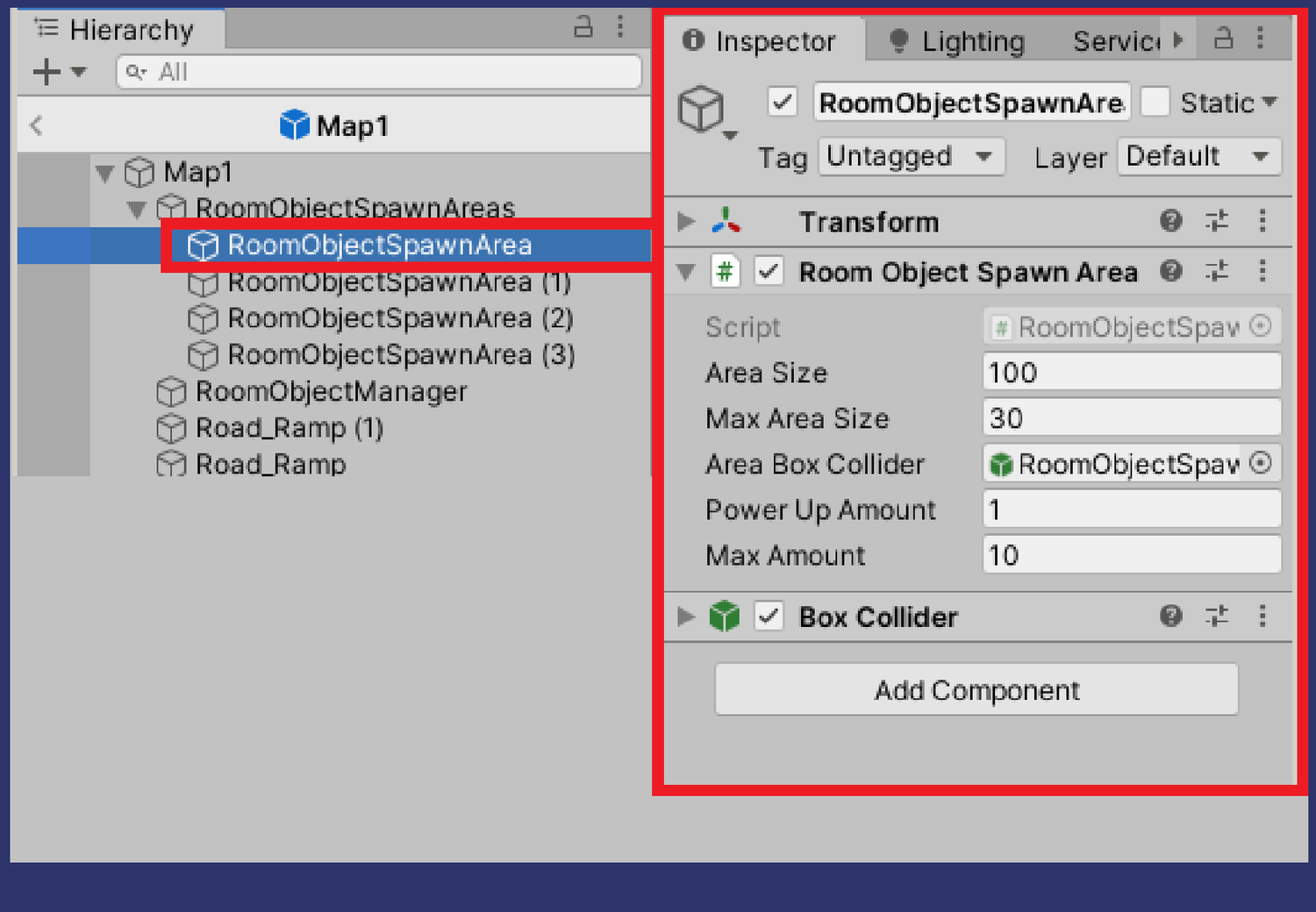


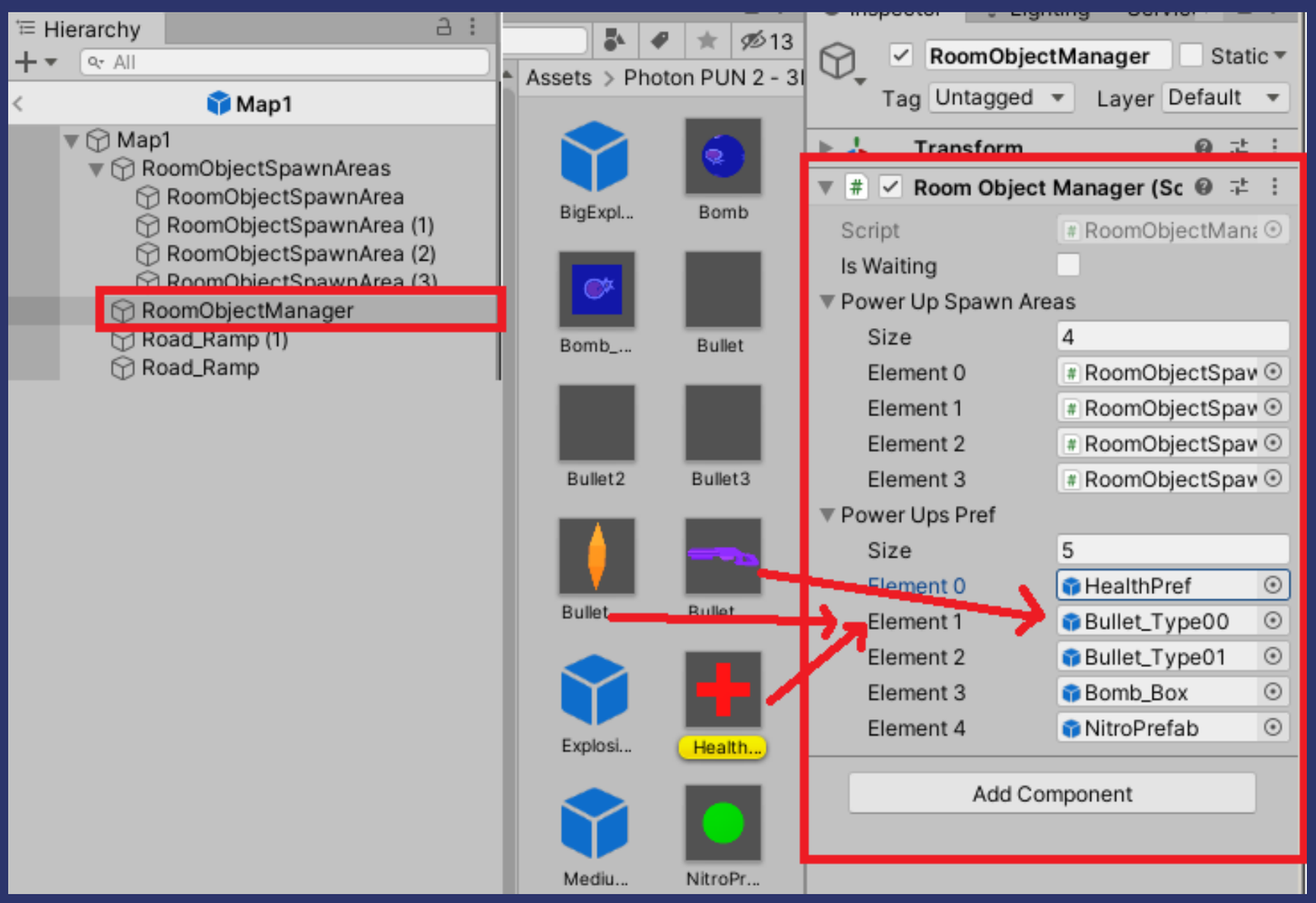
Power UP and Weapon Loot System

In the Resources folder are all Weapon and PowerUp Loots. Each of these objects contains a script called **RoomObject** and a **PhotonView** component. The **RoomObject** script manages the power Up or weapon loot



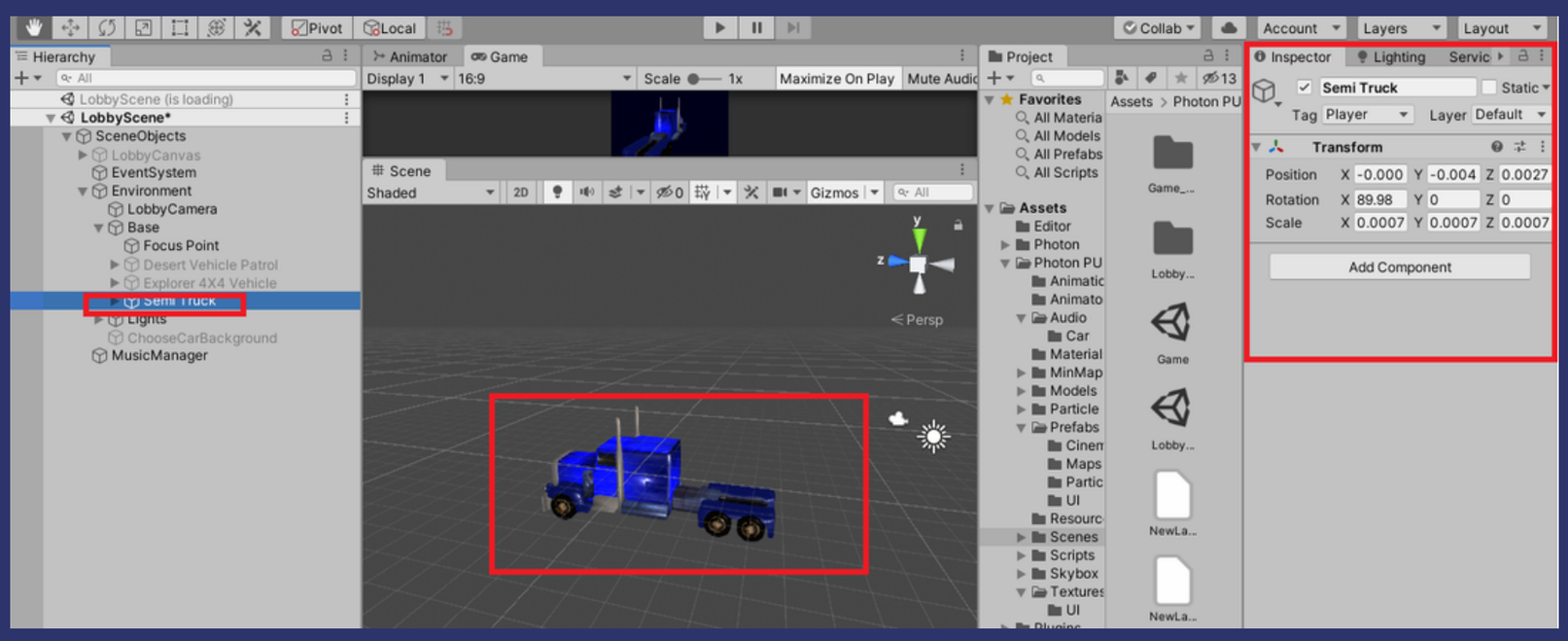
To instantiate these objects in the game arena, we use the **Room Object Manager** and **Room Object Spawn Area** scripts. This script can be found attached to the object of the same name within the prefabs **Map1**, **Map2** or **Map3**



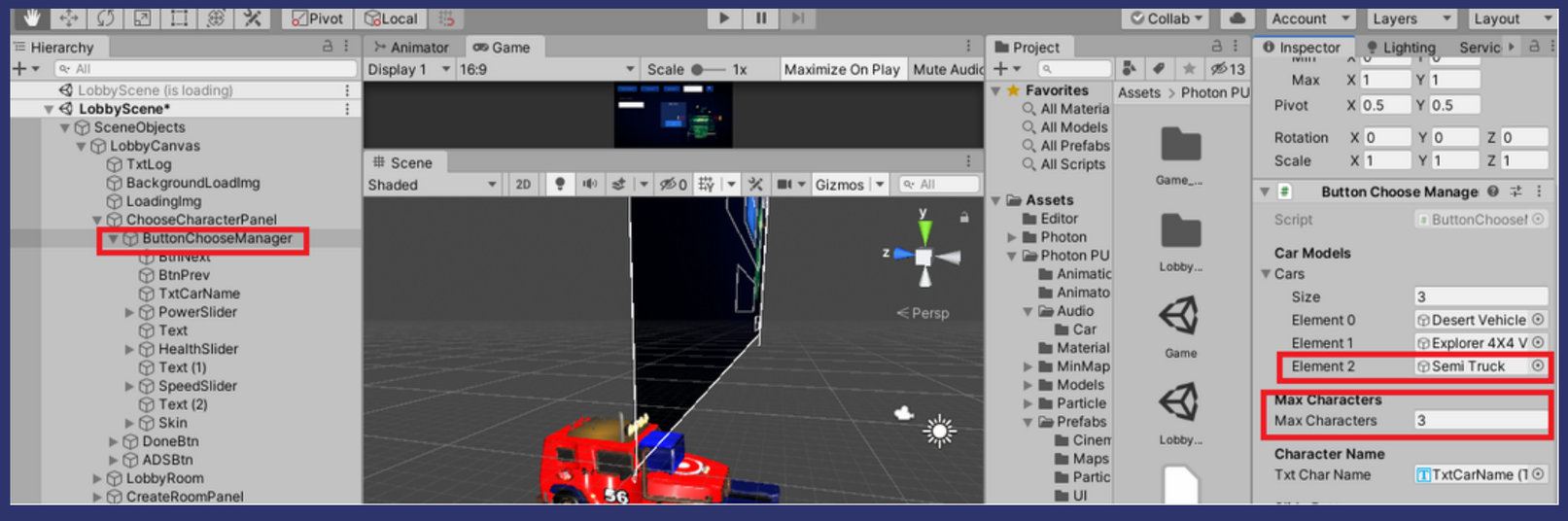

Hierarchy
All
Map1
Map1
RoomObjectSpawnAreas
RoomObjectSpawnArea
RoomObjectSpawnArea (1)
RoomObjectSpawnArea (2)
RoomObjectManager
Road_Ramp (1)
Road_Ramp
Assets > Photon PUN 2 - 3
BigExpl...
Bomb
Bomb_...
Bullet
Bullet2
Bullet3
Bullet
Explosi...
Health...
Mediu...
NitroPr...
RoomObjectManager
Static
Tag Untagged
Layer Default
Transform
Room Object Manager (Sc
Script
Is Waiting
Power Up Spawn Areas
Size 4
Element 0
Element 1
Element 2
Element 3
Power Ups Pref
Size 5
Element 0 HealthPref
Element 1 Bullet_Type00
Element 2 Bullet_Type01
Element 3 Bomb_Box
Element 4 NitroPrefab
Add Component

How To Add New Vehicle

to add a new vehicle model, we must first go to the Lobby Manager scene and insert the new model into Base Game Object.



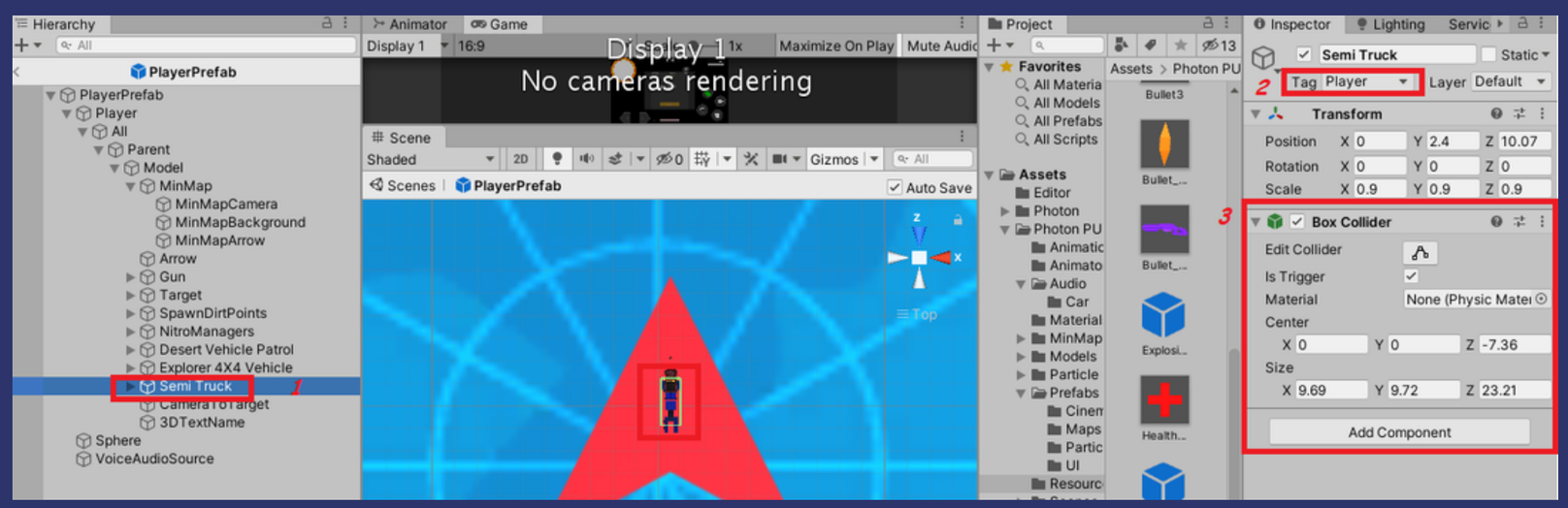
Then, go to Button Choose Manager Game Object and in the Button Choose Manager script find the vehicle vector "Car Models". In Car Models add the new vehicle. also change the total number of vehicles in the Max Characters variable



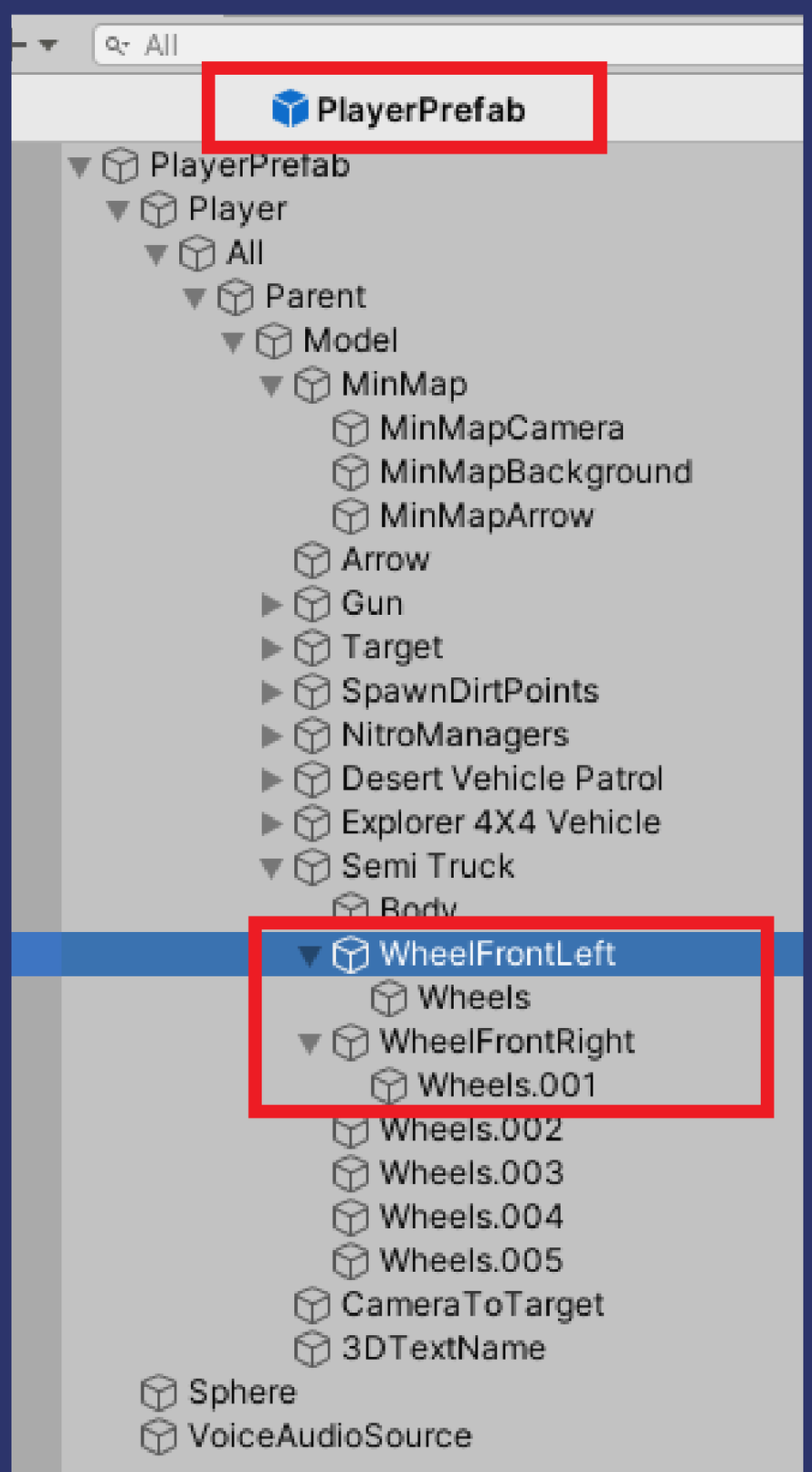
Go to the button choose Manager script and in the SetCarSkills () method, add the vehicle skils as shown in the image below:

```
/// <summary>
/// sets the skill of each character
/// </summary>
public void SetCarSkills()
{
    for(int i =0;i<=maxCharacters;i++)
    {
        CarSkills skill = new CarSkills ();
        skill.id = i;

        if(i==0)
        {
            skill.name = "Desert Vehicle Patrol";
            skill.power = 40;
            skill.health = 50;
            skill.speed = 90;
        }
        if(i==1)
        {
            skill.name = "Explorer 4X4 Vehicle";
            skill.power = 70;
            skill.health = 60;
            skill.speed = 50;
        }
        if(i==2)
        {
            skill.name = "Semi Truck";
            skill.power = 90;
            skill.health = 70;
            skill.speed = 40;
        }
```

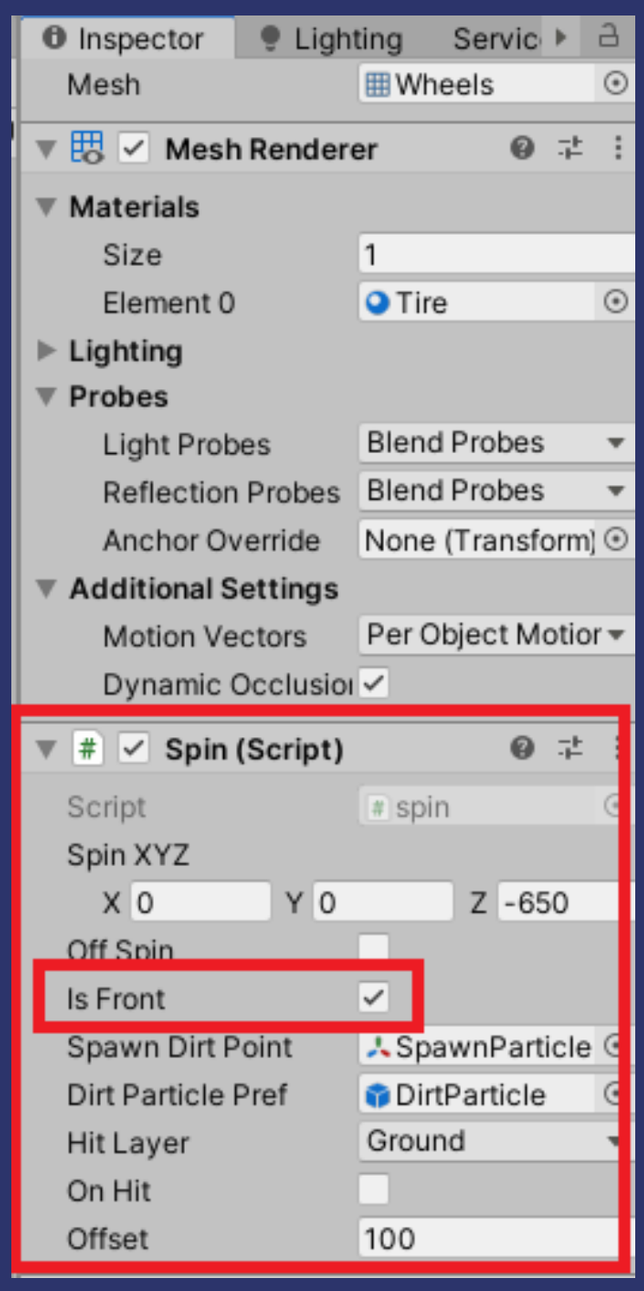
Now we will add the model in the player's prefab. go to the Resources folder and open PlayerPrefab. Add the new model inside the child object "Model". Add a box collider component and mark it as a trigger. Also add the tag "Player" to the new model.



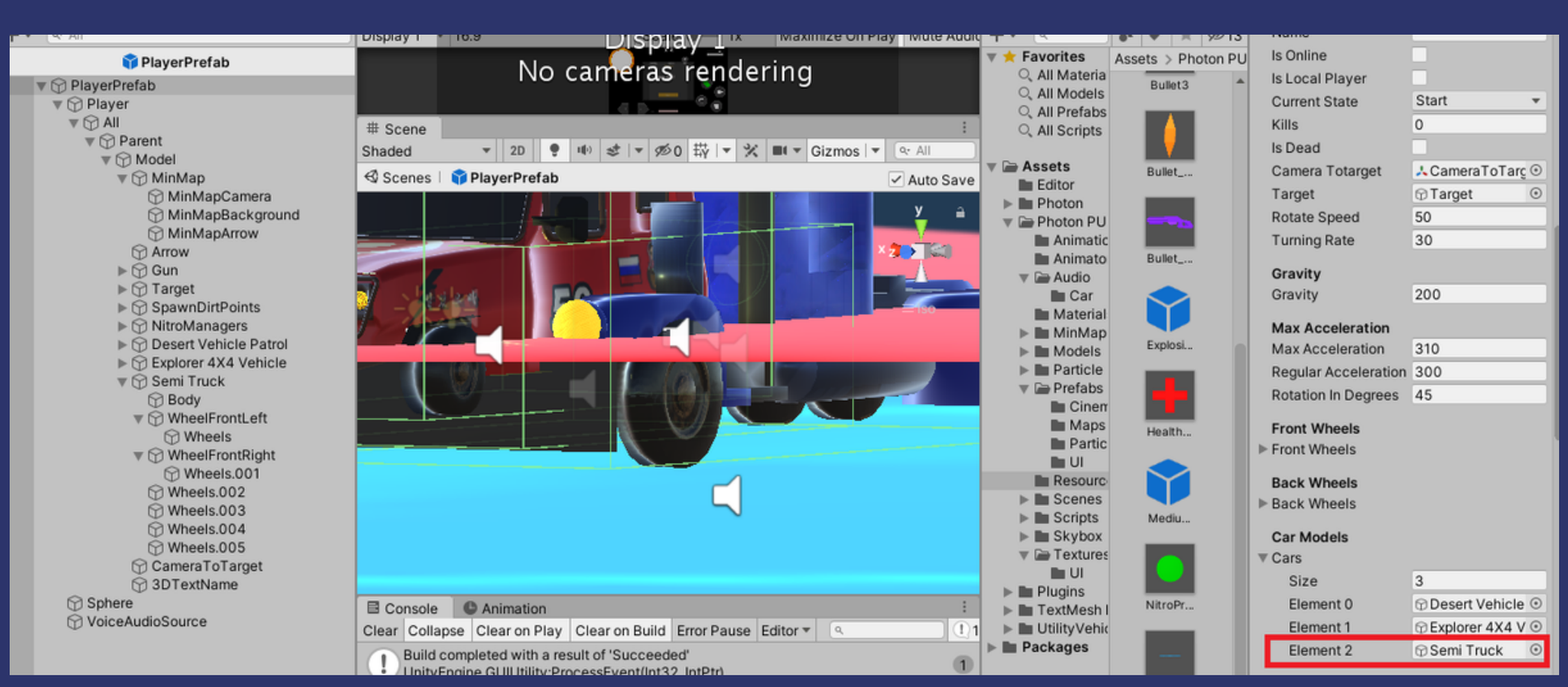
within the model, create two new empty game objects called WheelFrontLeft and WheelFrontRight respectively. copy the front wheel Tranform component for each empty game object, and add the vehicle's front wheels to each game object.



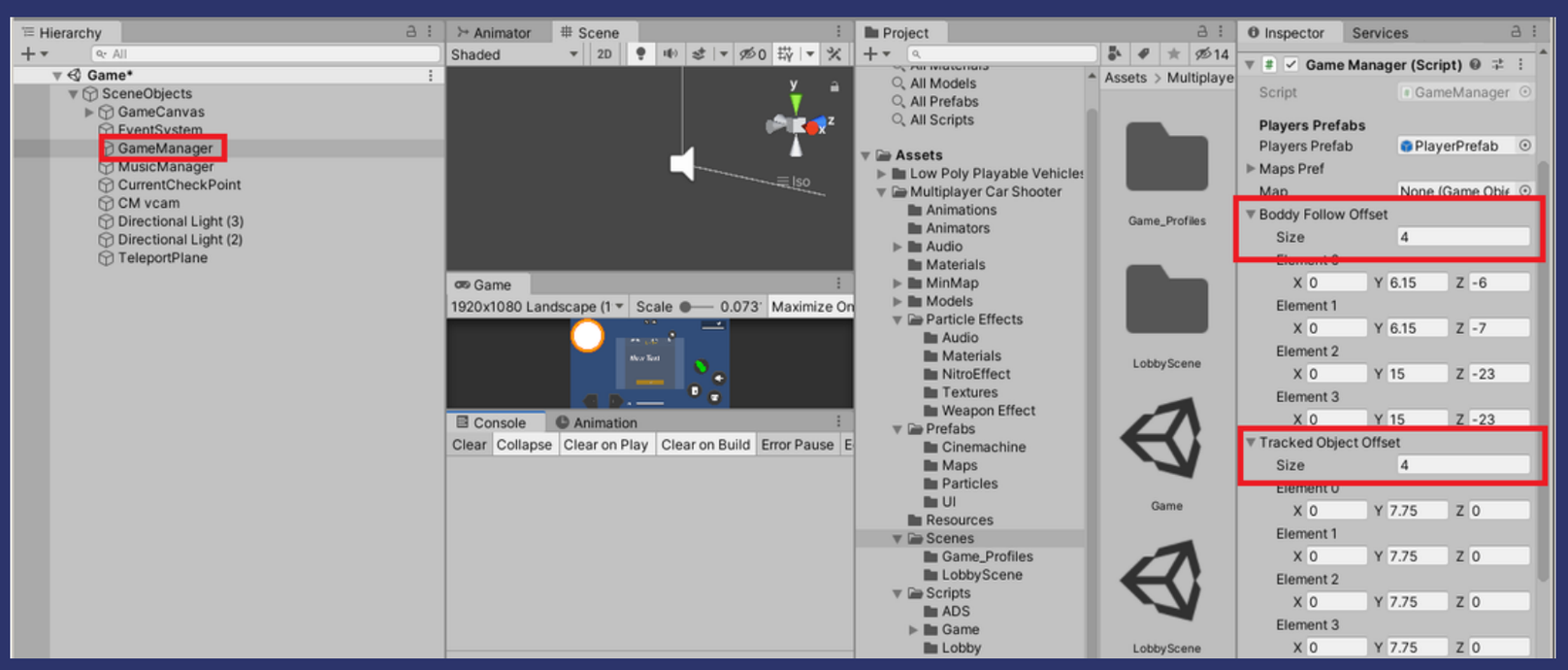
On each front wheel, add a Spin script and check the "IsFront" attribute. Add the Spin script to other wheels.



In Player Prefab go to the Player Manager script component, and in the Car Models attribute, add the new vehicle model. In the NitroManagers attribute, add one more element to the vector



Return to the game scene, and in GameManager object select the attributes "boddyFollowOffset" and "trackedObjectOffset" and add one more element to the vectors corresponding to the new vehicle. We need to do this to adjust the cineMachine camera to the new vehicle.



hands-on!

in this asset you will find the most varied photon functions examples for you to start learning how to develop your own application!

Contact us

rio3dstudios@gmail.com

rio3dstudios.com