**Technische Hochschule Ingolstadt**

# Anleitung zur Erstellung einer Multiuser-VR Anwendung in Unity mit Photon Engine Networking


## Inhaltsverzeichnis

# 1. Einrichtung eines Unity VR Projekts für eine HTC Vive Focus 3 (Android Plattform)

1.1. Erstellen eines neuen Unity 3D-Projekts mit einer Unity-Version höher als 2018.4.17. In der vorliegenden Anleitung ist die Unity Version 2021.2.9f1 verwendet worden.

1.2. Bearbeiten der „*manifest.json*“-Datei im „Packages“-Ordner des Unity Projekts.

➔ Einfügen von ScopedRegistries für die Freischaltung des Wave SDK (vgl. Anleitung: https://hub.vive.com/storage/docs/en-us/UnityXR/UnityXRGettingStart.html).

```
    "scopedRegistries": [
        {
            "name": "VIVE",
            "url": "https://npm-registry.vive.com",
            "scopes": [
                "com.htc.upm"
            ]
        }
]
```

1.3. Anschließend kann im Unity Package Manager das Custom-Asset „Vive Wave XR Plugin“ installiert werden → „My Registries“ → „Vive Wave XR Plugin“ (vgl. Abbildung 1).

1.4. Zusätzlich sollten im Unity Package Manager unter „My Registries“ das Vive Input Utility Asset sowie dessen Beispielprojekt „3D Drag“ importiert werden (vgl. Abbildung 1).

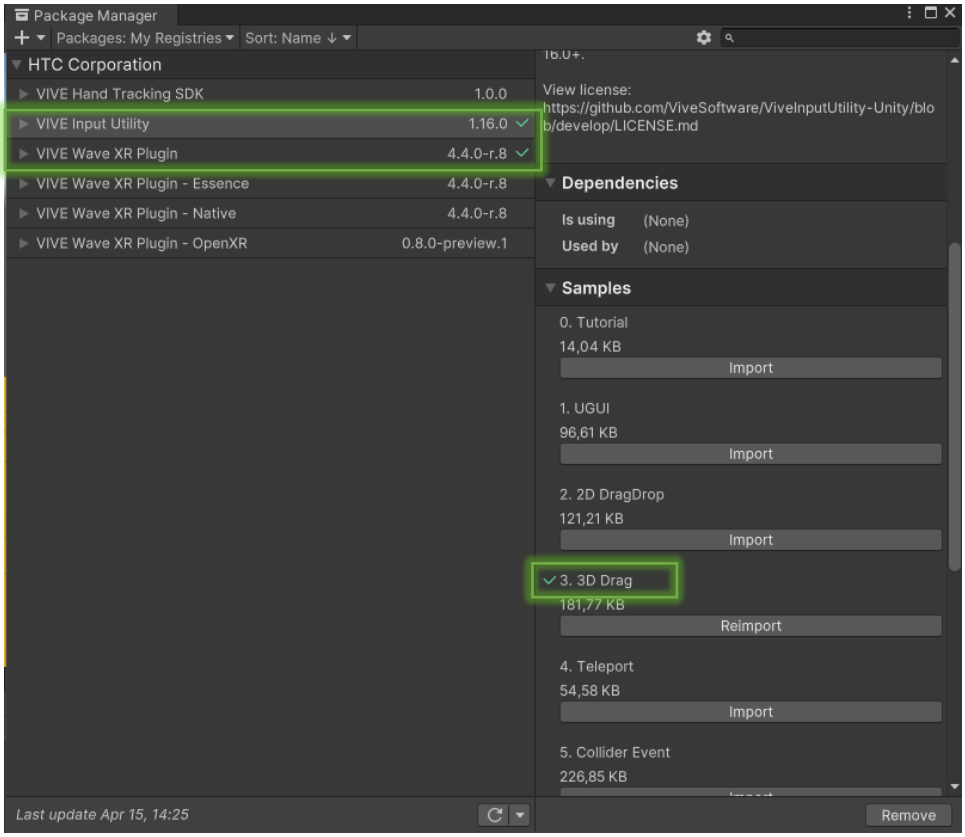


**Abbildung 1: Import des VIVE Wave XR Plugins und Vive Wave Input Utility mit dem Beispielprojekt 3D Drag**

1.5. In den Build Settings kann anschließend auf die Plattform „Android“ gewechselt werden und die Beispielszene „3D Drag“ geöffnet werden. Weiterhin sollte in den Project Settings das XR Plug-in Management auf WaveXR umgestellt werden, vgl. Abbildung 2.

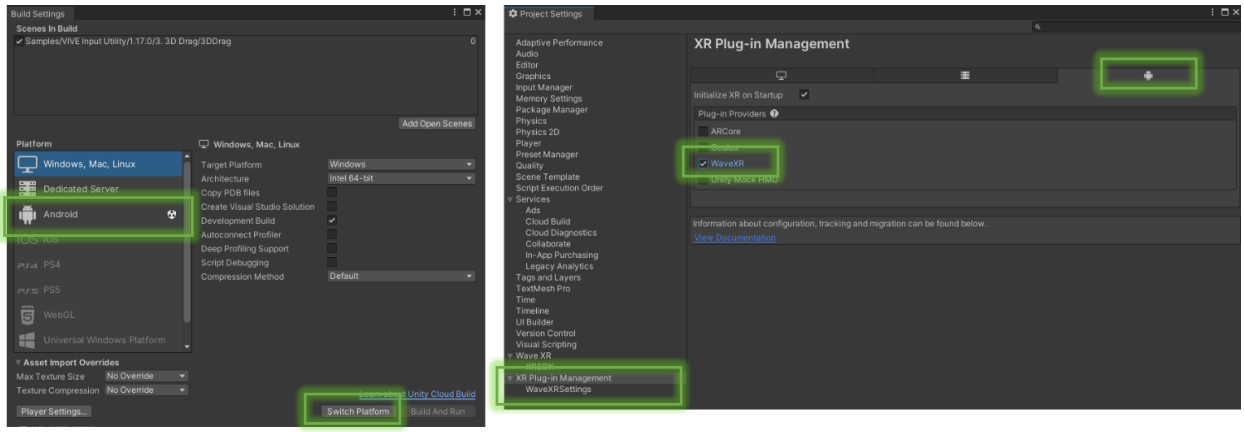


**Abbildung 2: Wechseln auf Android-Plattform in den Build Settings und Aktivieren des WaveXR Plugins in den Project Settings**

1.6. Die Beispielszene „3D Drag“ kann in Unity geladen und für einen Testbuild (als Android Build/.APK-Datei) gebildet werden. Anschließend kann die Android-Datei auf die HTC Vive Focus 3 gepatcht werden (Häkchen für Development setzen sowie USB-Debugging auf der HTC Vive Focus 3 aktivieren).

## 2. Photon Server-Einstellungen für die VR Multiuser-Anwendung

2.1. Erstellen einer neuen Szene im bereits bestehenden Unity Projekt (Unity-Version höher als 2018.4.17)

2.2. Importieren des kostenlosen PUN2 - Assets (vgl. Abbildung 3 und 4) über den Paket Manager in das Unity Projekt.

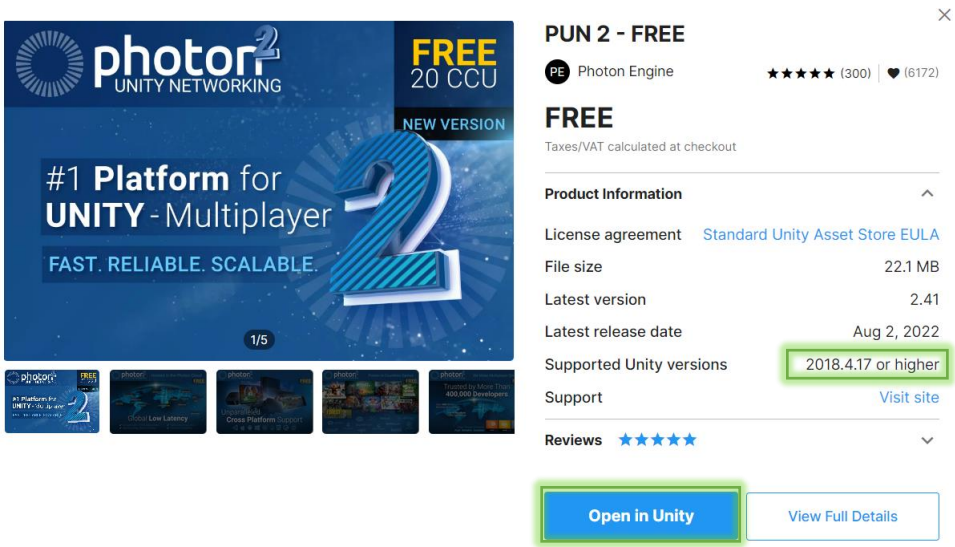


**Abbildung 3: PUN2 - Free Asset im Unity Asset Store**


**Abbildung 4: Importieren des „PUN2 - Free Assets“ in Unity über den Packet Manager**

2.3. Nach dem Import öffnet sich ein PUN Setup-Window (vgl. Abbildung 6). In diesem Fenster können die Grundeinstellungen des Photon Servers vorgenommen werden. Für die Einrichtung des Servers ist zuerst die Erstellung einer Photon App ID notwendig, welche mit einem kostenlosen Account auf *https://www.photonengine.com* erstellt werden kann.

- Auswählen des Photon Types „PUN“ und Erstellen der App ID (vgl. Abbildung 5)
- Anschließendes Kopieren der neu erstellten App ID (vgl. Abbildung 6)

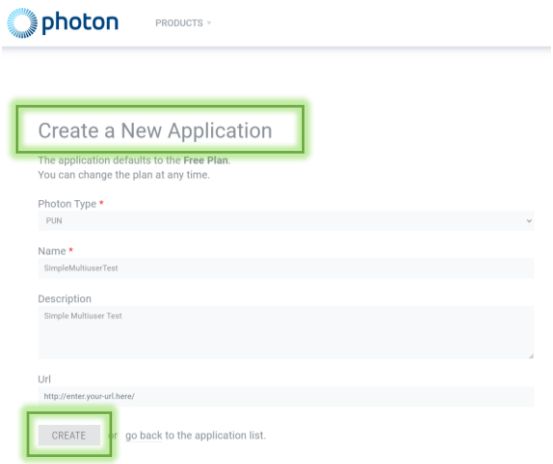


**Abbildung 5: Erstellen einer neuen App ID auf der Photon Engine Website**

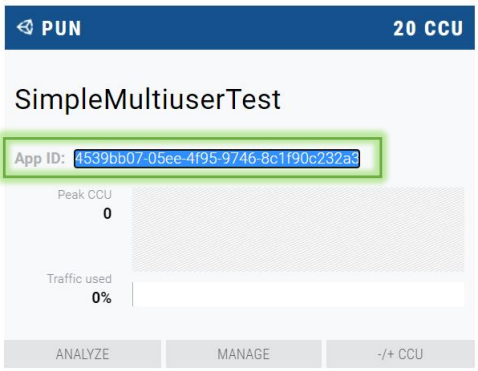


**Abbildung 6: Kopieren der neu erstellten App ID**

2.4. Einfügen der generierten App ID in die Dialogbox „PUN Setup“ (welche sich automatisch nach dem Import des PUN 2 – Free Packages öffnet). Abschließend mit dem Button „Setup Project“ bestätigen, vgl. Abbildung 7.

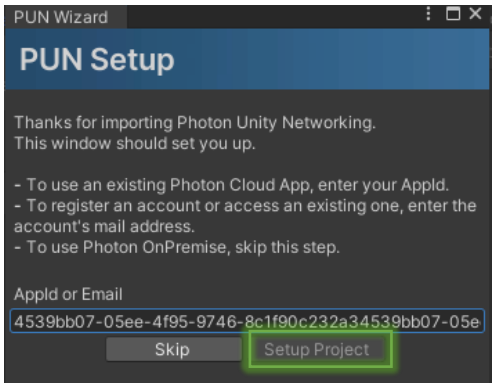


**Abbildung 7: Einrichtung der Server Settings mit dem erstellten App Key**

2.5. Die Server-Settings können in Unity nachträglich über den Tab: *Window → Photon Unity Networking → Highlight Server Settings* (vgl. Abbildung 8 und 9) geändert und überprüft werden. In den Server-Settings sollte „eu“ für Europa als „FixedRegion“ eingetragen werden.

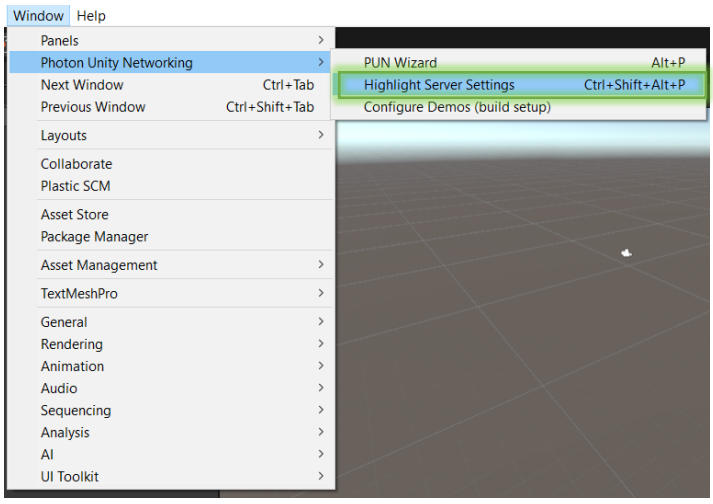


**Abbildung 8: Übersicht der Server-Einstellungen über die „Highlight Server Settings“**

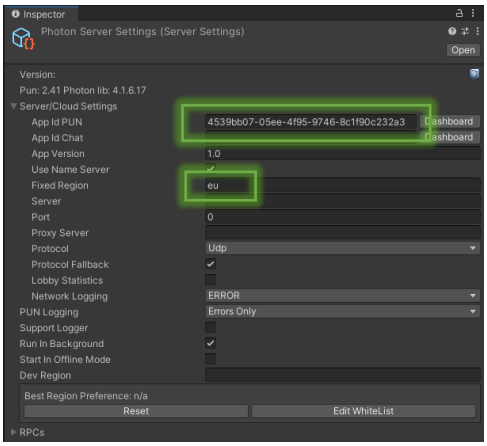


**Abbildung 9: „Highlight Server Settings“ zur Übersicht der App ID und Region „eu“ für Europa**

## 3. Erstellen und Spawnen von Objekten

3.1. Nachdem die Grundeinstellungen des Photon-Servers vorgenommen worden sind, können die 3D-Objekte erstellt werden. Dazu kann zuerst eine Plane als neues 3D-Object in der Hierarchie erstellt werden. Weiterhin sind zwei „Empty Game Objects“ notwendig. Diese können in „Network Manager“ und „Network Player“ umbenannt werden (vgl. Abbildung 10).

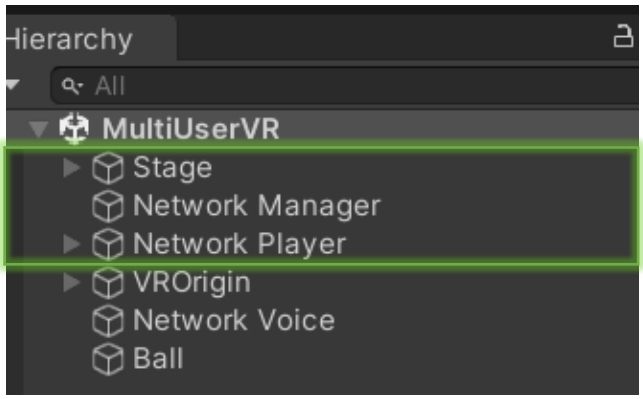


**Abbildung 10: Erstellen von 3D-Objekten in der Hierarchie: Plane, Network Manager und Network Player**

3.2. Zusätzlich ist ein Import des „XR Toolkit Assets“ für die Steuerung des gespawnten Teilnehmers mit dem XRRig notwendig. Dieses kann für neuere Unity Versionen, wie der verwendeten Version 2021.2.9f1, im Package Manager manuell unter „Adding Package by Name“, mit dem Namen „*com.unity.xr.interaction.toolkit“* geladen und in das Projekt importiert werden. Als XRRig sollte das „VROrigin“-Objekt aus der Beispielszene „3D Drag“ für die HTC Vive Focus 3 verwendet werden. Zudem kann das Skript „XROrigin.cs“ als Komponente auf das VROrigin-Objekt angefügt und folgenderweise angepasst werden:

- Drag-and-Drop des VR-Origin Objekts in das SerializeField „Origin Base GameObject”
- Drag-and-Drop des ViveCameraRig Objekts in das SerializeField „Camera Floor Offset Object”
- Drag-and-Drop der Camera in das SerializeField „Camera GameObject”
- Ändern des „Tracking Origin Modes“ auf Floor

3.3. Anschließend können an das 3D-Objekt „Network Manager“ die beiden C#-Skripte „NetworkManager.cs“ und „NetworkPlayerSpawner.cs“ als Komponenten angefügt werden (vgl. C#-Skripte im Anhang), vgl. Abbildung 10.

3.4. An das 3D-Objekt „Network Player“ kann das Skript „NetworkPlayer.cs“ angefügt werden (vgl. C#-Skript im Anhang), vgl. Abbildung 11.

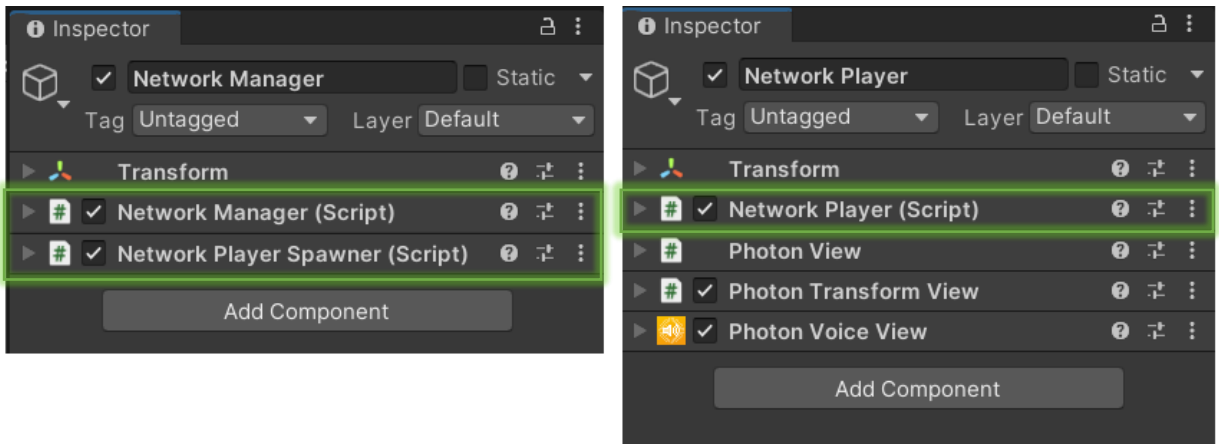


**Abbildung 11: Hinzufügen der Komponenten, links: "NetworkManager.cs", "NetworkPlayerSpawner.cs" an das „Network Manager“-Objekt, rechts: NetworkPlayer.cs“ an das „Network Player“-Objekt**

3.5. Erstellen eines neuen Ordners mit dem Namen „Resources“ im Asset-Folder (Der Name „Resources“ ist für die Photon-Default-Einstellungen wichtig und darf nicht umbenannt werden, ansonsten wird das Object nicht richtig gespawnt), vgl. Abbildung 12.

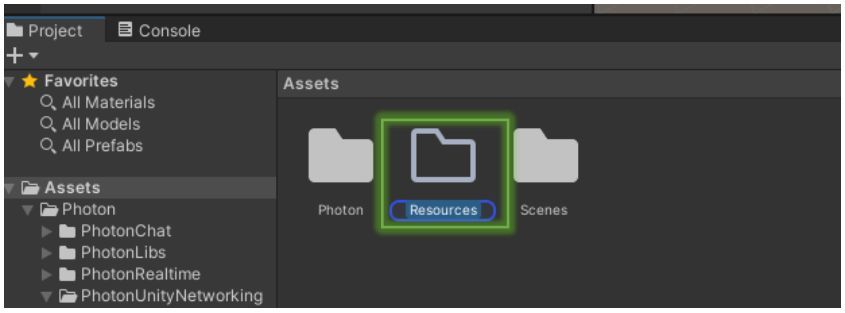


**Abbildung 12: Erstellen eines Resources Ordners für die alle Prefabs, welche als Teilnehmer gespawnt werden**

3.6. Aus dem Erstellten „Network Player“-Objekt kann ein Prefab durch einfaches „Drag-and-Drop“ aus der Hierarchie in den Resources-Folder (vgl. Abbildung 13) erstellt werden. Anschließend kann das 3D-Object „Network Player“ aus der Hierarchie wieder gelöscht werden.

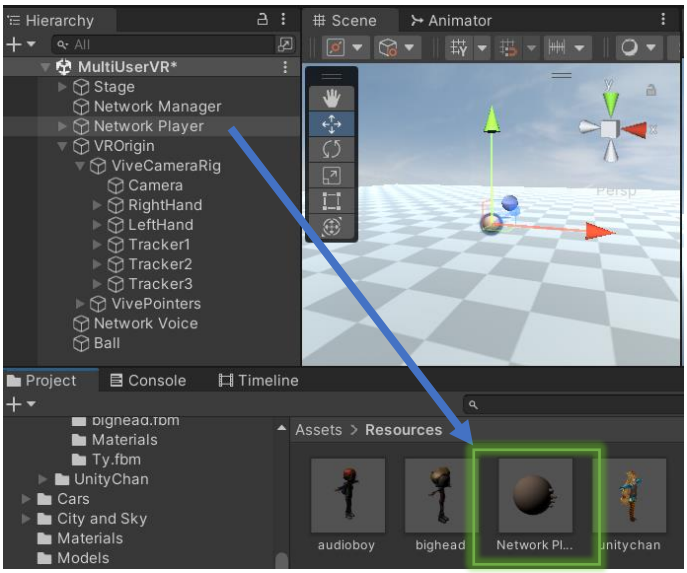


**Abbildung 13: Speichern eines 3D Objekts als Prefab im „Resources“- Ordner**

3.7. Das Prefab „Network Player“ sollte folgenderweise aufgebaut werden.

- Erstellen von weiteren drei „Empty Game Objects“ unter dem Parent-Objekt „Network Player“
- Umbenennen der neu erstellen Objekte in „Head“, „Left Hand“ und „Right Hand“.
- An die Empty Game Objects „Head“, „Left Hand“ und „Right Hand“ können nun 3D-Modelle als Child-Objekte, wie z.B. eine Sphere für den Kopf und Modelle für die linke und rechte Hand angefügt werden, vgl. Abbildung 14.

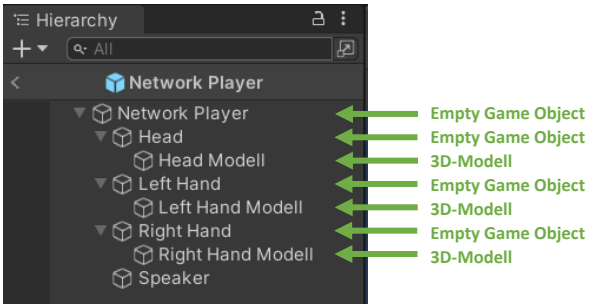


**Abbildung 14: Hierarchie und Aufbau des Prefabs „Network Player“**

3.8. Um die einzelnen Teilnehmer über den Photon-Server sichtbar zu machen, ist ein zusätzliches Hinzufügen der Komponente „Photon View“ zum Prefab „Network Player“ notwendig. Dazu kann das erstelle Prefab im Resources Folder durch Doppelklick geöffnet werden und das entsprechende C#-Skript als Komponente angefügt werden, vgl. Abbildung 15.

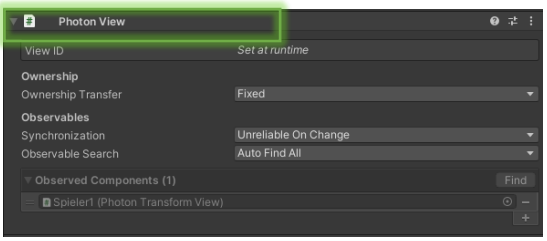


**Abbildung 15: Angefügten der Komponente „Photon View“ auf das Prefab „Network Player“**

3.9. Die VR Multiuser Szene „mit statischen Objekten“ ist soweit fertig eingerichtet. Die Szene kann in den „Build Settings“ hinzugefügt und anschließend als Android-Build auf eine HTC Vive Focus 3 gepatcht werden, vgl. Abbildung 16.

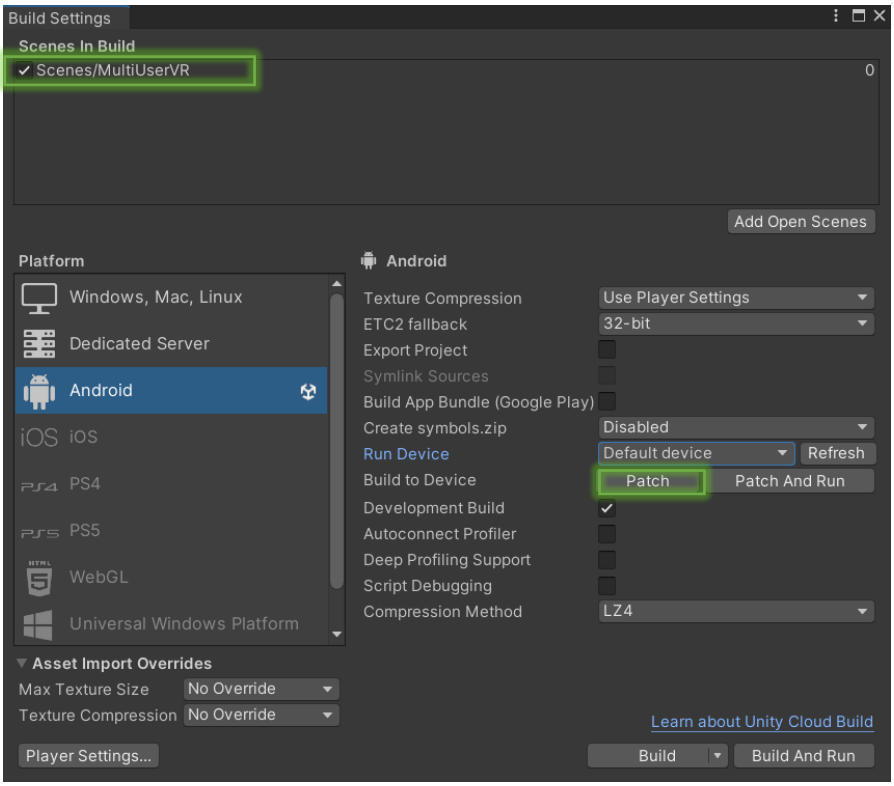


**Abbildung 16: Hinzufügen der MultiuserVR-Szene in den Build Settings und Patchen auf die HTC Vive Focus 3**

3.10. Nach dem Ausführen der Anwendung direkt in Unity und auf der HTC Vive Focus 3 wird für jede Instanz ein Teilnehmer im virtuellen Raum gespawnt. Die gespawnten Objekte werden zusätzlich in der Hierarchie angezeigt, vgl. Abbildung 17.


**Abbildung 17: Zwei Multi-User Objekte im virtuellen Raum mittels Photon Engine Networking in Unity**

## 4. Synchronisation von Bewegungen mit dem VR Headset über das Netzwerk

4.1. Um die Bewegungen der einzelnen Teilnehmer über den Photon-Server zu synchronisieren, ist ein zusätzliches Hinzufügen des Skripts „Photon Transform View“ zum Prefab „Network Player“ notwendig. Dazu kann das erstelle Prefab im Resources Folder durch Doppelklick geöffnet werden und das entsprechende Skript als Komponente angefügt werden, vgl. Abbildung 18.

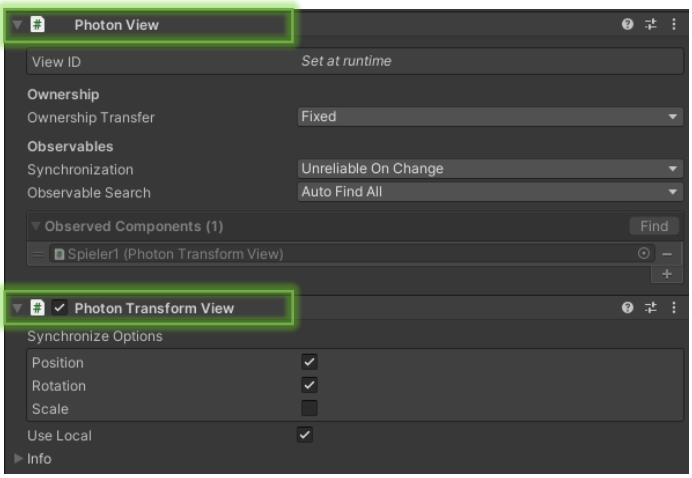


**Abbildung 18: Angefügte Komponenten Photon View und Photon Transform View auf dem Prefab „Network Player“**

4.2. Die Multiuser-Objekte werden für die Bewegung durch das „NetworkPlayer.cs“-Skript mit dem XRRig gekoppelt. Dazu können die drei Objekte „Head“, „Left Hand“ und „Right Hand“ des „Network Player“-Prefabs in das entsprechende SerializeField per Drag-and-Drop angefügt werden, vgl. Abbildung 19. Diese Bewegungen werden durch die „Photon Transform View“-Komponente über den Photon Server ausgetauscht.

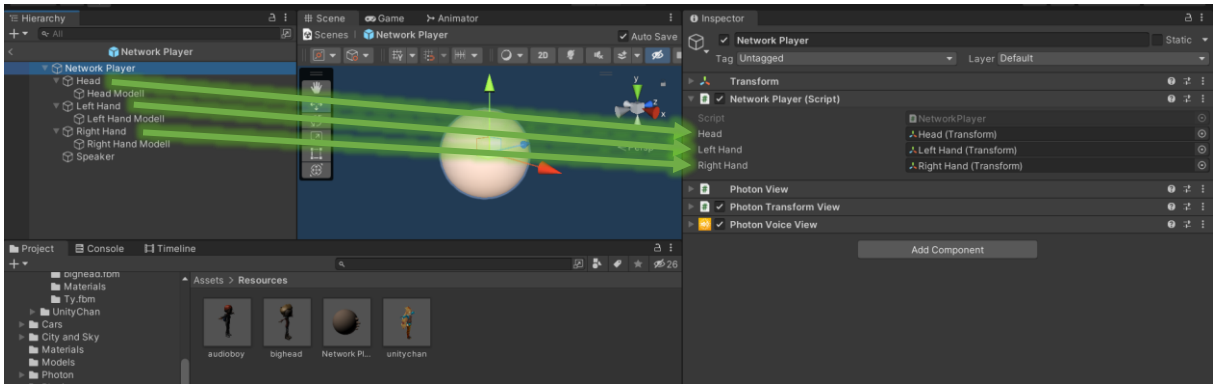


**Abbildung 19: Anfügen der Objekte "Head", "LeftHand", "RightHand" in die SerializeFields**

## 5. Interaktive Objekte

5.1. Erstellen eines neuen 3D Objekts in der Hierarchie, z.B. Sphere und umbenennen in „Interactable Object“ vgl. Abbildung 20.

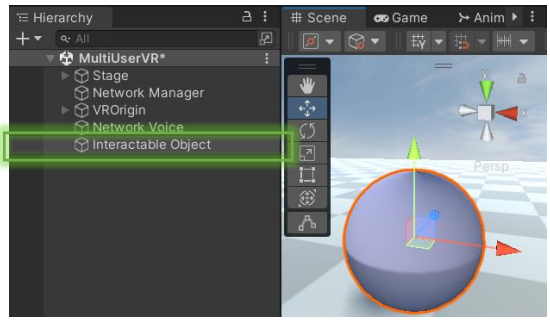


**Abbildung 20: Erstellen eines Cube als Interaktives Objekt in der Hierarchie**

5.2. Hinzufügen der Komponenten: Rigid Body, Photon View, Photon Transform View, Photon Rigidbody View und dem „Draggable“-C#-Skript (vgl. Abbildung 21).

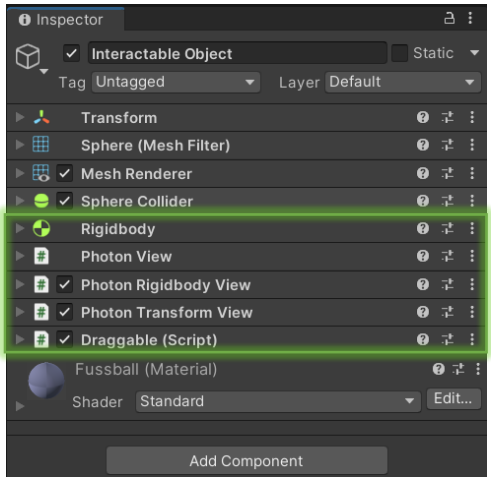


**Abbildung 21: Komponenten auf des Interaktiven Objekts**

5.3. In der „Photon View“ Komponente sollten folgende Anpassungen vorgenommen werden (vgl. Abbildung 22):

- Ownership Transfer auf „Takeover“ stellen
- Synchronization auf „Unreliable“ stellen
- Zu den “Observed Components” die (Photon Rigidbody View)-Komponente hinzufügen

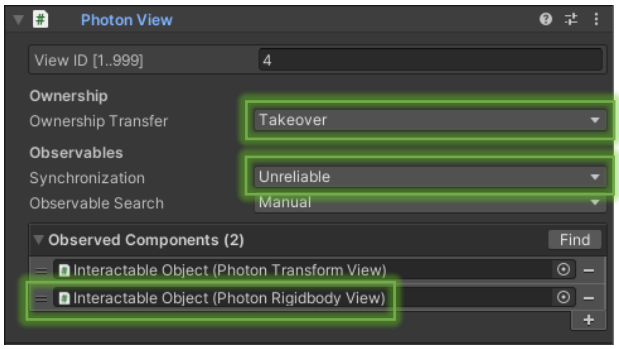


**Abbildung 22: Anpassungen in der Photon View Komponente für das Interaktive Objekt**

5.4. Weiterhin sollten in der Draggable-Komponente in den Unity-Events (After Grabbed, Before Release, On Drop) folgende Änderungen vorgenommen werden, damit die Interaction und Bewegung des Objekts für alle Teilnehmer und nicht nur dem Master-Client möglich ist (vgl. Abbildung 23):

- Drag-and-Drop des Objekts „Interactable Object“ in das SerializeField der Draggable-Komponente
- Auswählen der Funktion *PhotonView.OnRequestOwnership* des PhotonView-Skripts als Click-Event

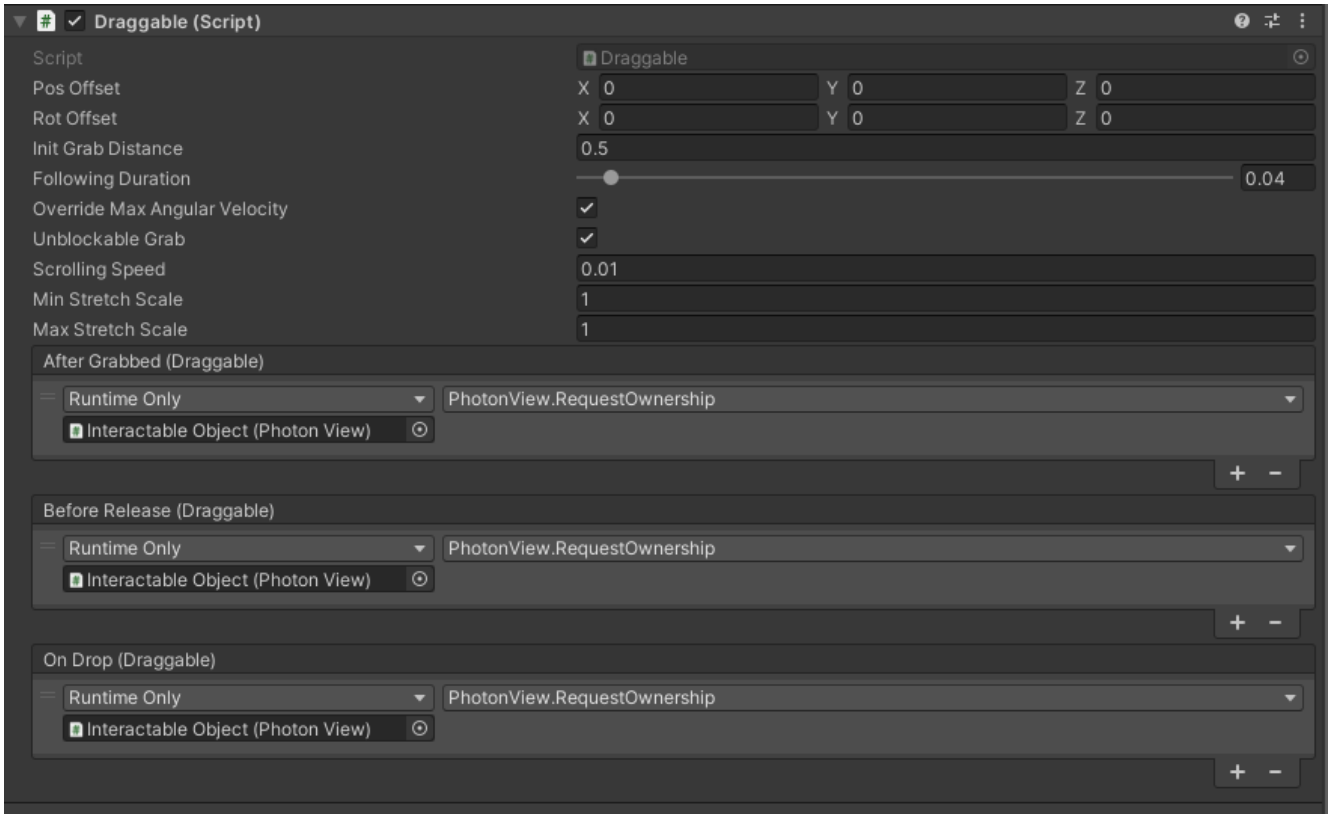


**Abbildung 23: Änderungen in der „Draggable“-Komponente für die Interaction aller Teilnehmern mit dem Objekt**

## 6. Voice Chat

6.1. Importieren des kostenlosen Photon Voice 2 - Assets (vgl. Abbildung 24) über den Paket Manager in das Unity Projekt.

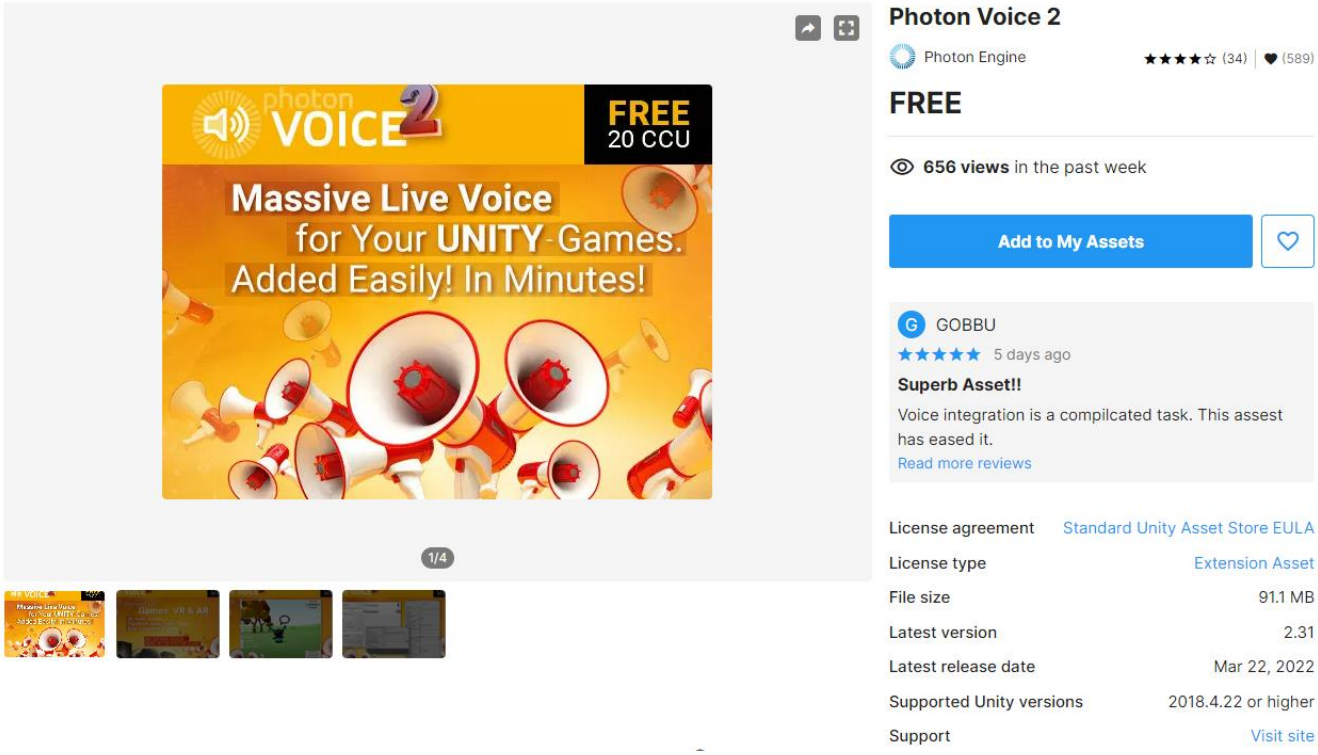


**Abbildung 24: Photon Voice 2 - Asset im Unity Asset Store**

6.2. Erstellung einer Photon Voice App ID mit einem kostenlosen Account auf https://www.photonengine.com.

- Auswählen des Photon-Types „PUN“ und Erstellen der App ID (vgl. Abbildung 25)
- Anschließendes Kopieren der neu erstellten App ID (vgl. Abbildung 26)

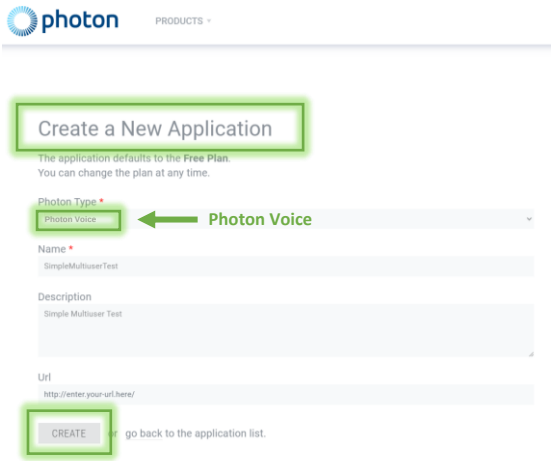


**Abbildung 25: Erstellen einer neuen Photon Voice App ID auf der Photon Engine Website**

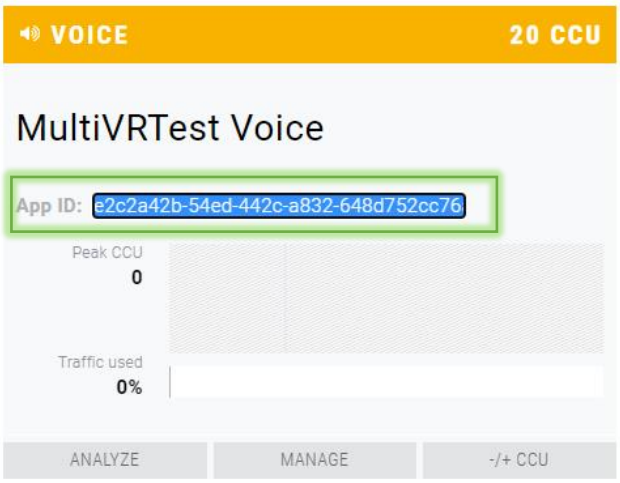


**Abbildung 26: Kopieren der erstellten Voice App ID**

6.3. Einfügen der generierten App ID in die Highlight Server Settings: *Window → Photon Unity Networking → Highlight Server Settings* in das SerializeField Photon ID Chat (vgl. Abbildung 27 und 28).

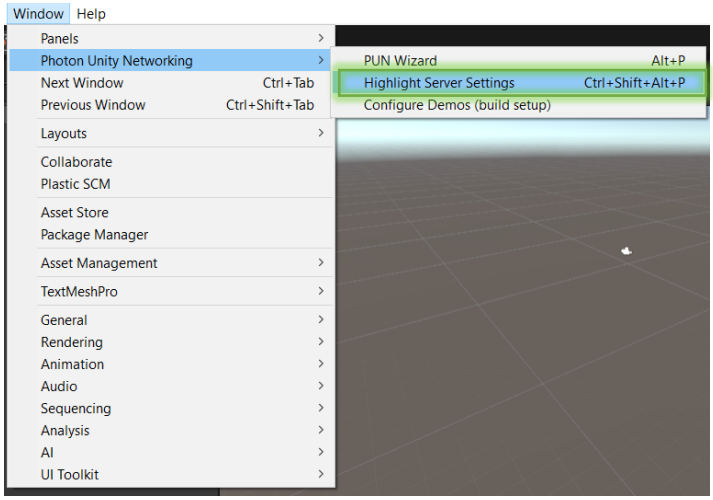


**Abbildung 27: Übersicht der Server-Einstellungen über die „Highlight Server Settings“**

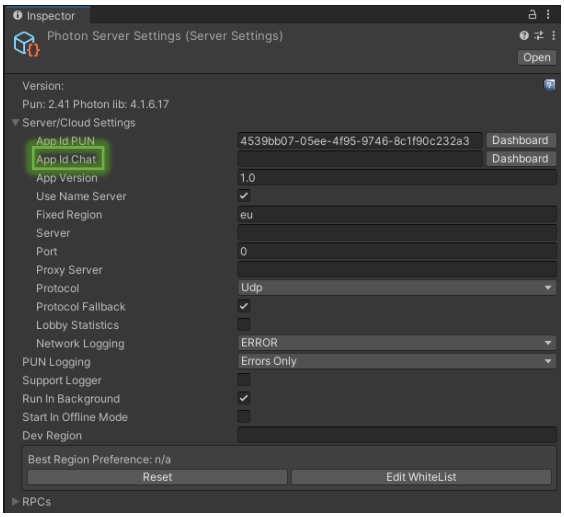


**Abbildung 28: „Highlight Server Settings“ zur Übersicht der App ID und Region „eu“ für Europa**

6.4. Erstellen eines neuen Empty Game Objects und Umbenennen in „Network Voice“.

6.5. Hinzufügen der Komponenten „Photon Voice Network“ und „Recorder“. Weiterhin sollten in der „Photon Voice Network“-Komponente alle Log Level auf „Error“ gestellt werden. In der „Recorder“-Komponente muss das Häkchen für „Transmit Enabled“ gesetzt werden. Die Recorder Komponente kann anschließend per Drag-and-Drop als Primary Recorder zur „Photon Voice Network“-Komponente hinzugefügt werden, vgl. Abbildung 29.

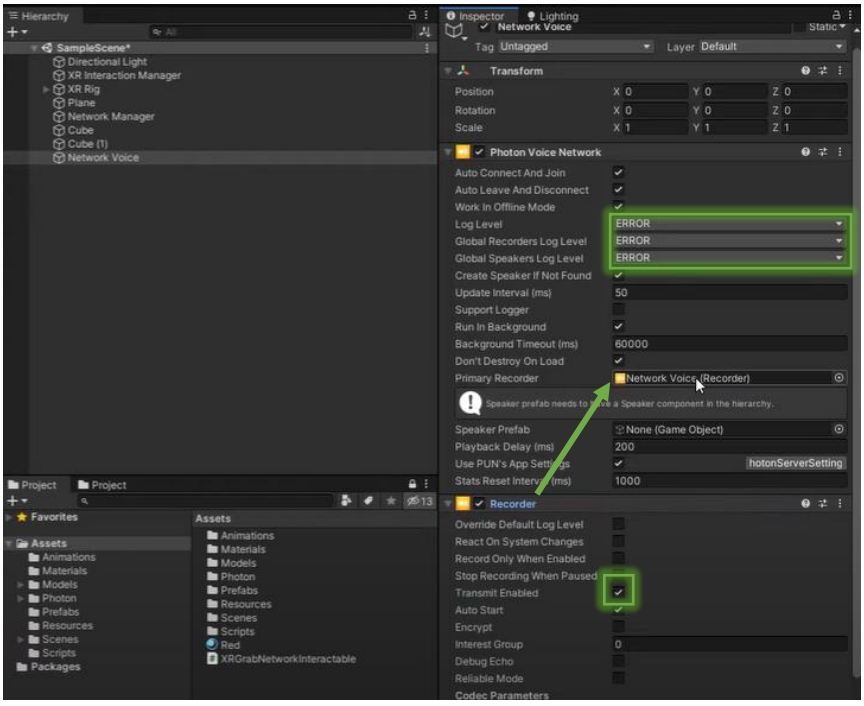


**Abbildung 29: Hinzufügen und Konfigurieren der Komponenten Photon Voice Network und Recorder auf das Objekt Network Voice**

6.6. Öffnen des Prefabs „Network Player“. Hinzufügen der Komponente „Photon Voice View“.

6.7. Erstellen eines neuen Empty Game Objects in dem Prefab „Network Player“ und Umbenennen in „Network Voice“. Hinzufügen der Komponente „Speaker“.

6.8. Hinzufügen des Game Objects „Network Voice“ per Drag and Drop zur Komponente „Photon Voice View“ auf dem „Network Player“. Weiterhin sollte in der Komponente „Photon Voice View“ das Häkchen bei „Use Primary Recorder“ gesetzt werden, vgl. Abbildung 30.

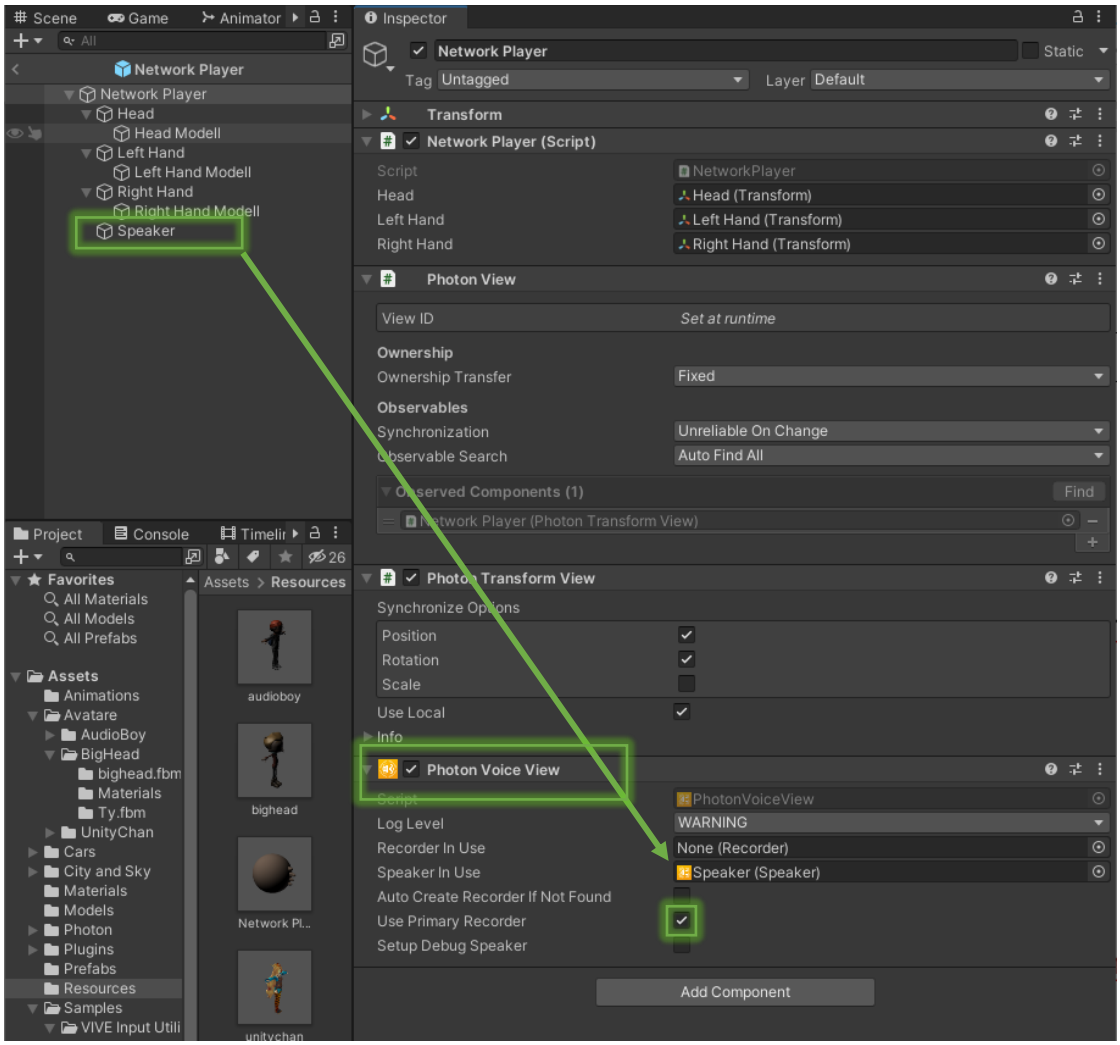


**Abbildung 30: Hinzufügen und Konfiguration der Komponente "Photon Voice View" sowie eines neuen Empty Game Objects Speaker mit der Komponente „Speaker“**

6.9. Zusätzlich ist in der Main-Szene der Multiuser ein „Audio Listener“ notwendig. Dazu sollten auf dem Camera-Objekt des XRRigs die Komponenten „Audio Listener“, „Recorder“ und das „Microphone Permission“-C#-Skript hinzugefügt werden, vgl. Abbildung 31.

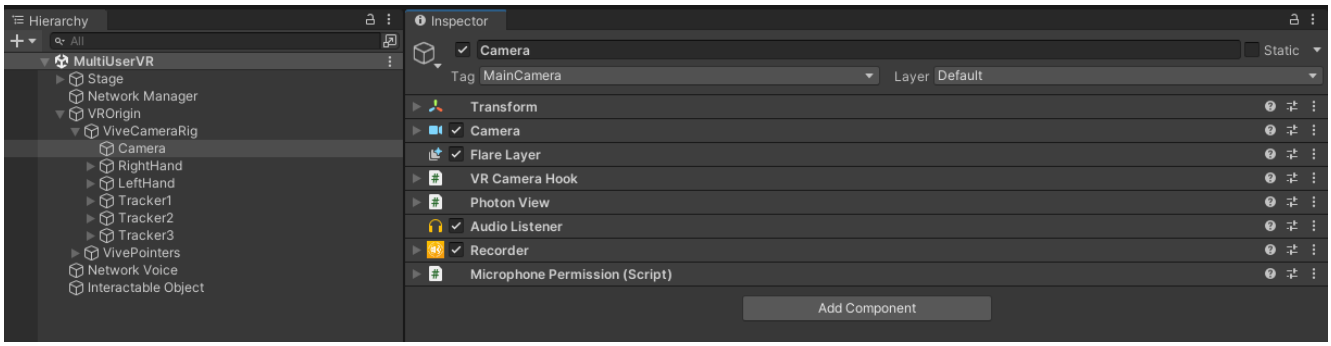


**Abbildung 31: Hinzufügen der Komponente „Audio Listener“, „Recorder“ und „Microphone Permission“ auf die Camera des XRRigs**

## Anhang

**NetworkManger.cs**

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;

public class NetworkManager : MonoBehaviourPunCallbacks
{
    void Start()
    {
        ConnectToServer();
    }

    void ConnectToServer()
    {
        PhotonNetwork.ConnectUsingSettings();
        Debug.Log("PhotonLogin: Verbindung zum Server wird hergestellt...");
    }

    public override void OnConnectedToMaster()
    {
        Debug.Log("Verbunden zum Server.");
        base.OnConnectedToMaster();
        RoomOptions roomOptions = new RoomOptions();
        roomOptions.MaxPlayers = 10;
        roomOptions.IsVisible = true;
        roomOptions.IsOpen = true;

        PhotonNetwork.JoinOrCreateRoom("Room 1", roomOptions, TypedLobby.Default);
    }

    public override void OnJoinedRoom()
    {
        Debug.Log("Joined a Room");
        base.OnJoinedRoom();
    }

    public override void OnPlayerEnteredRoom(Player newPlayer)
    {
        Debug.Log("Ein neuer Teilnehmer hat den Raum betreten.");
        base.OnPlayerEnteredRoom(newPlayer);
    }
}
```

**NetworkPlayerSpawner.cs**

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

public class NetworkPlayerSpawner : MonoBehaviourPunCallbacks
{
    private GameObject spawnedPlayerPrefab;

    public override void OnJoinedRoom()
    {
        base.OnJoinedRoom();
        spawnedPlayerPrefab = PhotonNetwork.Instantiate("Network Player", transform.position,
transform.rotation);
    }

    public override void OnLeftRoom()
    {
        base.OnLeftRoom();
        PhotonNetwork.Destroy(spawnedPlayerPrefab);
    }
}
```

**NetworkPlayer.cs**

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.XR.CoreUtils;
using UnityEngine.XR;
using Photon.Pun;
using UnityEngine.XR.Interaction.Toolkit;


public class NetworkPlayer : MonoBehaviourPun
{
    public Transform Head;
    public Transform LeftHand;
    public Transform RightHand;
    private PhotonView photonView;

    private Transform headRig;
    private Transform rightHandRig;
    private Transform leftHandRig;


    void Start()
    {
        photonView = GetComponent<PhotonView>();
        XROrigin rig = FindObjectOfType<XROrigin>();
        headRig = rig.transform.Find("ViveCameraRig/Camera");
        leftHandRig = rig.transform.Find("ViveCameraRig/LeftHand");
        rightHandRig = rig.transform.Find("ViveCameraRig/RightHand");
    }

    void Update()
    {
        if (photonView.IsMine)
        {
            MapPosition(Head, headRig);
            MapPosition(LeftHand, leftHandRig);
            MapPosition(RightHand, rightHandRig);
        }
    }

    void MapPosition(Transform target, Transform rigTransform)
    {
        target.position = rigTransform.position;
        target.rotation = rigTransform.rotation;
    }
}
```