Tutorial 7.

Photon Unity Networking (PUN)

2021-2022

COMP3329 Computer Game Design and Programming

Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)

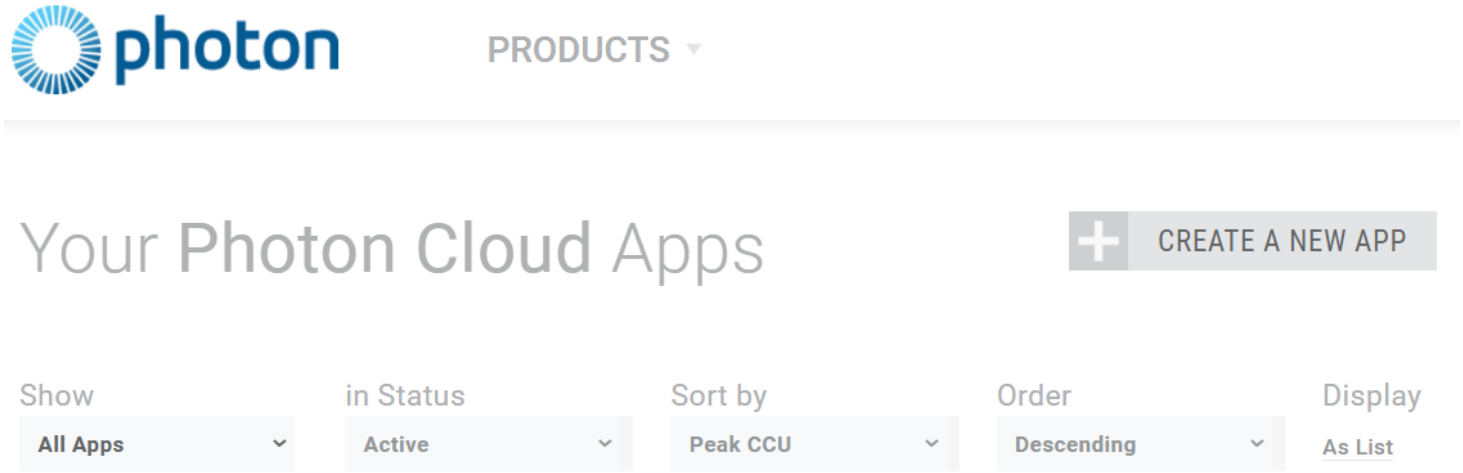
Department of Computer Science, The University of Hong Kong

Photon

- Photon Unity Networking (PUN) allows us to quickly create multiplayer games.
- Photon Cloud provides servers that we can connect to for testing our games.
- We can use Photon on our own servers, but we will save time by using Photon's free plan (after setting up an account).
- Photon's free plan supports 20 concurrent users (CCU).

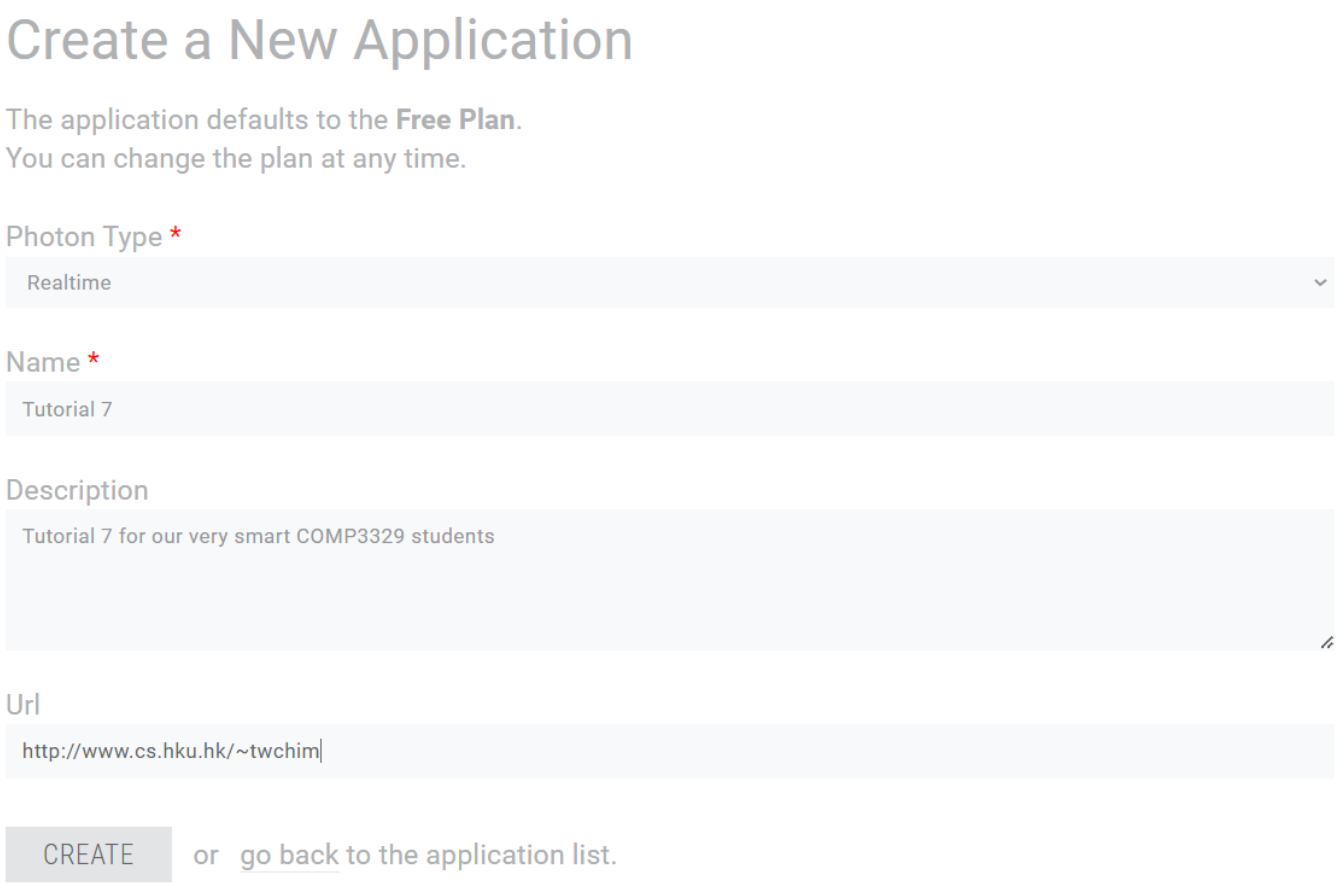
Photon Cloud Account Setup

- Go to Photon's registration page https://dashboard.photonengine.com/en-US/account/SignUp to register an account.
- After that, sign in and you will see the following.
- Click "CREATE A NEW APP".



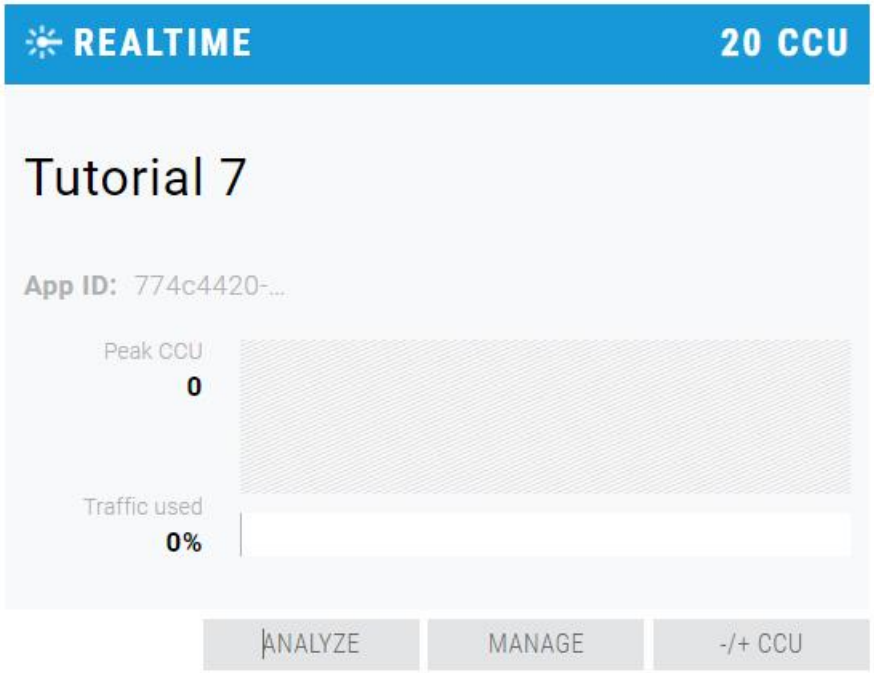
Photon Cloud Account Setup

- Fill in the information and click "CREATE".



Photon Cloud Account Setup

- After a short while, you will see the following on your dashboard.
- Click on the App ID section and you will see a long string. We will use it later in Unity.

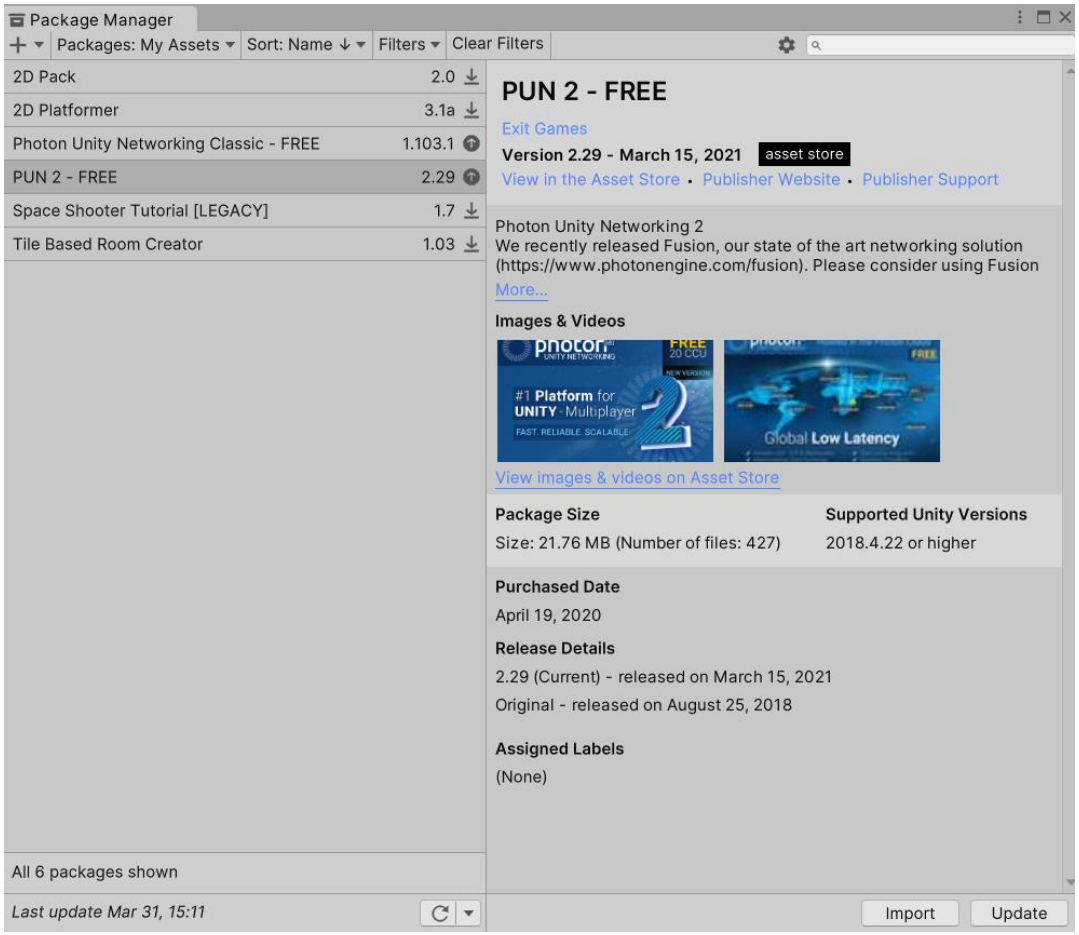


Setup PUN 2

- Create a new 3D Unity project.
- Go to the "Window → Assets Store" and click "Search online". The Unity Assets Store will appear in a browser window.
- Type "PUN" in the search bar and choose "PUN 2 – FREE".
- Press "Import" to import PUN 2 libraries into your project.
- Click "Open in Unity".

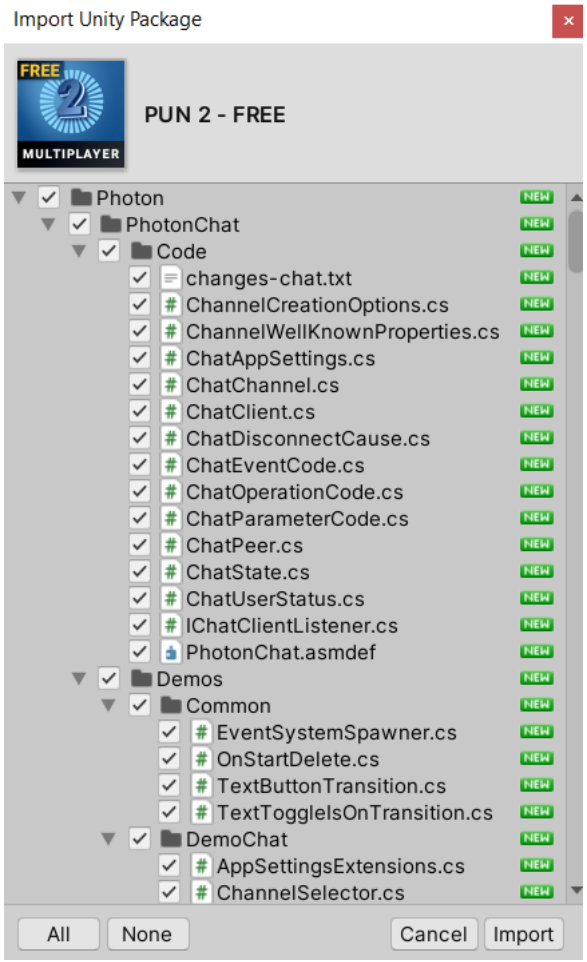
Setup PUN 2

- Click “Import” in the pop-up window.



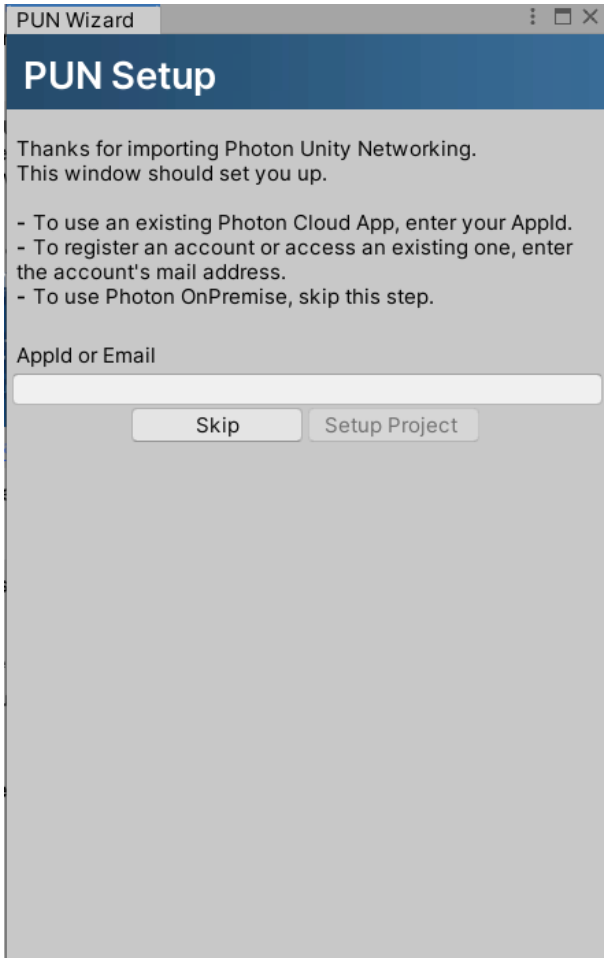
Setup PUN

- By default, all essential packages are selected. Just press “Import”.



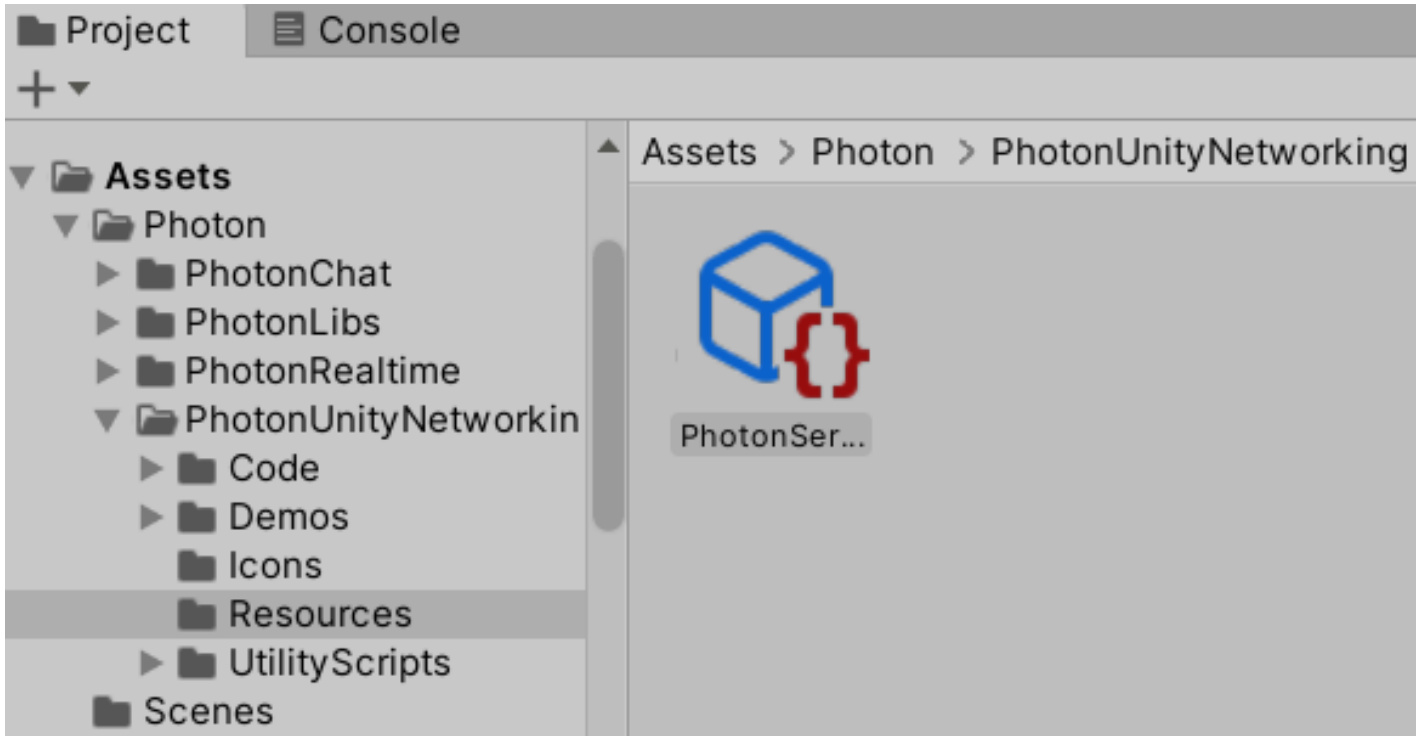
Setup PUN

- Enter the App ID you obtained from Photon dashboard.
- Click “Setup Project”.



Setup PUN

- Up to this point, all "Photon" stuff has been included into your Unity environment.



Connecting to Photon Servers

- Create a new C# script named "NetworkConnector.cs" and include the following code.

```
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;
public class NetworkConnector : MonoBehaviourPunCallbacks {
        void Start() {
                PhotonNetwork.ConnectUsingSettings();
        }

        #region Pun Callbacks
        public override void OnConnectedToMaster() {
                Debug.Log("Connected to photon!");
        }
        public override void OnDisconnected(DisconnectCause cause) {
                Debug.LogWarning($"Failed to connect: {cause}");
        }
        #endregion
}
```

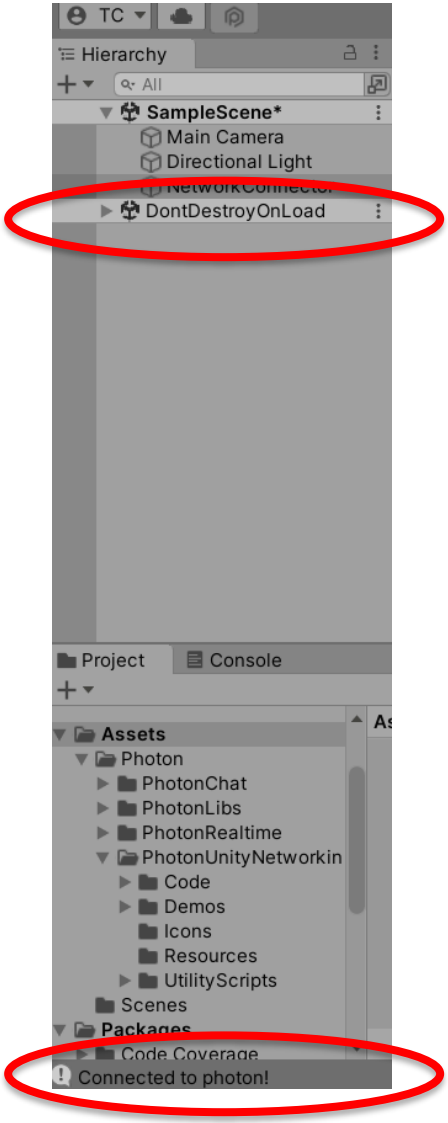
MonoBehaviorPunCallbacks class holds all methods we can override to react to events in Photon.

Key steps:

- Try to connect to Photon using the settings configured in Server Settings.
- Log a message when connected to the Photon Master server.
- Log a warning when connection cannot be established.

Connecting to Photon Servers

- Create an empty **GameObject** and name it “NetworkConnector”.
- Associate the newly created “NetworkConnector.cs” script with it.
- Run the game and you should see the circled items.



Room Concept

- The Photon Cloud is built with "room-based games" in mind, meaning that there are many players per match, separated from anyone else.
- In a room (usually) everyone receives whatever the others send. Outside of a room, players are not able to communicate, so we always want them in rooms as soon as possible.
- Unless the room is full or closed, we can join it. Conveniently, the Master Server maintains a list of rooms, and this is denoted as a lobby.

Create Room / Join Room

- Modify "NetworkConnector.cs" as follows:
  - Add the following into "OnConnectedToMaster()" method:

```
// Try to join a random room
PhotonNetwork.JoinRandomRoom();
```

  - Add the "OnJoinRandomFailed()" and "OnJoinedRoom()" methods:

```
public override void OnJoinRandomFailed(short returnCode, string message) {
        // Failed to connect to random, probably because none exist
        Debug.Log(message);
        // Create a new room
        PhotonNetwork.CreateRoom("My First Room");
}
public override void OnJoinedRoom() {
        Debug.Log($"{PhotonNetwork.CurrentRoom.Name} joined!");
}
```

When you run your game, you should first see the message "No match found" (meaning there is no random room to join) and then the message "My First Room joined!".

Create a Player Prefab with Networking

- When creating a network game, we need to think about networked objects (not only local objects).
- We can't simply use "Instantiate()" method because this will create the player object only for the local player, instead of for all players in the room.
- Photon provides "PhotonNetwork.Instantiate()" method. This will create the object in the room for all players to see.
- If we want to update its values (say position) and share to other players, we need to add a "PhotonView" component to the object.

Create a Player Prefab with Networking

- Please follow the steps below:
  - Create a new "3D Object → Capsule" and name it as "Player".
  - Reset its position to 0, 0, 0.
  - Add a "Rigibody" component, under "Constraints → Freeze Rotation", check X, Y, and Z. Tick "Is Kinematic".
  - Add a "Photon Networking → PhotonView" component.
    - For the "Observable Search", change it to "Manual".
    - In the "Observed Components" section, drag the "Player" object to it. This should create a Photon Transform View which will automatically synchronize our object's transform to other players in the room.
  - Uncheck "Rotation" under "Photon Transform View".

Create a Player Prefab with Networking

- Create a new C# script named "BasicMovement.cs" and associate it with the player.

```
using UnityEngine;
using Photon.Pun;

public class BasicMovement : MonoBehaviour {
	public float speed;
		Rigidbody rb;
		PhotonView photonView;

		// Start is called before the first frame update
		void Start() {
			rb = GetComponent<Rigidbody>();
			photonView = GetComponent<PhotonView>();
		}
```

Create a Player Prefab with Networking

```
    void FixedUpdate() {
        // Only move the player object if it's the local user's player
        if (photonView.IsMine) {
            Move();
        }
    }

    void Move() {
        float horizontal = Input.GetAxis("Horizontal") * speed * Time.deltaTime;
        float vertical = Input.GetAxis("Vertical") * speed * Time.deltaTime;
        Vector3 newVector = new Vector3(horizontal, 0, vertical);
        rb.position += newVector;
    }
}
```

Create a Player Prefab with Networking

- Very brief explanations:
  - Based on the WASD or keyboard arrow keys, we can move our player forward, back, left, and right.
  - One exception is the "photonView" section. We have already added the "PhotonView" component to our player object. Here we get a reference to it and use it to make sure we only move the player object that belongs to the local user ("photonView.IsMine").
  - Each player will need to see the other player's object, so we will create a player object for each player when they join the room. However, we don't want other users controlling our local character!

- Add the "BasicMovement.cs" script to our player object and set the speed to 10.

Saving the Prefab

- Create a "Resources" folder and drag the player object into it to create a prefab.
- Important: Because of the way Photon searches for GameObjects to instantiate, they must be inside of a "Resources" folder. The folder could be nested within a "Prefab" folder, but it must be named "Resources".

Instantiate “Remote” Prefabs

- Modify “NetworkConnector.cs” as follows:
  - Add the following:

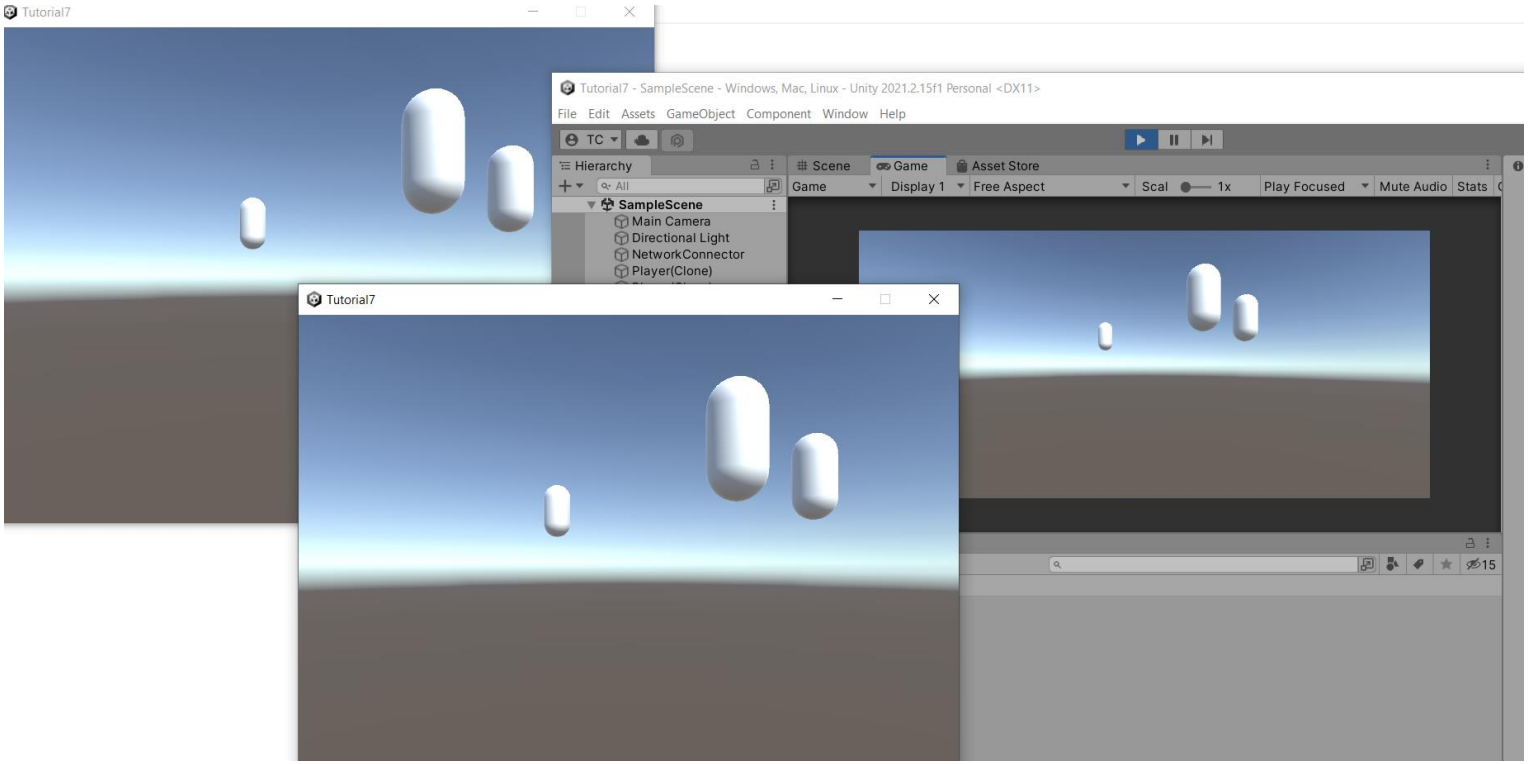
    ```
    public GameObject playerPrefab;
    ```

  - Add the following into “OnJoinedRoom()” method:

    ```
    PhotonNetwork.Instantiate(playerPrefab.name,
    new Vector3(0, 3.0f, 0), Quaternion.identity);
    ```

Test Multiple Instances

- Build the exe so that you can run another instance of the game.
- When you run both instances, you can see the motion of each other players.



Save and Submit Your Work

- Please save your work, zip the project folder and submit to Moodle by April 28, 2022 (Thursday) as a proof of tutorial participation.

We are with you!

If you encounter any problems in understanding the materials in the lectures, **please feel free to contact me or my TA**.
We are always with you!

We wish you enjoy learning game programming in this class.

Tutorial 7.

End

2021-2022

COMP3329 Computer Game Design and Programming

Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)

Department of Computer Science, The University of Hong Kong