# Unity – Networking

*PhotonNetwork will be used as the Network connector. PhotonNetwork is downloaded as a plugin form the assets store for free but I have already included it in the project.*

A NetworkManager script takes care of most thing and can be seen as the central hub for the networking.

TODO

Create server @ <https://www.exitgames.com/en/OnPremise/Pricing>

**Basics**

To allow networking in a class inherit Photon.MonoBehaviour instead of the Monobehaviour.

void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)

{

    if (stream.isWriting)

        stream.SendNext(rigidbody.position);

    else

        rigidbody.position = (Vector3)stream.ReceiveNext();

}

Is an example of serializing and reading over the network. Can be used when Photon.MonoBehaviour is inherited.

Or if you want to lerp the movement use

private float lastSynchronizationTime = 0f;

private float syncDelay = 0f;

private float syncTime = 0f;

private Vector3 syncStartPosition = Vector3.zero;

private Vector3 syncEndPosition = Vector3.zero;

void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)

{

    if (stream.isWriting)

    {

        stream.SendNext(rigidbody.position);

    }

    else

    {

        syncEndPosition = (Vector3)stream.ReceiveNext();

        syncStartPosition = rigidbody.position;

        syncTime = 0f;

        syncDelay = Time.time - lastSynchronizationTime;

        lastSynchronizationTime = Time.time;

    }

}

WITH:

void Update()

{

    if (photonView.isMine)

    {

        InputMovement();

    }

    else

    {

        SyncedMovement();

    }

}

private void SyncedMovement()

{

    syncTime += Time.deltaTime;

    rigidbody.position = Vector3.Lerp(syncStartPosition, syncEndPosition, syncTime / syncDelay);

}

This will lerp the movement so for the other players will have a smooth movement.

## Movement Prediction

Though the transitions look smooth, you notice a small delay between the input and the actual movement. This is because the position is updated after the new data is received. Until we invent time travel, all we can do is predict what is going to happen based on the old data.

Use:

void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)

{

    if (stream.isWriting)

    {

        stream.SendNext(rigidbody.position);

        stream.SendNext(rigidbody.velocity);

    }

    else

    {

        Vector3 syncPosition = (Vector3)stream.ReceiveNext();

        Vector3 syncVelocity = (Vector3)stream.ReceiveNext();

        syncTime = 0f;

        syncDelay = Time.time - lastSynchronizationTime;

        lastSynchronizationTime = Time.time;

        syncEndPosition = syncPosition + syncVelocity \* syncDelay;

        syncStartPosition = rigidbody.position;

    }

}

## Remote Procedure Calls

Another method of network communication is Remote Procedure Calls (RPCs), which is more useful for data that does not constantly change. We will change the color of a player over the network.

private void InputColorChange()

{

    if (Input.GetKeyDown(KeyCode.R))

        ChangeColorTo(new Vector3(Random.Range(0f, 1f), Random.Range(0f, 1f), Random.Range(0f, 1f)));

}

[RPC]

void ChangeColorTo(Vector3 color)

{

    renderer.material.color = new Color(color.x, color.y, color.z, 1f);

    if (photonView.isMine)

        photonView.RPC("ChangeColorTo", PhotonTargets.OthersBuffered, color);

}

References:

<http://docs.unity3d.com/ScriptReference/Network.html>

<https://unity3d.com/learn/tutorials/modules/intermediate/live-training-archive/merry-fragmas-multiplayer-fps-3>

<http://www.paladinstudios.com/2014/05/08/how-to-create-an-online-multiplayer-game-with-photon-unity-networking/>