Photon Cloud(二)
單向傳輸

主講：紀曲峰

PUN

- PUN是Photon Unity Networking的縮寫

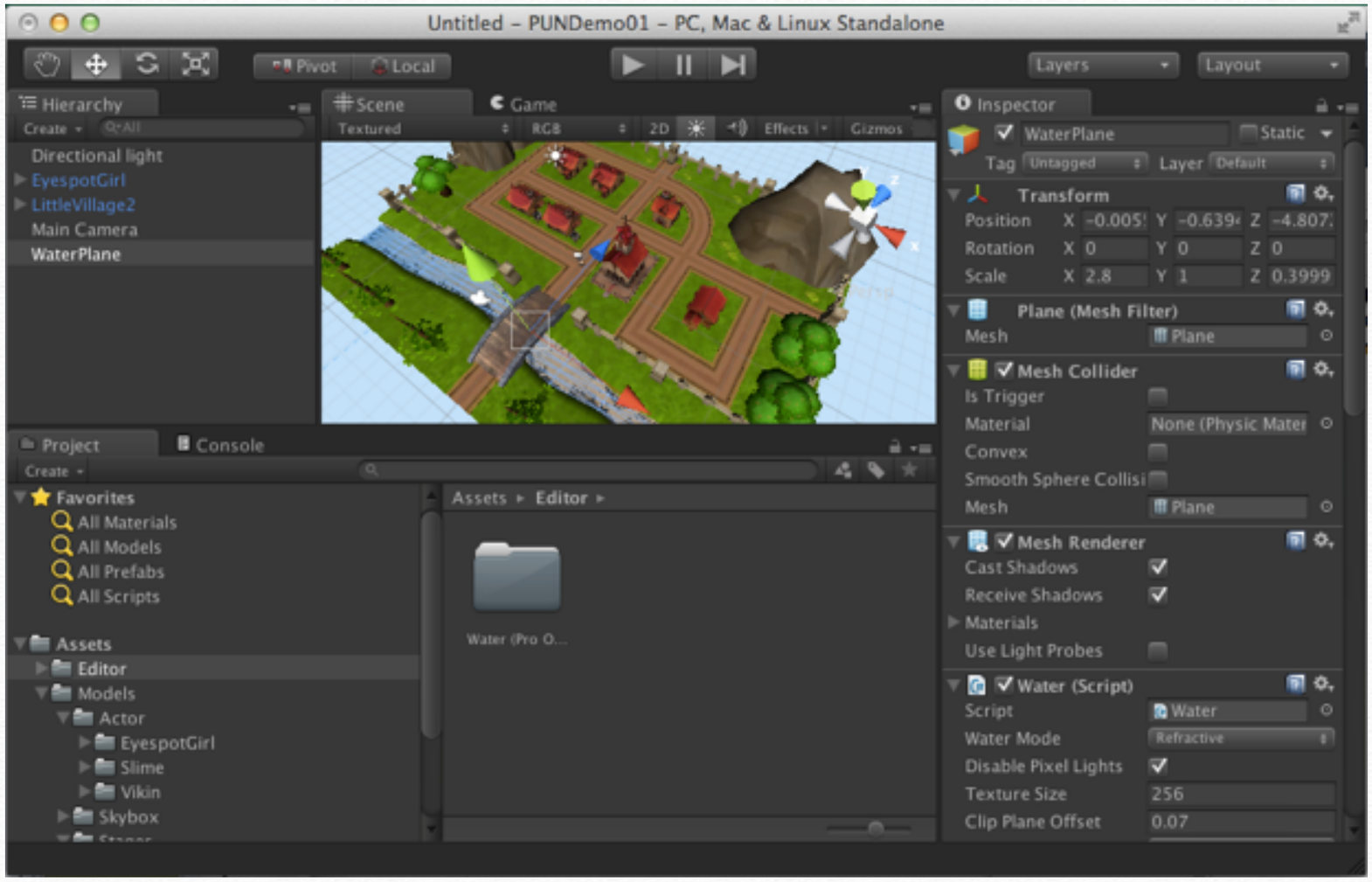
- PUN為Photon針對Unity重新包裝，為最容易使用的雲端遊戲伺服器解決方案

- 使用PUN不需要有Photon的基礎，但有更佳

準備PUN專案

準備專案

- 建立一個新專案，並將必要場景和角色放入

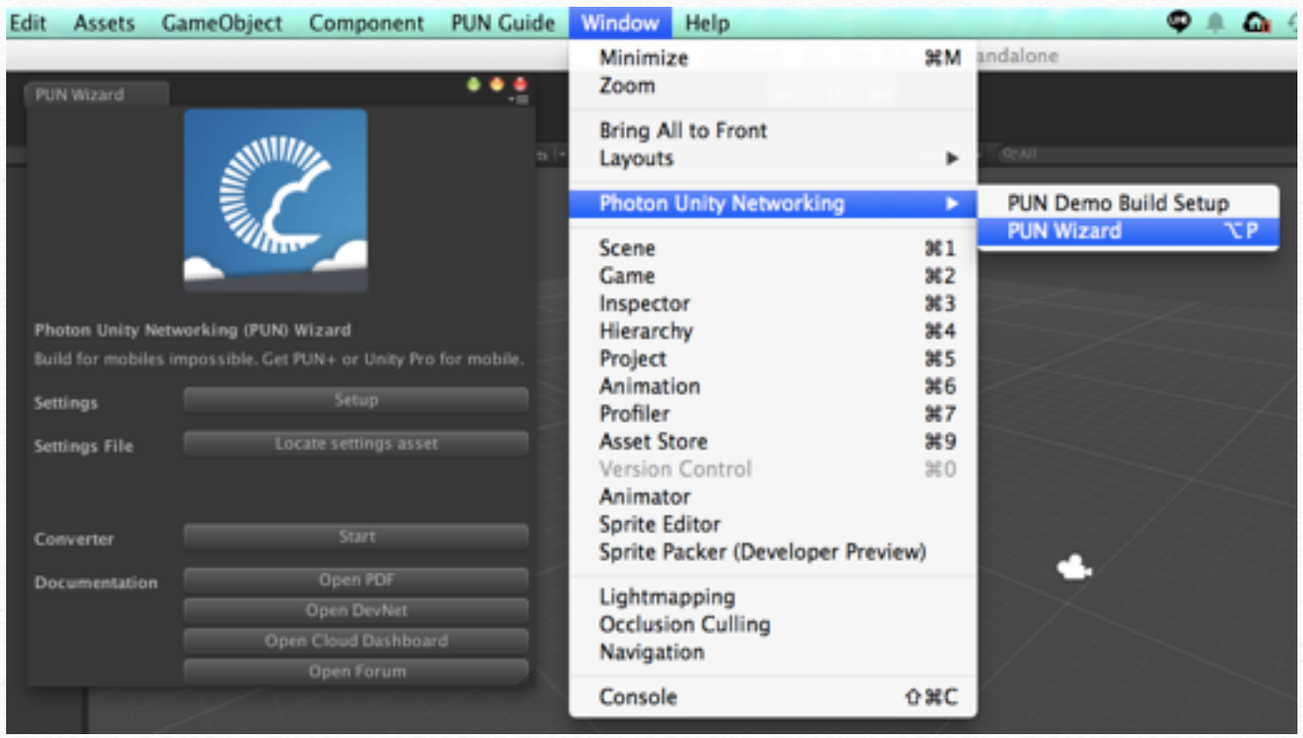


匯入PUN

- 開啟Asset Store找到PUN並匯入

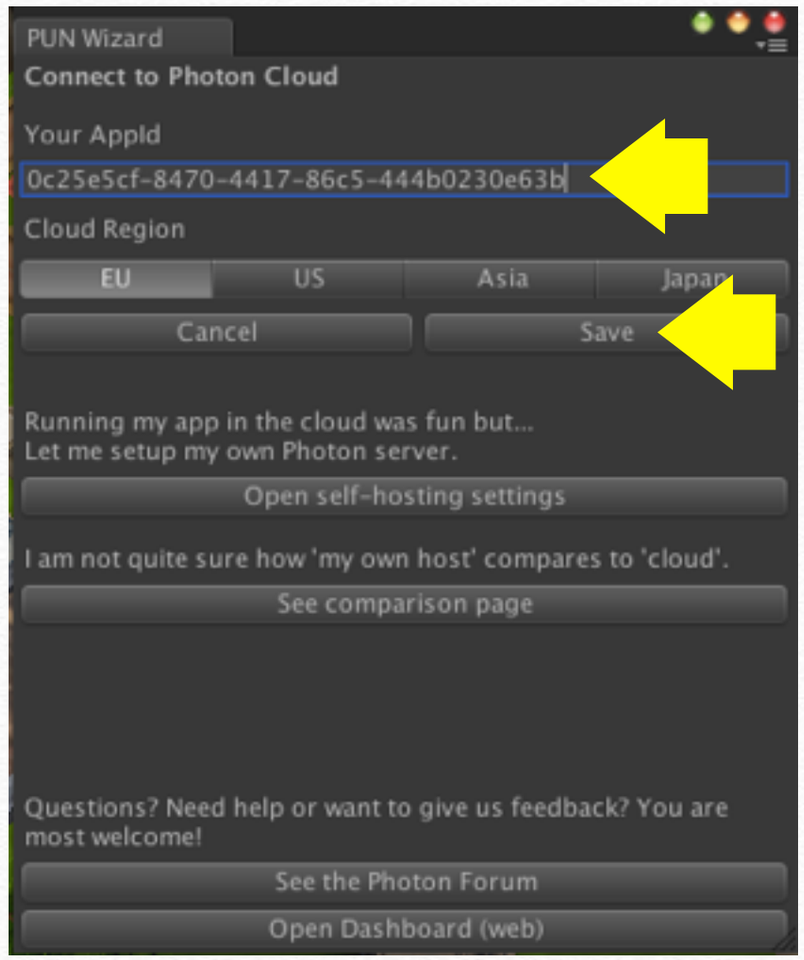
啟動設定視窗

- 匯入PUN會跳出一個設定視窗
- 若不小心關掉設定視窗，可執行下面指令重新開啟
  Window > Photon Unity Networking > PUN Wizard



輸入AppId

- 在PUN Wizard視窗裡輸入AppID後按下Save儲存，若尚無AppID請先註冊，在視窗裡輸入Mail後收取郵件密碼即可完成註冊

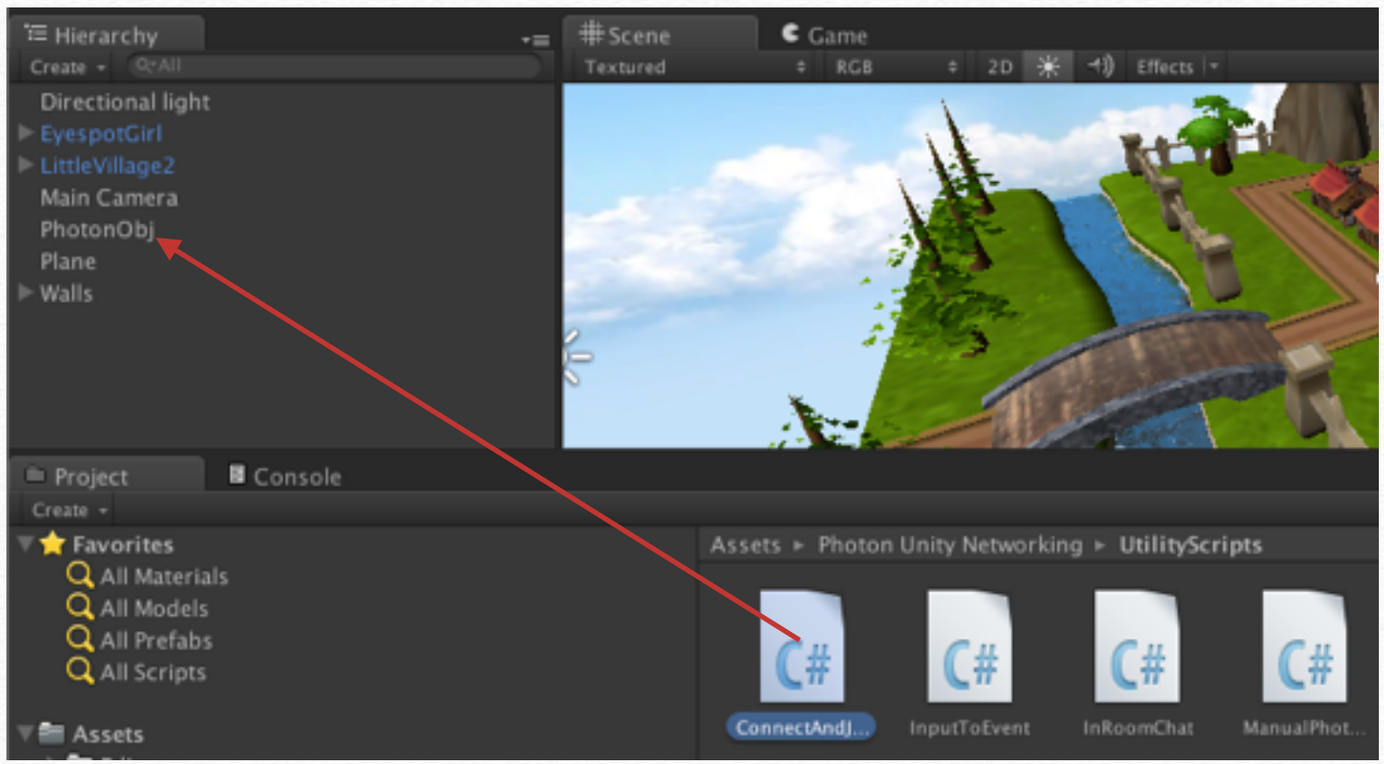


- AppID取得請參考前份簡報

PUN基本連線

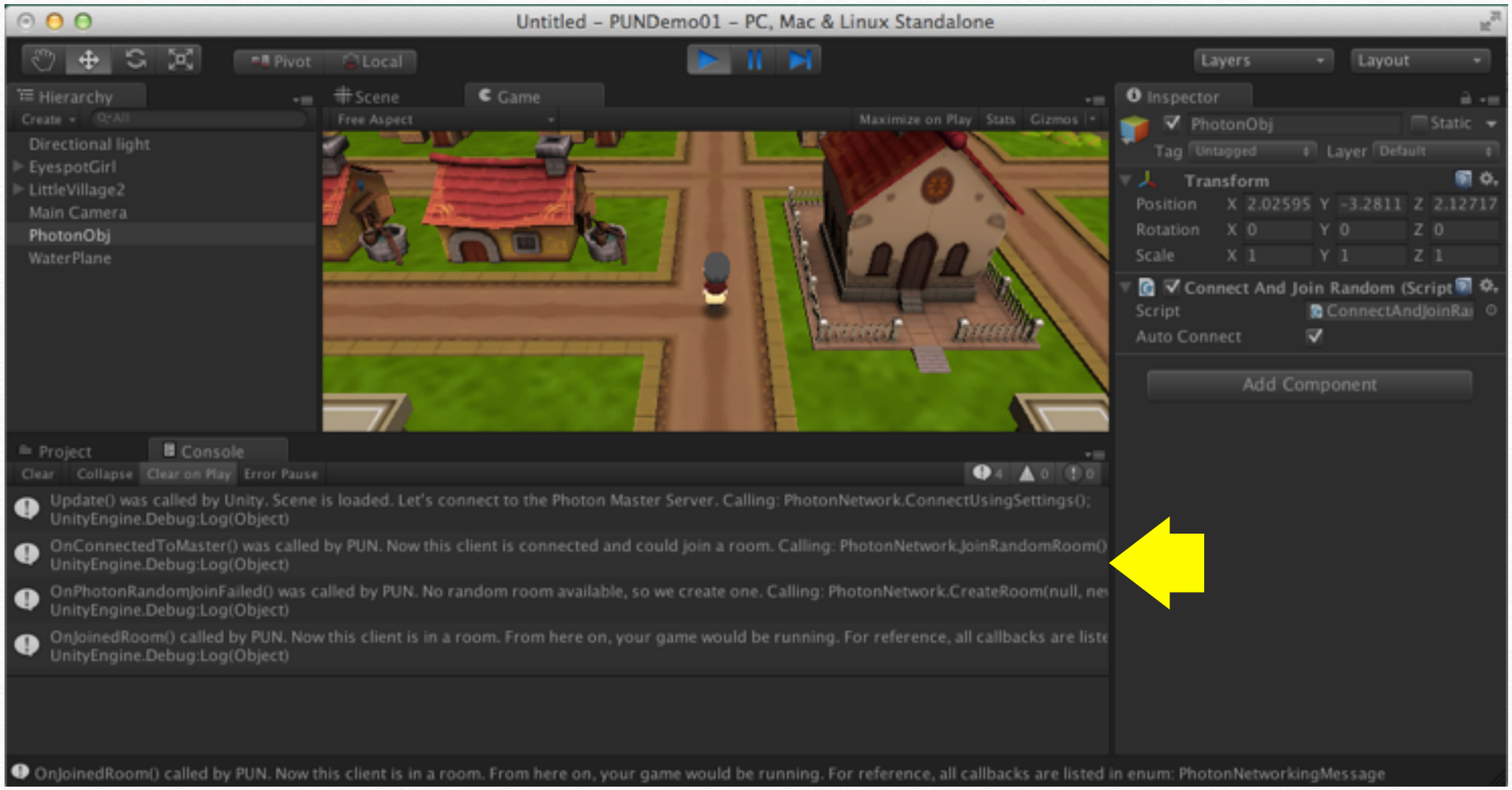
使用現成腳本快速建立連線

- 找到UtilityScripts/ConnectAndJoinRandom.cs
- 建立一個Empty物件，這裡將之取名為PhotonObj
- 將ConnectAndJoinRandom拉到PhotonObj上即完成



執行程式

- 直接執行後，連線結果將顯示於主控台



在遊戲中顯示連線資訊

- 建立一個腳本，取名為PhotonGUI.cs
- 輸入以下內容

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(PhotonView))]
public class PhotonGUI : MonoBehaviour {

    public void OnGUI()
    {
        GUILayout.Label(PhotonNetwork.connectionStateDetailed.ToString());
    }
}
```

無所不在的PhotonView

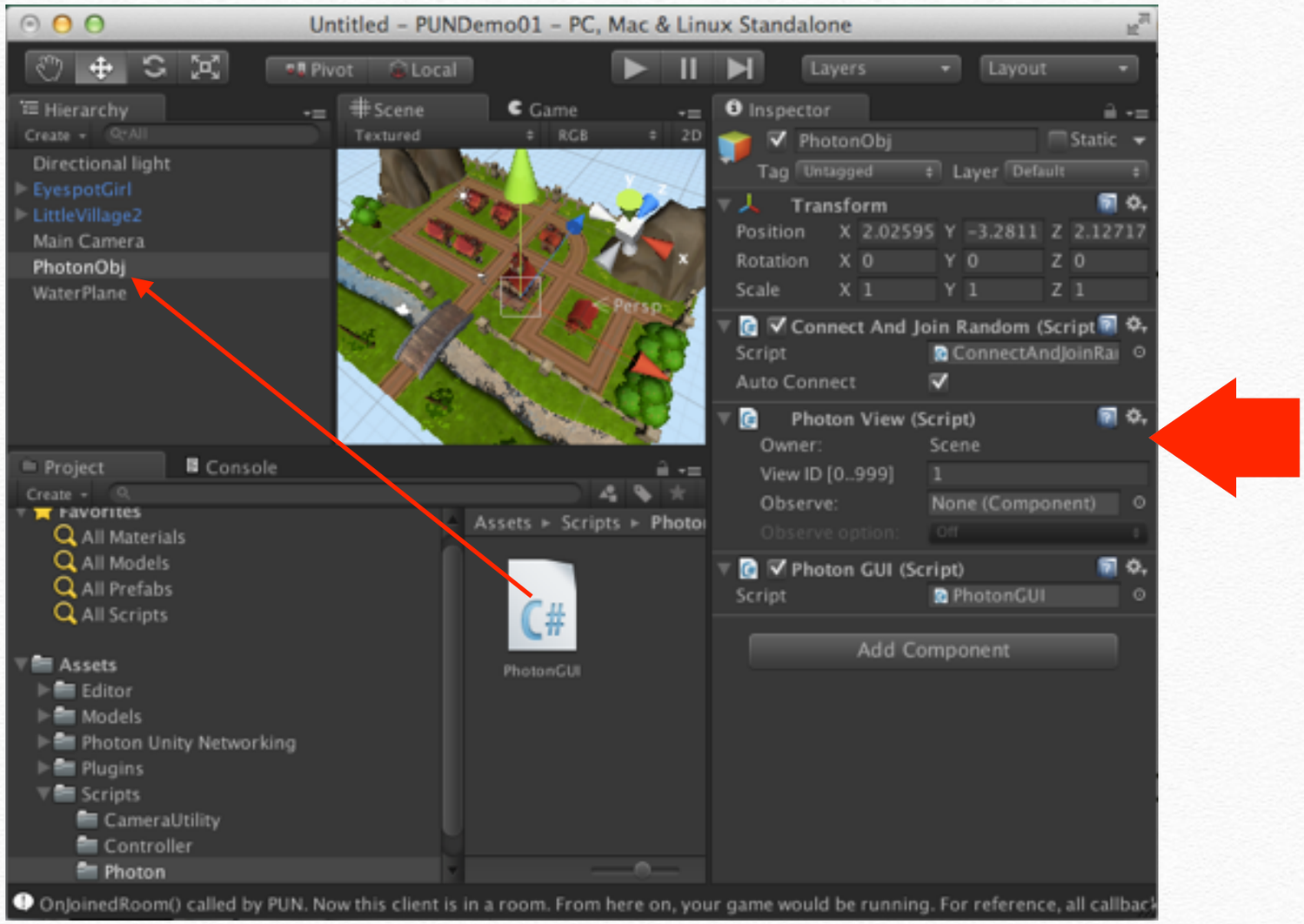
- 這是類似Unity的NetworkView，為主要的傳輸元件，只要是需要收發訊號的GameObject都要加入PhotonView

- 此腳本為於PhotonNetwork/PhotonView

- 若PhotonView為必要項可在腳本前加入
  [RequireComponent(typeof(PhotonView))]

將PhotonGUI放到場景

- 把PhotonGUI拉到PhotonObj上，可以看到PhotonView被自動附加上去



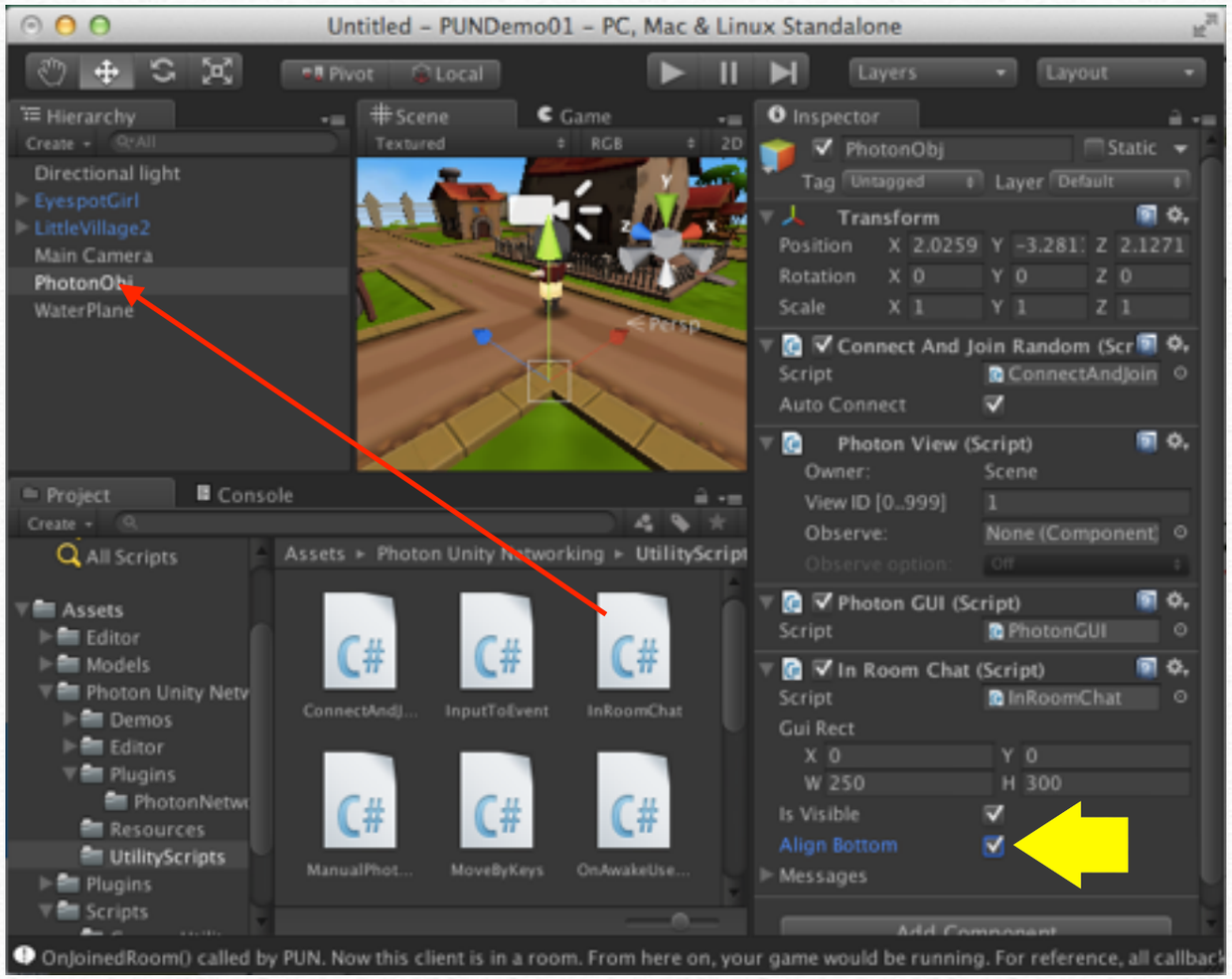
重新執行

- 重新執行後左上角可以看到GUI畫出的連線狀態了

聊天室

加入聊天室

- 找到 UtilityScripts/InRoomChat.cs ，拉到PhotonObj上
- 將Align Bottom打勾以對齊下方



執行聊天室

- 執行後即可聊天，亦支援多國語言

建立單向傳輸

基礎知識

- 在P2P遊戲中通常有其中一個客端遊戲做為邏輯的Game Server用
- 通常是最早連線進去或是建立房間的客端當作Game Logic Server用
- Photon Cloud非完全P2P，因此遊戲邏輯端稱之為Master而不叫Game Server

建立框架

- 建立新腳本PhotonMover並加入以下內容

```
using UnityEngine;
using System.Collections;

public class PhotonMover : Photon.MonoBehaviour {
    void Update () {
        if (PhotonNetwork.connectionStateDetailed != PeerState.Joined) {
                return;
        }
        if (PhotonNetwork.isMasterClient) { // 若是Master則處理遊戲邏輯
        }
        else {                              // 若是純客端則處理顯示
        }
    }
}
```

利用RPC(遠端程序呼叫)傳值

建立變數

- 建立變數以存放移動後的角度及位置

```
private Vector3 targetPos;
private Vector3 targetRot;

void Start() {
    targetPos = this.transform.position;
    targetRot = this.transform.eulerAngles;
}
```

建立RPC方法

- 加入一個RPC方法，以用作傳值及接收用

```
[RPC]
void SetStatus(Vector3 newPos, Vector3 newRot)
{
    targetPos = newPos;
    targetRot = newRot;
}
```

加入Master及客端處理

- 修改 Update 的內容

```
void Update () {
    float moveSpeed = 3;
    if (PhotonNetwork.connectionStateDetailed != PeerState.Joined) {
            return;
        }

        if (PhotonNetwork.isMasterClient) { // 若是Master則處理遊戲邏輯
            photonView.RPC ("SetStatus", PhotonTargets.Others,
transform.position, transform.eulerAngles);
        }
        else {                                  // 若是純客端則處理顯示
            this.transform.position = Vector3.Lerp (transform.position,
targetPos, Time.deltaTime * moveSpeed);
            this.transform.eulerAngles = targetRot;
        }
    }
```

發送RPC的指令

- photonView.RPC ("方法名稱", PhotonTargets, 參數...);
- PhotonTargets：指定傳輸的目標，Other代表傳給不包含自己的其他人

建立好可以移動的角色

- 可利用Unity的內建腳本建立好可以控制的角色
- 為角色加上PhotonView腳本，此腳本為於PhotonNetwork/PhotonView
- 為角色加入自已寫的PhotonMover

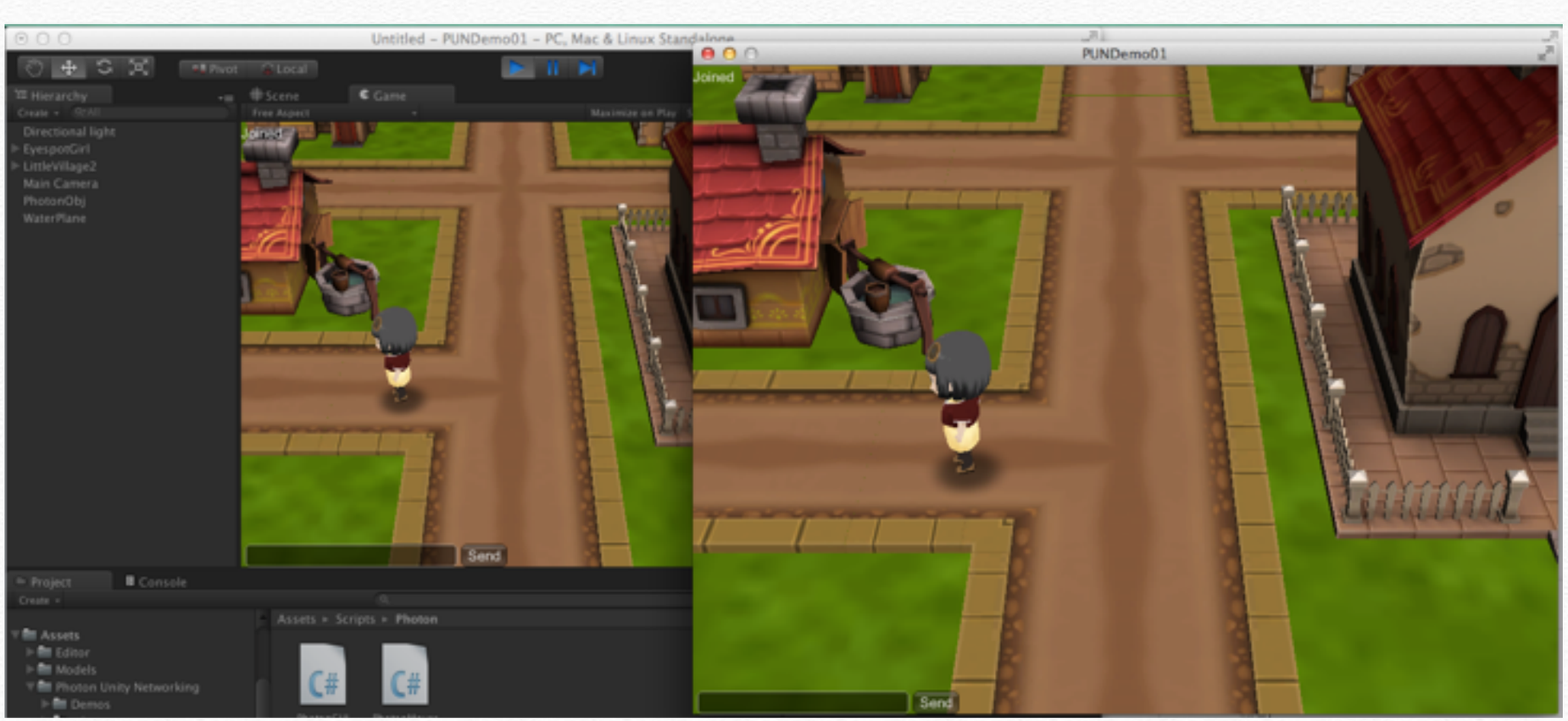
登入完成後才能控制

- 取得角色控制腳本，待登入完成且身為Master才將腳本的enabled設為true

```
public class PhotonMover : Photon.MonoBehaviour {
    . . . . . .
    private RayHitController rayController;

    void Start() {
        . . . . . .
        rayController = this.GetComponent<RayHitController> ();
        rayController.enabled = false;
    }

    void Update () {

        . . . . . .

        if (PhotonNetwork.isMasterClient) { // 若是Master則處理遊戲邏輯
            if( !rayController.enabled )
            {
                rayController.enabled = true;
            }
            photonView.RPC ("SetStatus", PhotonTargets.Others,
transform.position, transform.eulerAngles);
        }
```

編譯後測試

- 編譯後執行即可看到可以順利傳輸，因為沒有傳動畫參數因此角色只能移動無法播動畫，只有先進入的可以控制(因為沒有廣播動畫代號因此角色不會有動作)



利用Serialization傳輸

建立新的腳本

- 建立腳本取名為PhotonMoverStream.cs
- 將原本角色身上的PhotonMover.cs腳本移除

更改繼承

- 更改繼承為Photon.MonoBehaviour

```
public class PhotonMoverStream : Photon.MonoBehaviour {

}
```

加入變數

- 加入以下的變數

```
    private Vector3 correctPlayerPos = Vector3.zero; // 移動後的位置
    private Vector3 correctPlayerRot = Vector3.zero; // 移動後的方向

    private RayHitController rayController;
    private string sendAniName = "";          // 發送的動畫
    private string receiveAniName = "";          // 接收的動畫
```

從Awake取得角色控制腳本

- 取得角色的控制腳本以取得動畫名稱，作為播放動畫用

```
// Use this for initialization
void Awake () {
    correctPlayerPos = this.transform.position;
    correctPlayerRot = this.transform.eulerAngles;
    rayController = this.GetComponent<RayHitController> ();
    rayController.enabled = false;
}
```

加入串流傳輸方法

- 加入OnPhotonSerializeView方法，利用SendNext傳輸資料，用ReceiveNext接收資料，傳輸及接收需對應

```
void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
    if (stream.isWriting) {
        stream.SendNext(transform.position);
        stream.SendNext(transform.eulerAngles);
        stream.SendNext(sendAniName);
    }
    else {
        correctPlayerPos = (Vector3)stream.ReceiveNext();
        correctPlayerRot = (Vector3)stream.ReceiveNext();
        receiveAniName = (string)stream.ReceiveNext();
    }
}
```

加入Update

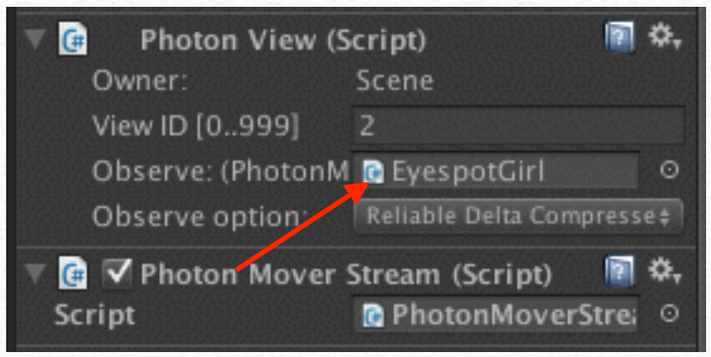
- 在Update裡移動角色及播放動畫

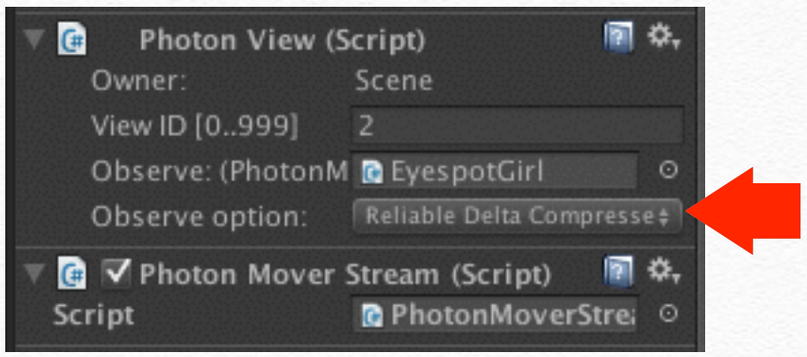
```
void Update () {
    float moveSpeed = 3;
    if (PhotonNetwork.connectionStateDetailed != PeerState.Joined) {
        return;
    }

    if (PhotonNetwork.isMasterClient) { // 若是Master則處理遊戲邏輯
        if( !rayController.enabled )
        {
            rayController.enabled = true;
        }
        sendAniName = rayController.CharacterAniName;
    }
    else {                                  // 若是純客端則處理顯示
        this.transform.position = Vector3.Lerp (transform.position,
correctPlayerPos, Time.deltaTime * moveSpeed);
        this.transform.eulerAngles = correctPlayerRot;

        if( receiveAniName.Length > 0 )
        {
            this.animation.Play (receiveAniName);
        }

    }

}
```

將腳本加到角色身上

- 將PhotonMoverStream拉到角色上
- 到Inspector將PhotonMoverStream拉到PhotonView的Observe

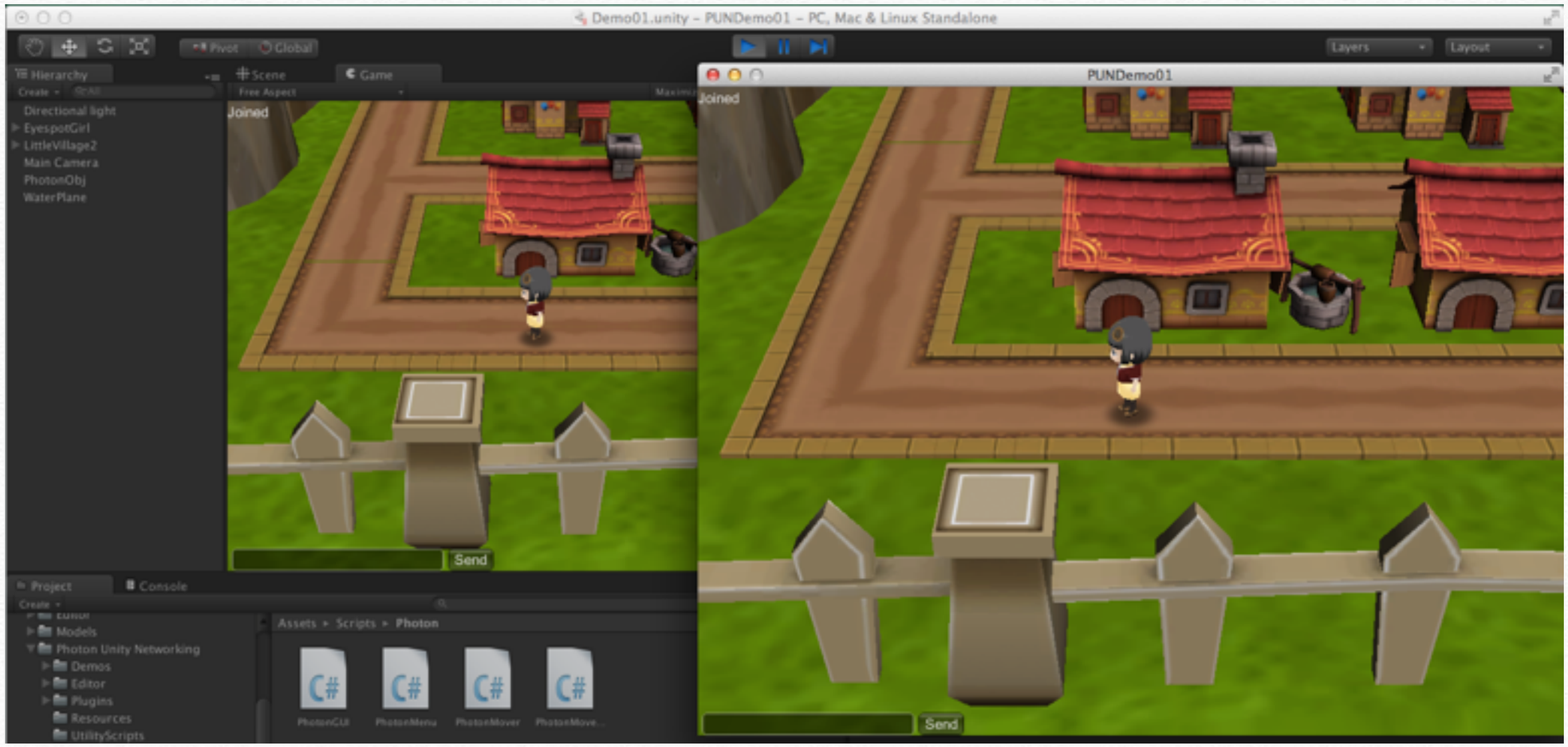
- 將Observe option改成Reliable Delta Compressed





測試

- 編譯後執行


Demo01.unity – PUNDemo01 – PC, Mac & Linux Standalone
Hierarchy
Create
Directional light
EyespotGirl
LittleVillage2
Main Camera
PhotonObj
WaterPlane
Scene
Game
Free Aspect
Pivot
Global
Layers
Layout
Joined
Send
PUNDemo01
Joined
Send
Project
Console
Create
Assets ▸ Scripts ▸ Photon
Models
Photon Unity Networking
Demos
Editor
Plugins
Resources
UtilityScripts
PhotonCtl
PhotonMenu
PhotonMover

RPC vs Serialization

RPC和Serialization的比較

- RPC的運作方式為呼叫遠端的程式
- Stream的作用則為 主->客 的廣播方式
- Stream效能較高但彈性較小，適合廣播角色移動資訊之類
- RPC必須自行發送，Stream的傳輸為全自動，因此遊戲邏輯會使用RPC傳輸

End