

# Photon Cloud(四)

## 自訂房間及建立物件探討

主講：紀曲峰

# 遊戲房間的運用

- ❖ PUN的遊戲房間由Client端建立，但是在Server端進行管理的
- ❖ 大部份的休閒遊戲都會限制一局的人數，對於Master的負擔也會比較輕
- ❖ 建立房間後可以指定加入某房間，亦可隨機加入房間

# 自訂遊戲房間

# 建立腳本

- ❖ 建立新腳本PhotonRoomMenu，設置繼承自Photon.MonoBehaviour

```
public class PhotonRoomMenu : Photon.MonoBehaviour {  
  
}
```

# 加入變數

## ❖ 加入建立房間或加入房間需要的變數

```
private string roomName = ""; // 房間名稱  
private string playerName = ""; // 玩家暱稱  
private string ErrorMessage = ""; // 錯誤訊息  
private int roomSel = 0; // 選取現有房間
```

# 啟動時連線

## ❖ 加入Awake時自動連線

```
void Awake()  
{  
    PhotonNetwork.ConnectUsingSettings("1.0");  
}
```

# 建立OnGUI框架

❖ 若尚未連線則顯示連線按鈕讓玩家重新連線

```
void OnGUI()
{
    GUILayout.Label("Connection status: " +
PhotonNetwork.connectionStateDetailed);

    if (!PhotonNetwork.connected && !PhotonNetwork.connecting) // 若尚未連線
    {
        if (GUILayout.Button("Connect"))
        {
            PhotonNetwork.ConnectUsingSettings("1.0"); // 連線
        }
    }
    else
    {
        // 這裡要處理開啟房間和加入房間的內容
    }

    if (ErrorMessage.Length > 0)
        GUILayout.Label (ErrorMessage);
}
```

# 玩家取暱稱的部份

- ❖ 不管是自建房間或加入現有房間都需要取一個暱稱，所以一開始先加入讓玩家取暱稱的內容

```
// 這裡要處理開啟房間和加入房間的內容

// 玩家取暱稱
GUILayout.Label("Player name :");
playerName = GUILayout.TextField(playerName,
                                GUILayout.Width(200));

// 玩家建立房間
. . . . .

// 加入現有房間
. . . . .
```



# 玩家建立房間

- ❖ 若玩家按下建立房間，先判斷暱稱和房間名稱是否都有打，若有打就呼叫CreateRoom建立房間，這裡的數人上限設為4人

```
// 玩家建立房間
GUILayout.Label("Room name :");
roomName = GUILayout.TextField(roomName, GUILayout.Width(200));
if (GUILayout.Button("Create And Join Room"))
{
    ErrorMessage = "";
    if( playerName.Length > 0 && roomName.Length > 0)
    {
        PhotonNetwork.playerName = playerName;

        PhotonNetwork.CreateRoom(roomName, new RoomOptions()
{ maxPlayers = 4 }, null);
    }
    else
    {
        ErrorMessage = "You must input playername and roomname.";
    }
}
```

# 取得及繪製房間列表供玩家選取

- ❖ 利用GetRoomList取得房間列表後，轉成字串陣列，再用SelectionGrid繪製選取用的按鈕
- ❖ 最後加上進入房間的按鈕

```
// 加入現有房間
// 取得房間列表
RoomInfo[] roomInfo = PhotonNetwork.GetRoomList();
if( roomInfo.Length > 0 )
{
    // 將所有房間列表轉成字串陣列
    string[] roomNames = new string[roomInfo.Length];
    for( int i=0; i<roomInfo.Length; i++ )
    {
        roomNames[i] = roomInfo[i].name;
    }

    // 繪製列表選取按鈕
    roomSel = GUILayout.SelectionGrid(roomSel, roomNames, 1,
    GUILayout.Width(200));

    if (GUILayout.Button("Join Room")) // 若加入房間
    {
    }
}
}
```

# 若按下加入房間

- ❖ 若按下加入房間，先判斷是否有設定暱稱，若有呼叫「JoinRoom(房間名稱)」加入房間

```
if (GUILayout.Button("Join Room")) // 若加入房間
{
    ErrorMessage = "";
    if( playerName.Length > 0 )
    {
        PhotonNetwork.playerName = playerName;

        PhotonNetwork.JoinRoom(roomNames[roomSel]);
    }
    else
    {
        ErrorMessage = "You must input playername.";
    }
}
```

# 建立房間完成及加入房間完成的事件

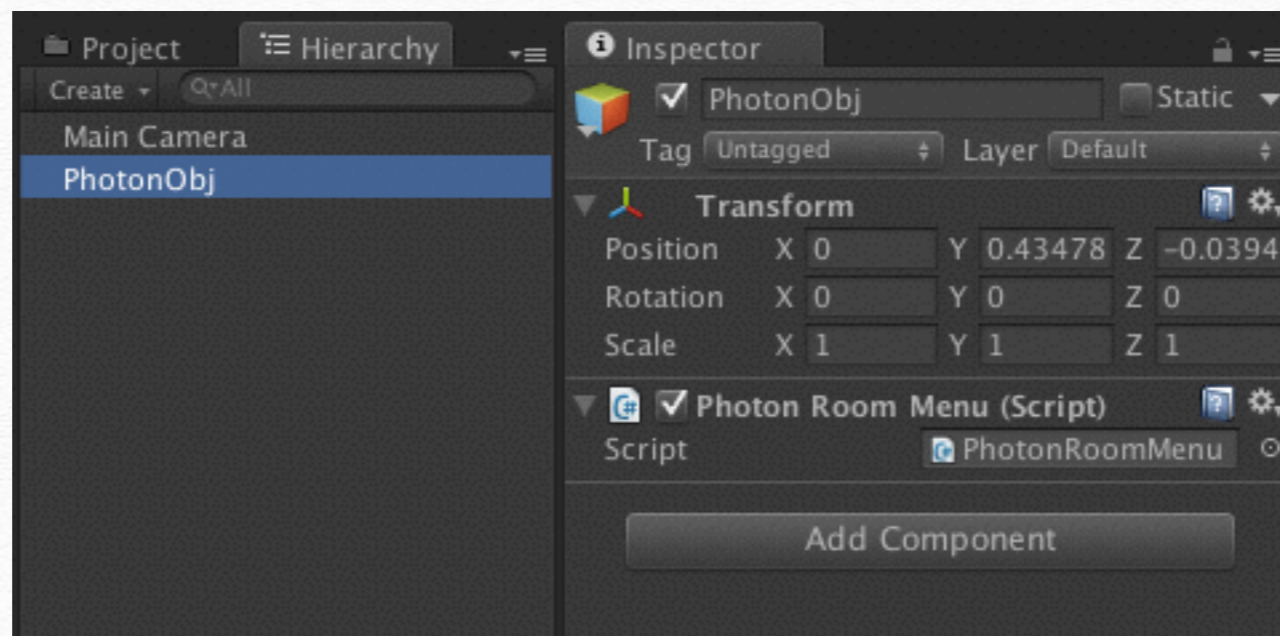
- ❖ 最後加入建立房間完成及進入房間完成的事件，若有成功則呼叫LoadLevel跳到遊戲舞場場景
- ❖ 建立房間會自動加入，不需要再重新加入房間

```
public void OnJoinedRoom()
{
    Debug.Log("OnJoinedRoom");
    PhotonNetwork.LoadLevel("Stage01");
}

public void OnCreatedRoom()
{
    Debug.Log("OnCreatedRoom");
    PhotonNetwork.LoadLevel("Stage01");
}
```

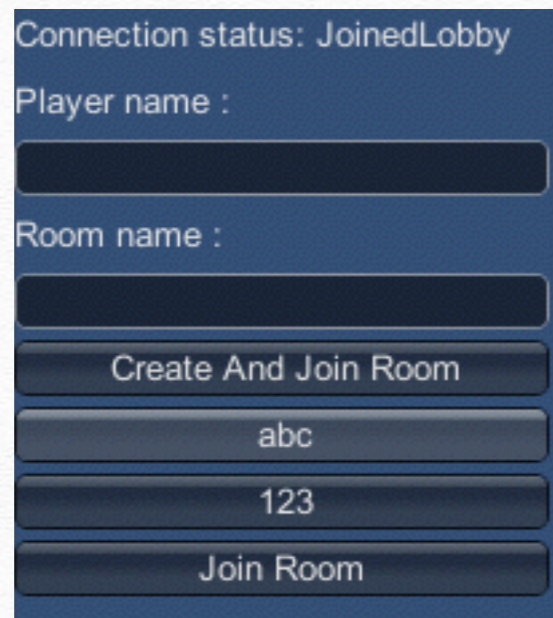
# 將腳本加到場景中

- ❖ 選取PhotonObj，把原本的腳本全部移除，加入剛才寫的PhotonRoomMenu腳本



# 執行遊戲

- ❖ 執行遊戲，若有建立房間則會出現已建立房間的列表，否則只會有建立房間的按鈕



Connection status: JoinedLobby

Player name :

Room name :

Create And Join Room

abc

123

Join Room

# 利用Photon建立物件

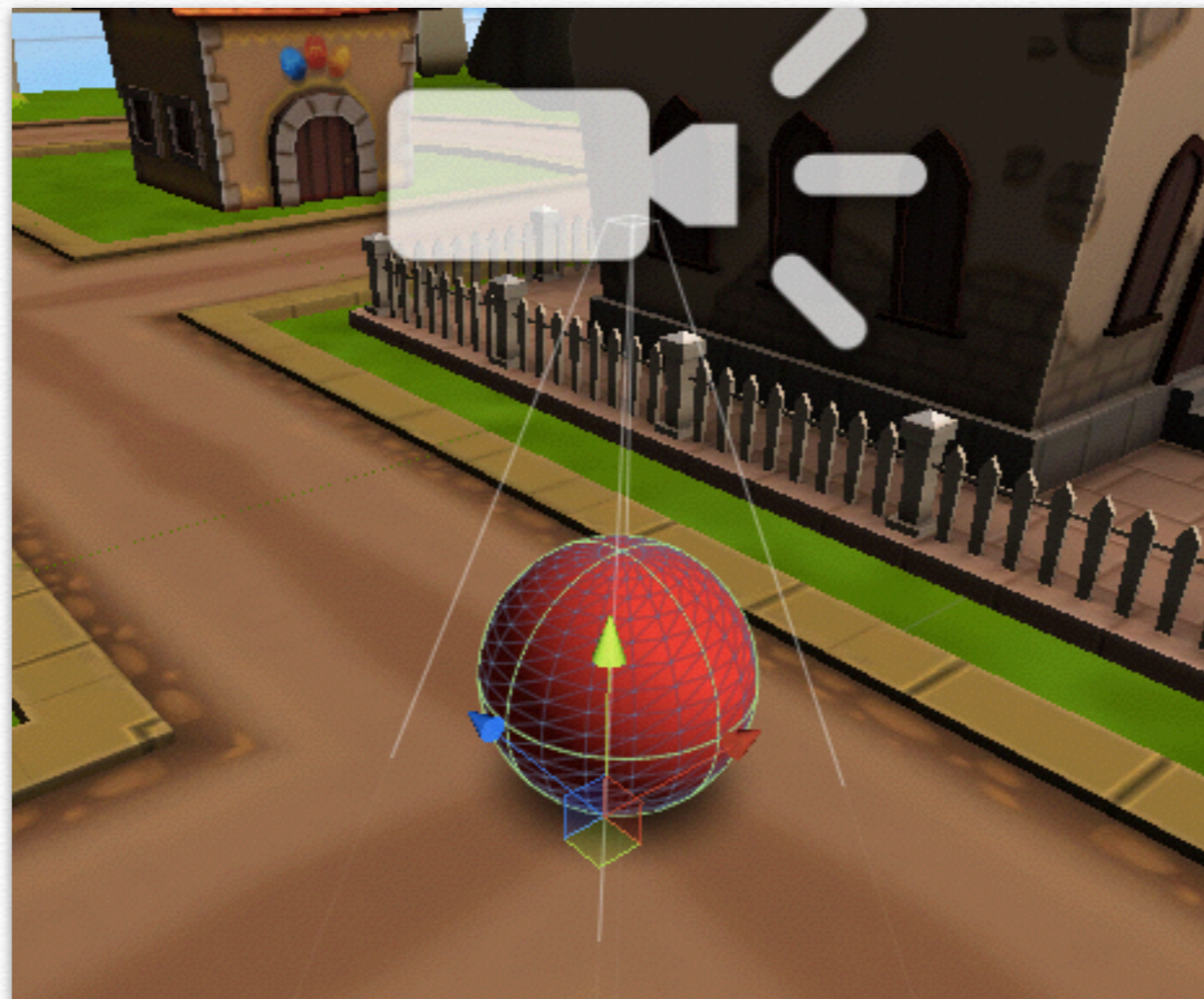
# 利用Photon管理動態物件

- ❖ 我們可以利用Photon簡單的建立物件，Photon可以幫我們管理所有動態建立之物件
- ❖ 並不是所有物件都適合使用Photon幫我們管理，以下便會進行此問題之探討



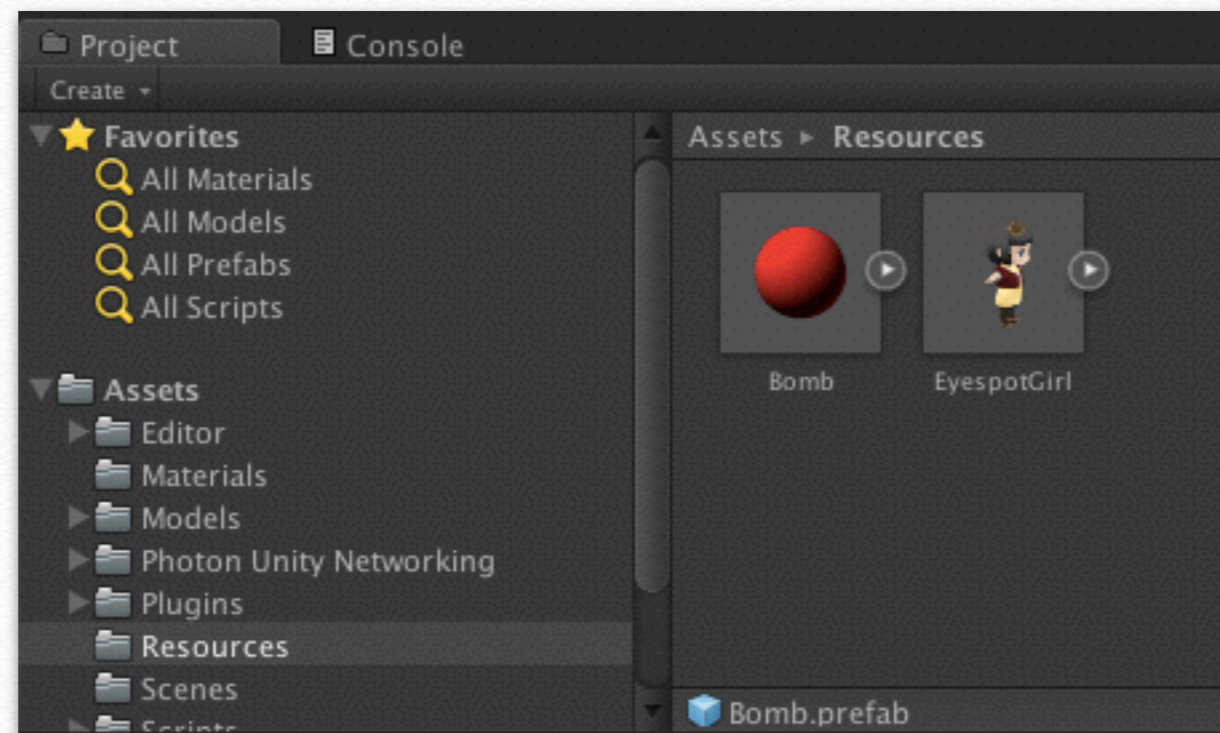
# 建立動態產生的物件

- ❖ 利用內建功能建立一顆球當作炸彈，也可以自行製作其他物件代替



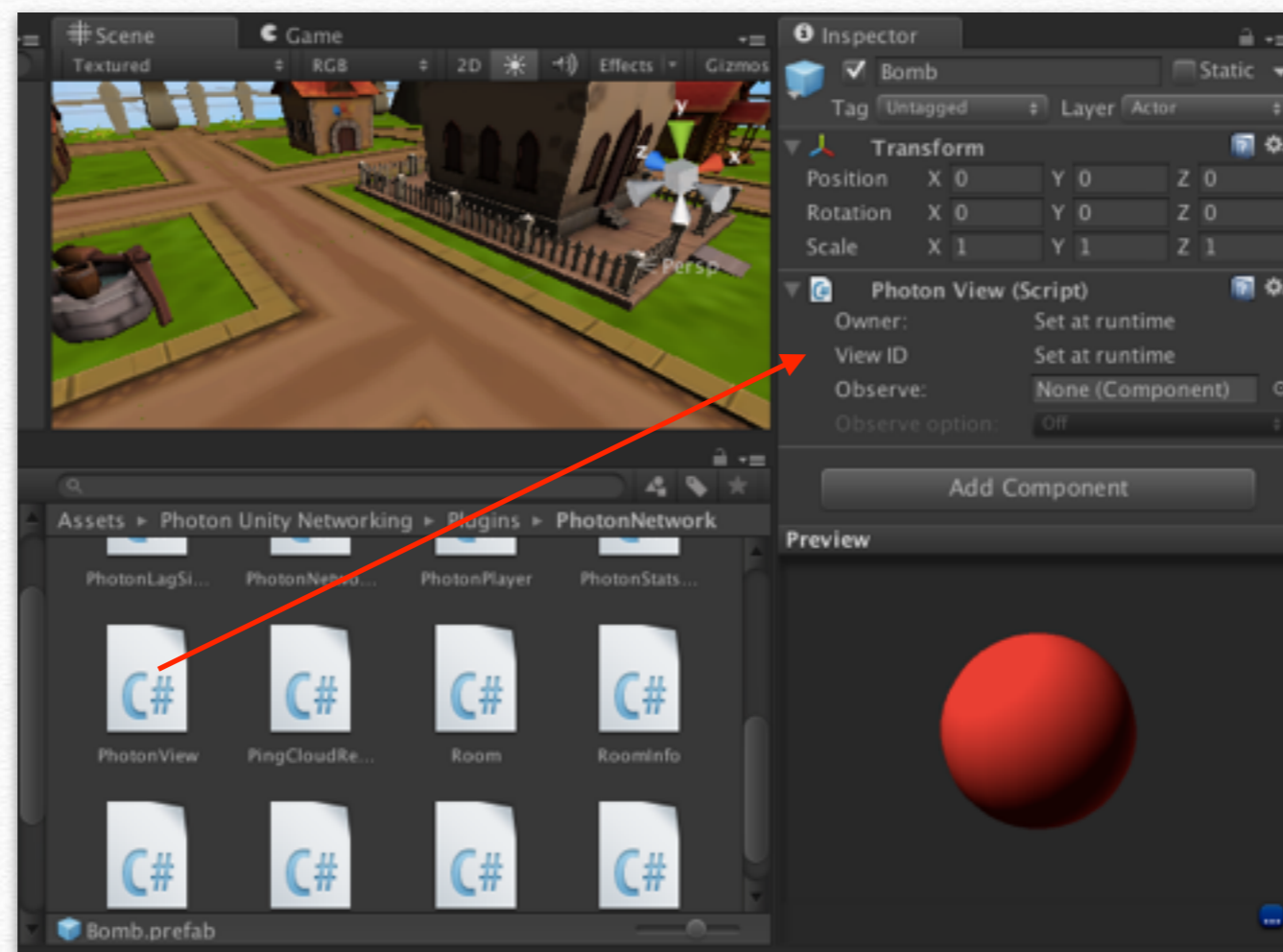
# 將炸彈做成Prefab

- ❖ 到Resources資料夾內建立一個Prefab，取名為Bomb，然後將做好的模型拉進去，完成後可刪除場景中的炸彈



# 為炸彈加入PhotonView

- ❖ 若要讓Photon自動維護物件，必須加入PhotonView腳本，請將PhotonView加到Prefab上



# 加入建立炸彈的腳本

- ❖ 建立腳本PhotonInstantiateBomb.cs，並加入以下的程式碼，必須為本人角色才能放炸彈，因此有判斷 `!photonView.isMine`

```
using UnityEngine;
using System.Collections;

public class PhotonInstantiateBomb : Photon.MonoBehaviour {

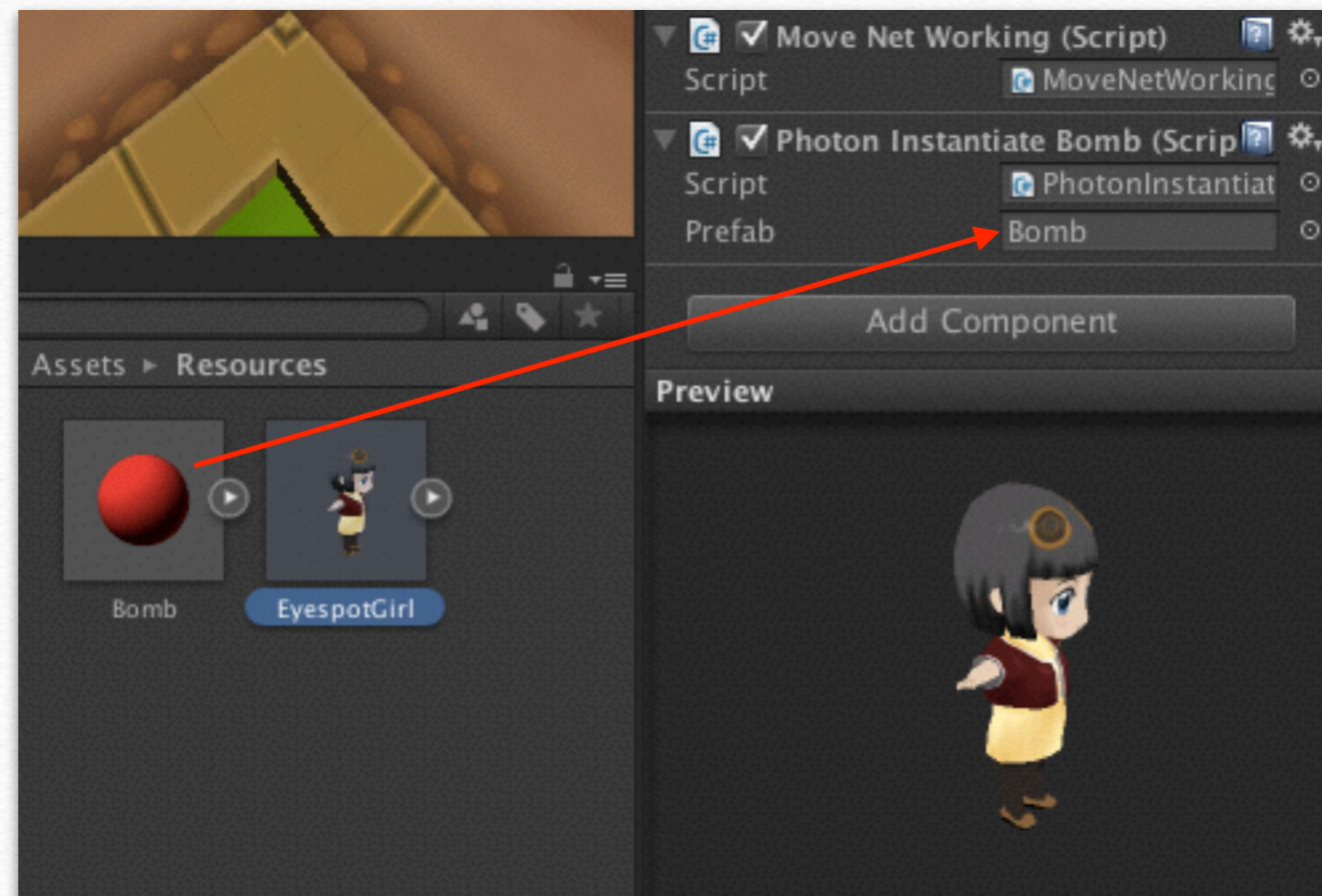
    public GameObject Prefab;

    void Update () {
        if ( PhotonNetwork.connectionStateDetailed != PeerState.Joined || !
photonView.isMine)
        {
            return;
        }

        if (Input.GetMouseButtonDown (1)) {
            PhotonNetwork.Instantiate (Prefab.name, this.transform.position,
Quaternion.identity, 0);
        }
    }
}
```

# 將腳本加到角色身上

- ❖ 將腳本加到角色的Prefab上，並將炸彈拉到變數內



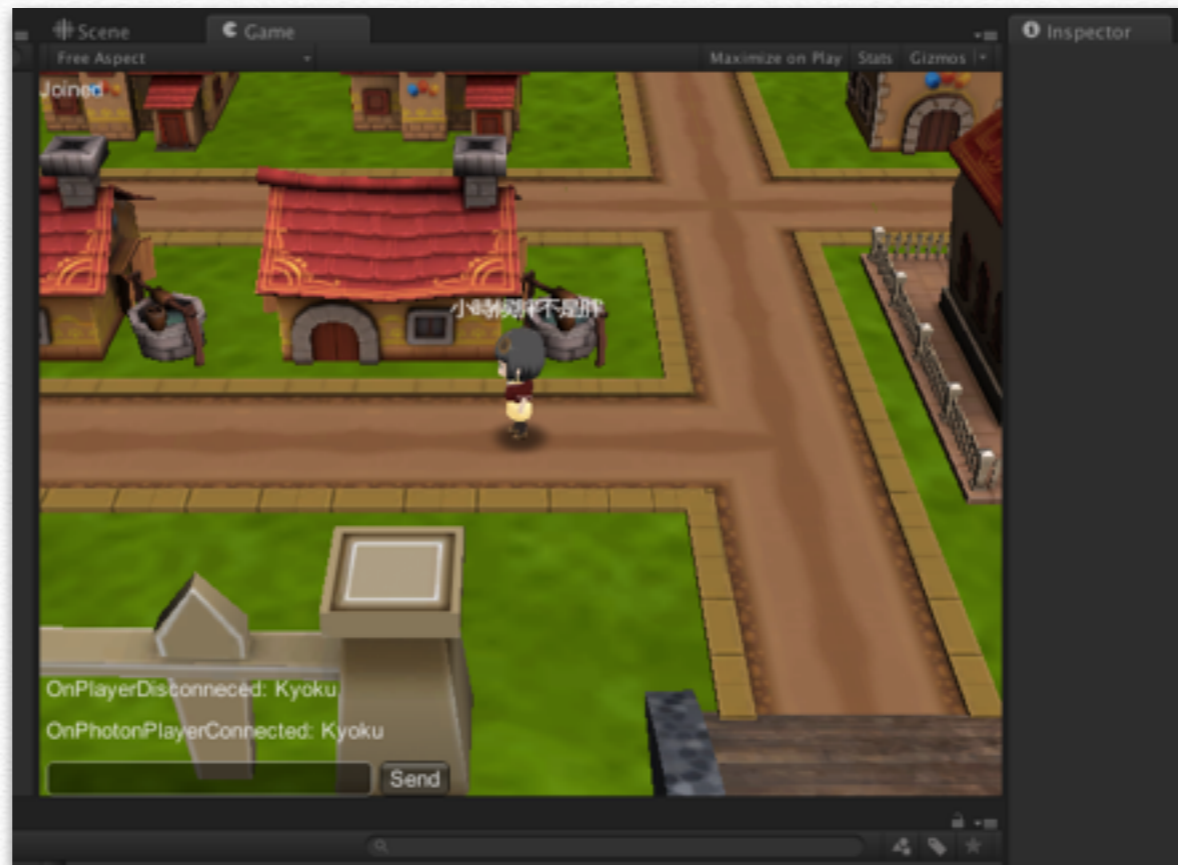
# 測試

- ❖ 執行後按滑鼠右鍵會動態建立炸工彈，而且雙方都看得到



# 玩家離線

- ❖ 一旦玩家離線後，由該玩家動態建立的物件也不見了，因此像炸彈這種東西並不適合由Photon維護



利用RPC傳輸後由Client  
自行建立物件



# 由Client自行建立炸彈

- ❖ 比較理想的做法是由RPC廣播建立炸彈的命令，再由各Client自行建立炸彈物件，如此一來就不會因為玩家離開炸彈就消失了

# 加入RPC方法

- ❖ 加入放炸彈的RPC方法，傳入炸彈座標後在該處建立炸彈

```
[RPC]
void PutBomb(Vector3 BombPos)
{
    Instantiate (Prefab, BombPos, Quaternion.identity);
}
```

# 修改滑鼠右鍵命令

- ❖ 當按下滑鼠右鍵，先放自己的炸彈，然後以RPC呼叫其他的Client放置炸彈

```
if (Input.GetMouseButtonDown (1)) {  
    //PhotonNetwork.Instantiate (Prefab.name,  
this.transform.position, Quaternion.identity, 0);  
    Instantiate (Prefab, transform.position, Quaternion.identity);  
    photonView.RPC ("PutBomb", PhotonTargets.Others,  
                    transform.position);  
}
```

# 測試

- ❖ 編譯後測試，所有的炸彈都可以完美建立



# 玩家離線

- ❖ 玩家離線後，所有的炸彈都保留下來了



# 怪物的維護

# 前置

- ❖ 由於怪的維護過於複雜，因此這裡改成一旦Master離開所有人即回到登入畫面，若需要接手Master請自行修改
- ❖ 怪物的移動必須使用串流傳輸方法 `OnPhotonSerializeView`，勿使用RPC，以避免過多的傳輸造成Queue溢滿引發 `QueueIncomingReliableWarning` 錯誤

# 修改CharacterInGame

- ❖ 將CharacterInGame的OnMasterClientSwitched事件改成一旦發生即跳回登入畫面

```
public void OnMasterClientSwitched(PhotonPlayer player)
{
    Debug.Log("OnMasterClientSwitched: " + player);

    Application.LoadLevel("MenuStage");
    return;
}
```



# 修改生怪磚

- ❖ 通常生怪磚會使用Instantiate建立角色副本，請改成PhotonNetwork.Instantiate建立，並將類別繼承自Photon.MonoBehaviour，Awake加上若不是Master則不運作

```
[RequireComponent(typeof(PhotonView))]  
  
public class EnemyNidus : Photon.MonoBehaviour {  
  
    void Awake () {  
        if( PhotonNetwork.isMasterClient ) { // 只有Master可以生怪  
            this.enabled = true;  
        }  
        else {  
            this.enabled = false;  
        }  
    }  
  
    . . . . .  
    // GameObject enemy = (GameObject)Instantiate(enemyObj[enemyNum],  
this.transform.position, Quaternion.identity);  
    GameObject enemy =  
        (GameObject)PhotonNetwork.Instantiate(enemyObj.name,  
        this.transform.position, Quaternion.identity, 0);  
  
    . . . . .  
}
```

# 怪的行為

- ❖ 建立一個類別PhotonEnemyBehaviour，並繼承自Photon.MonoBehaviour

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(PhotonView))]
[RequireComponent(typeof(EnemyBase))]

public class PhotonEnemyBehaviour : Photon.MonoBehaviour {
}
```

# 加入變數

- ❖ 使用的變數和主角很像，EnemyBase是敵物原本的行程腳本，而characterName存的不是暱稱，而是photonView.viewID

```
private EnemyBase enemyBase;

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

private string sendAniName = ""; // 發送的動畫
private string receiveAniName = ""; // 接收的動畫
private float sendAniSpeed = 1f;
private float receiveAniSpeed = 1f;

private Vector3 namePosition;
private string characterName; // 角色ID
```

# Awake事件

- ❖ 在Awake加入若不是Master則原本的角色行為不運作，只有Master負責進行敵物角色行為運算，暱稱使用ViewID代替以檢視ViewID內容，完成後可移除敵物暱稱

```
void Awake () {  
  
    enemyBase = GetComponent<EnemyBase> ();  
  
    if( PhotonNetwork.isMasterClient ) {  
        enemyBase.enabled = true;  
    }  
    else {  
        enemyBase.enabled = false;  
    }  
  
    characterName = photonView.viewID.ToString();  
}
```

# 串流通訊的行為

- ❖ 加入串流通訊行為，因為怪物數量通常很多，勿使用RPC以免造成Queue溢滿

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

# Update內容

- ❖ Update和主角的控制上基本一樣，只多了設定動畫速度，若敵物沒有此屬性可自行修改

```
void Update () {  
    float moveSpeed = 5;  
    if (PhotonNetwork.isMasterClient)  
    {  
        sendAniName = enemyBase.CharacterAniName;  
        sendAniSpeed = enemyBase.AnimationSpeed;  
    }  
    else  
    {  
        transform.position = Vector3.Lerp(transform.position,  
correctPlayerPos, Time.deltaTime * moveSpeed);  
        transform.eulerAngles = correctPlayerRot;  
        if( receiveAniName.Length > 0 )  
        {  
            this.animation[receiveAniName].speed = receiveAniSpeed;  
            this.animation.Play (receiveAniName);  
        }  
    }  
  
    namePosition = Camera.main.WorldToScreenPoint (new  
Vector3(this.transform.position.x, this.transform.position.y+1f,  
this.transform.position.z));  
}
```

# 顯示ViewID

- ❖ 在OnGUI加入暱稱顯示，以檢視敵物的ID編號

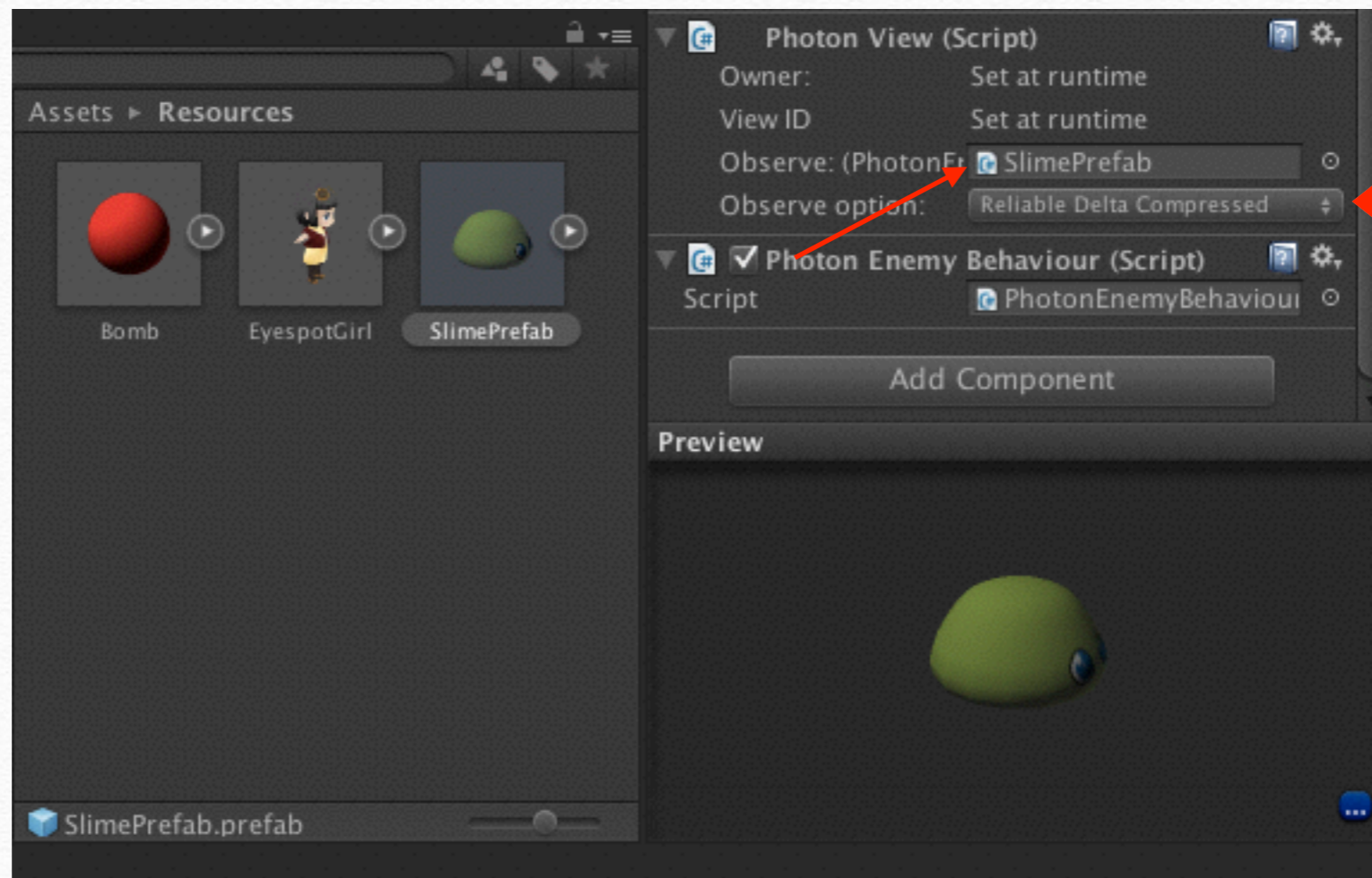
```
void OnGUI () {
    GUIStyle LabelStyle = new GUIStyle(GUI.skin.label);

    LabelStyle.alignment = TextAnchor.MiddleCenter;
    LabelStyle.fontStyle = FontStyle.Bold;
    LabelStyle.normal.textColor = Color.white;

    GUI.Label (new Rect (namePosition.x - 50, Screen.height -
namePosition.y, 100, 20), characterName, LabelStyle);
}
```

# 將腳本加入Prefab

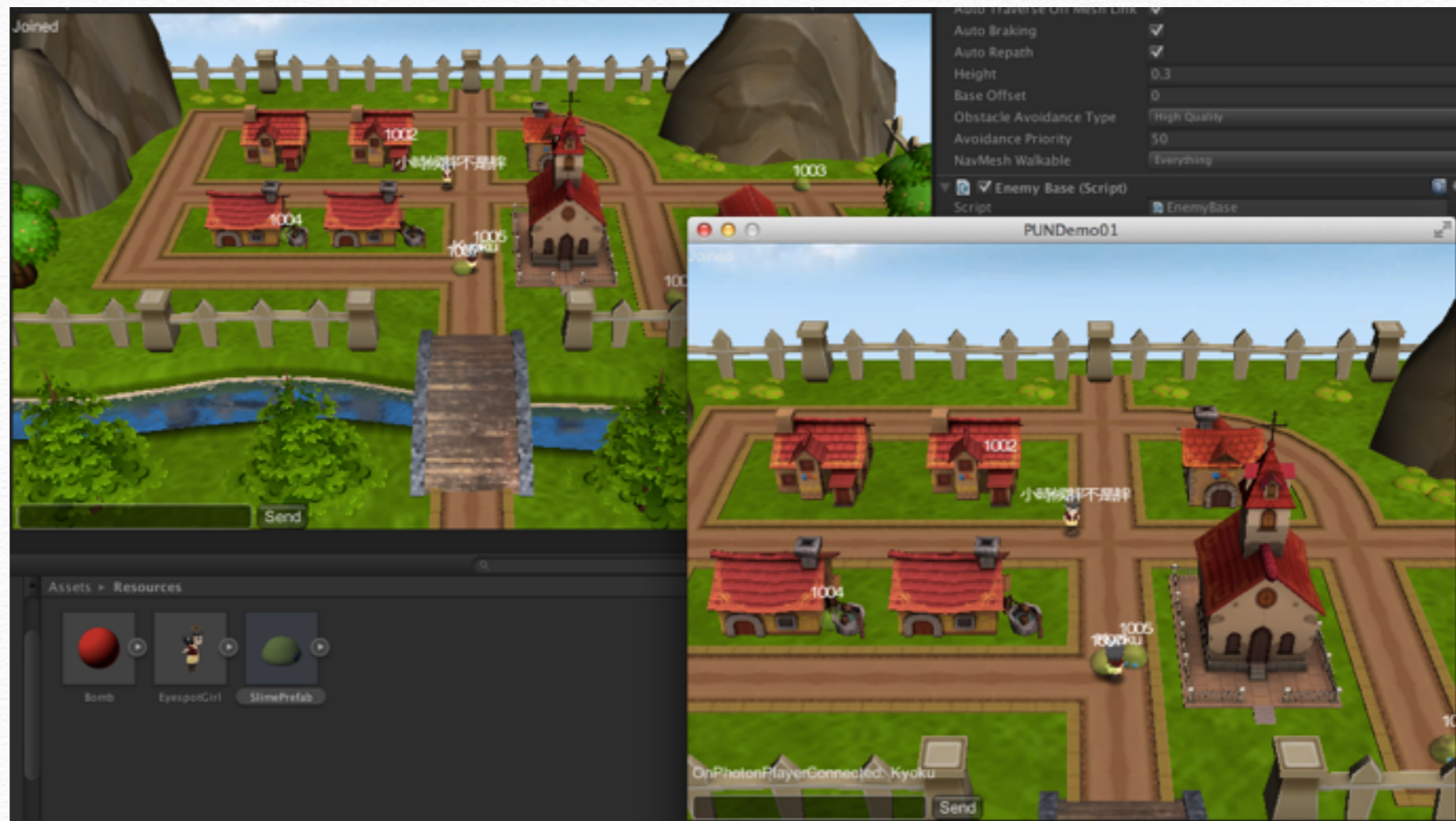
- ❖ 請將敵物做成Prefab並放在Resources資料夾內
- ❖ 參考主角的加入方式將腳本加到敵物Prefab上





# 執行測試

- ❖ 執行後可以看到所有的敵物都同步了，而且ViewID也都有同步



# 關於敵物的攻擊或死亡

- ❖ 通常敵物的攻擊行為由Master判斷
- ❖ 當敵物有發生被攻擊行為時利用RPC將ViewID傳回Master扣血
- ❖ 當敵物死亡時利用RPC將ViewID傳回Master進行物件的消滅

End