

Commercial Server Solutions

Photon Unity Network

Arthur Pai

Photon

- Photon Cloud (SaaS)
 - Photon Cloud是 Exit Games 公司開發的雲端遊戲伺服器，提供了便宜、快速、容易使用、支援市面上常見多種平台的線上遊戲伺服器服務
 - 效能高
 - 支援平台多
 - 不敷使用時可以無痛轉移至Photon Server
- Photon Server (On Premise)
 - 可以跑在 Windows 上的服務或程式
 - 是個 Socket Server, 提供了 Reliable UDP, TCP, HTTP 及 WebSockets 的網路傳輸協定, 能夠讓我們任意的跨平台開發網路相關即時連線的程式, 尤其是多人連線遊戲.
 - 可以架在 Google Cloud, Azure VMs, Amazon EC2 或是自己家裡公司的 Local Server 上



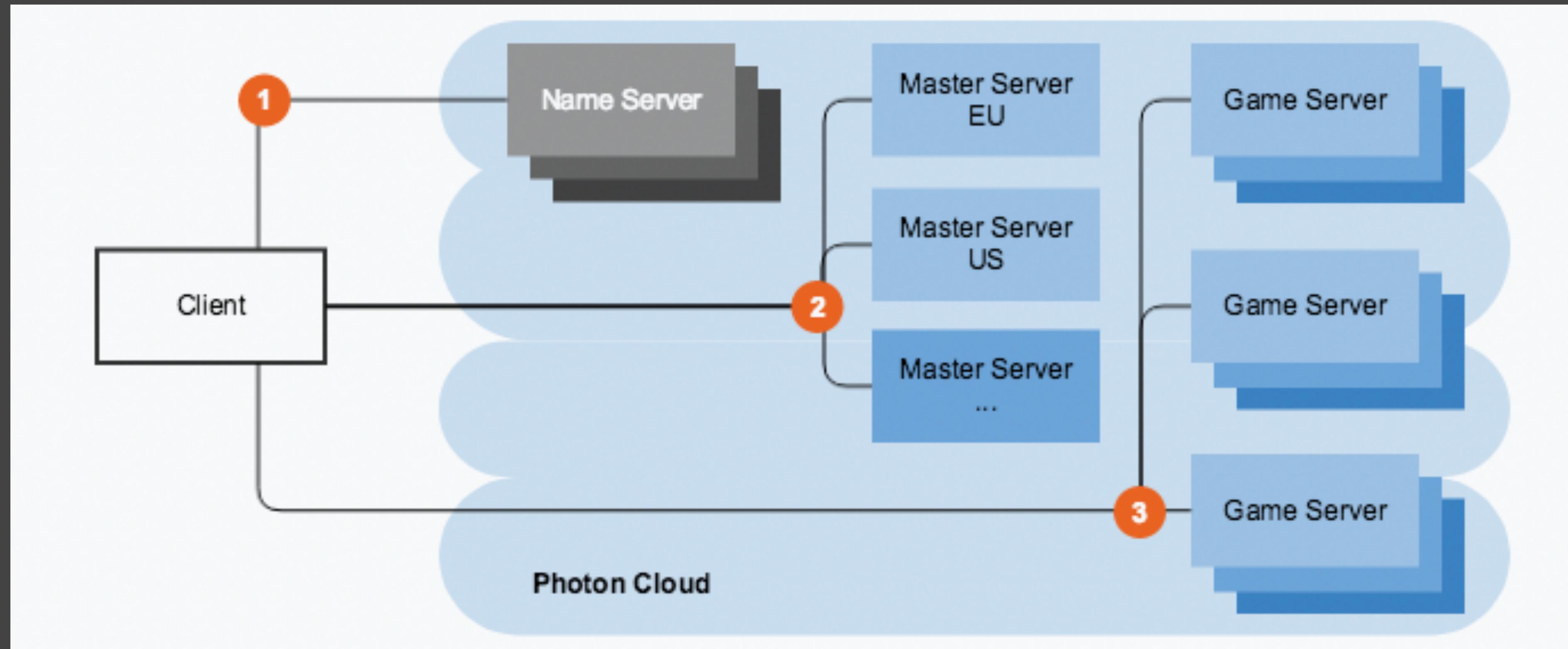
<https://www.photonengine.com/en-US/Photon>

Photon Realtime

	Photon Cloud	Photon Server
伺服器管理	不須管理伺服器	須自行管理伺服器
Scalability 可擴充性	Photon Cloud會自動根據玩家數量調整，並做負載平衡	要自行處理
Game Logic	提供現成的多人遊戲邏輯 像是 Photon Realtime , Photon Chat	可以使用C#擇寫需要的遊戲邏輯
Start instantly	註冊後直接使用	下載後，在自家主機上執行 start your Photon Server in less than 5 minutes
Licensing	Photon Realtime has a free plan for up to 20 CCU.	Photon Server is available with a free license for up to 100 CCU.

<https://doc.photonengine.com/en-us/realtime/current/getting-started/onpremises-or-saas>

Photon Realtime



<https://doc.photonengine.com/zh-tw/pun/current/connection-and-authentication/regions>

Photon Quantum

超高效能與高精度同步，適合各類型節奏快速的遊戲
能用於MOBA、RTS、橫向捲軸動作、運動、格鬥、以及其他相似形態的遊戲



<https://www.photonengine.com/zh-TW/Quantum>



Photon Fusion

The Benchmark for Multiplayer FPS/TPS Games

Fusion

Highly optimized:
High player counts, tick rates & performance.

Game	Year	Players
Valheim	2021	10
CoD Warzone	2020	160
Valorant	2020	10
Fall Guys	2020	60
APEX	2019	60
Fortnite	2017	100
PUBG	2017	100
Overwatch	2016	10
Rocket League	2015	16
Rust	2013	200

Evolution of High-End Multiplayer

Unity 2005

FPS multiplayer concepts that are still state of the art today.

Tribes 1998/2001

Counter Strike 2000

Quake 3 1999

Bolt 2014

PUN 2011

Client Side Prediction
Allows a client to have instant response to his own input despite playing in a server authoritative game.

Full Physics Prediction
Delicate interactions between players, often with complex physical object interactions make use of full physics prediction.

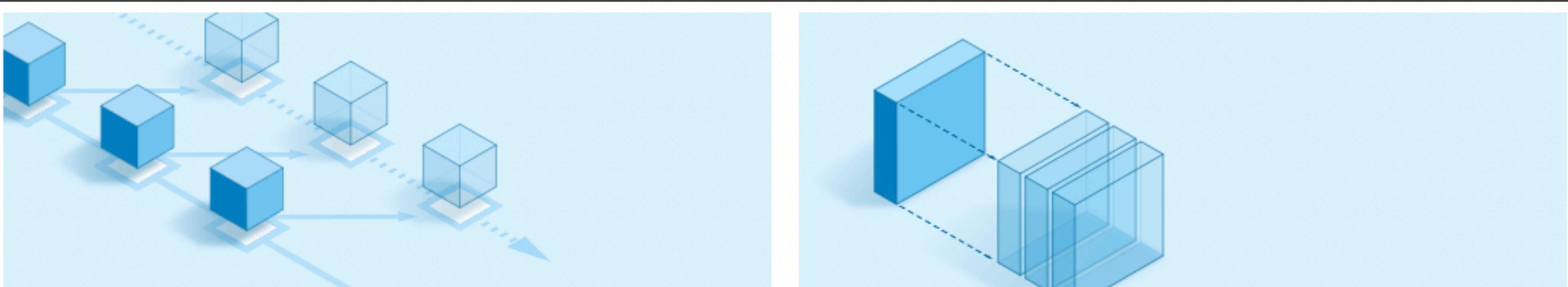
Snapshot Interpolation
Provide smooth visual rendering of remote objects for a client, even during bad networking conditions.

Replication Systems
Fusion implements two best in class algorithms:
Delta Snapshots (2 to 32 players) and **Eventual Consistency** (32+ players).

Lag Compensation
Compensate for the delay between the client firing his weapon and the server registering the potential hit.



Photon Fusion



Tick-based模擬

穩定準確的網路核心功能。客戶端預測和快照插值的基礎。

客戶端預測

即使在高延遲和網路丟失的情況下，也可以在不放棄伺服器權限的情況下讓玩家對自己的輸入做出即時響應。



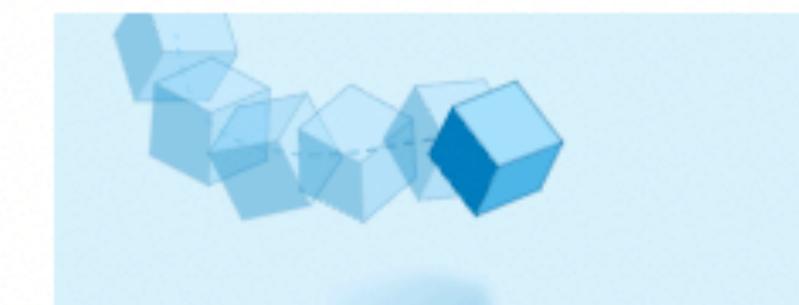
完全物理預測

通過使用完全物理預測，從玩家和物件之間的複雜物理交互中獲得最佳體驗。



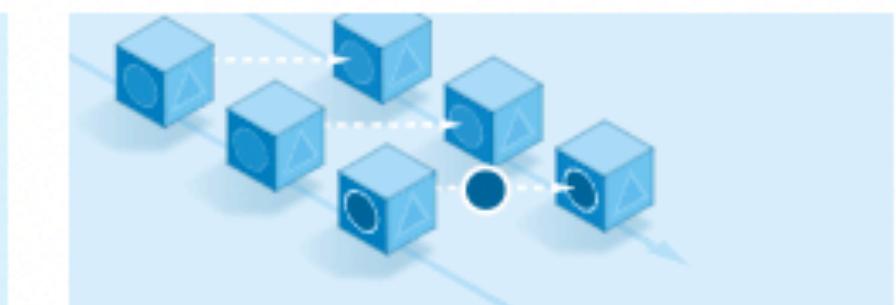
快照插值

即使在網絡不好的條件下，也可以使用自動或自定義插值來實現流暢的視覺渲染。



延遲補償

內置延遲補償 Hitbox，具有易於使用的 API，適用於電子競技級遊戲機制。



同步狀態資料的演算法 慢表系統

Delta 快照和最終一致性與有趣管理(Interest Management)的複製算法的一流實現。

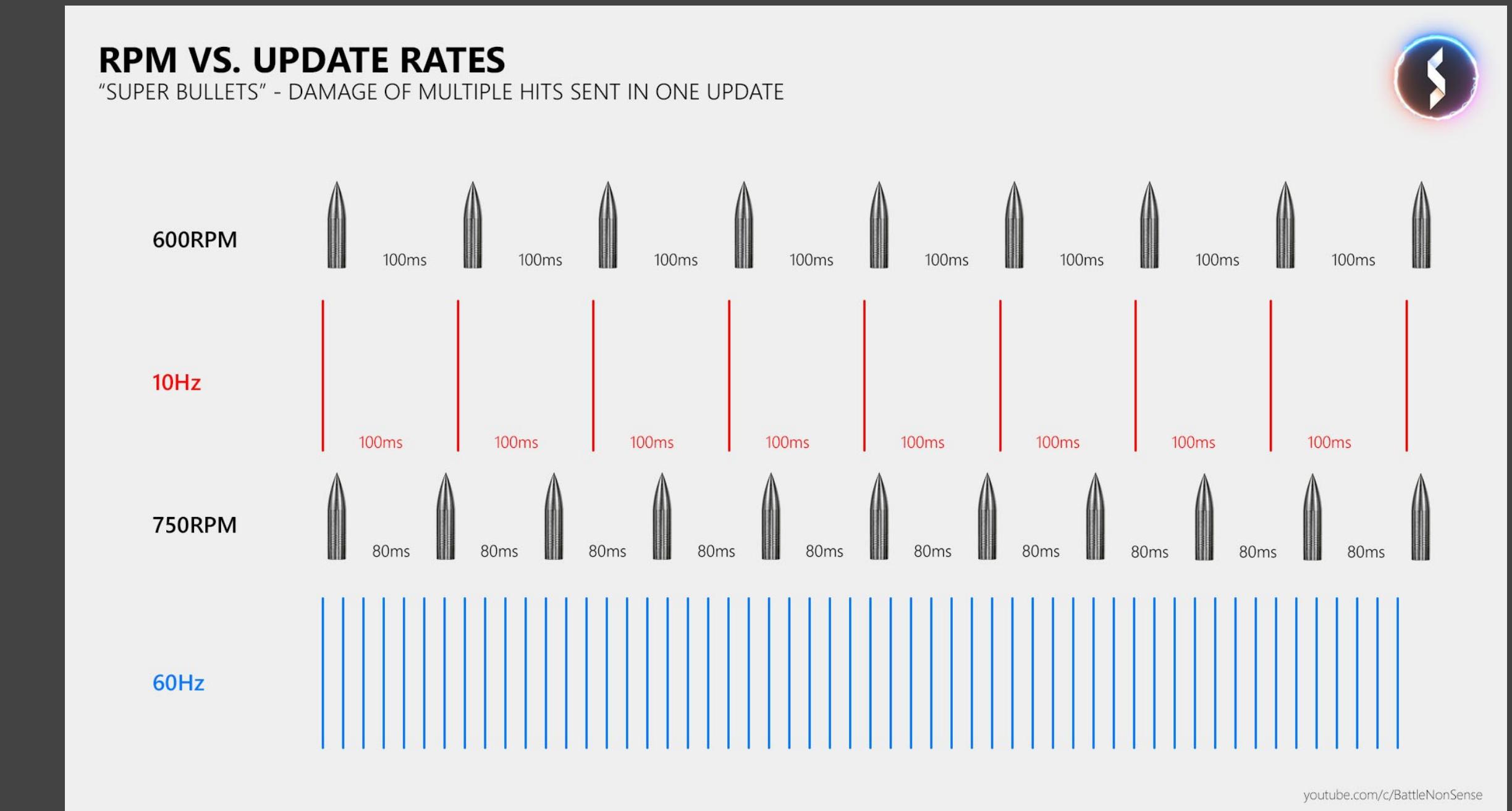
<https://www.photonengine.com/en-US/Fusion>

Tick Rate / Time

- Tick/Simulation Time
 - 收集訊息、事件、資料
 - 處理資料
 - 模擬遊戲世界狀態：場景、物件(Ex. 飛彈)
 - 檢查碰撞：飛彈是否擊中、腳是否踢到其他玩家
 - 玩家狀態：血量、位置、方向等等。
 - 預測玩家位置(如果玩家訊息延遲 Latency/Lag，會發生瞬移，採用內差/外差 插值演算法)
 - AI：狀態、尋路(path-finding)
 - 送出結果
- 60 Hz => $1000/60 \text{ ms} => 16.6666667 \text{ ms}$ / 每次
- 如果一個Tick處理太久
 - 處理不來時，跳過下個Tick
 - 一直持續的話，遊戲就會開始出現延遲
 - Rubber banding、瞬移、命中被拒絕和物理失敗

Super Bullets

- Tick/Simulation Rate
 - Counter-Strike Global Offensive(絕對武力：全球攻勢) : 64/128-tick
 - Valorant(特戰英豪) : 128-tick
 - Dota2 : 64-tick
 - PUBG(絕地求生) : 60-tick
 - Call of Duty(決勝時刻) : 22-tick / 12-tick
 - Overwatch(鬥陣特攻) : 20-tick

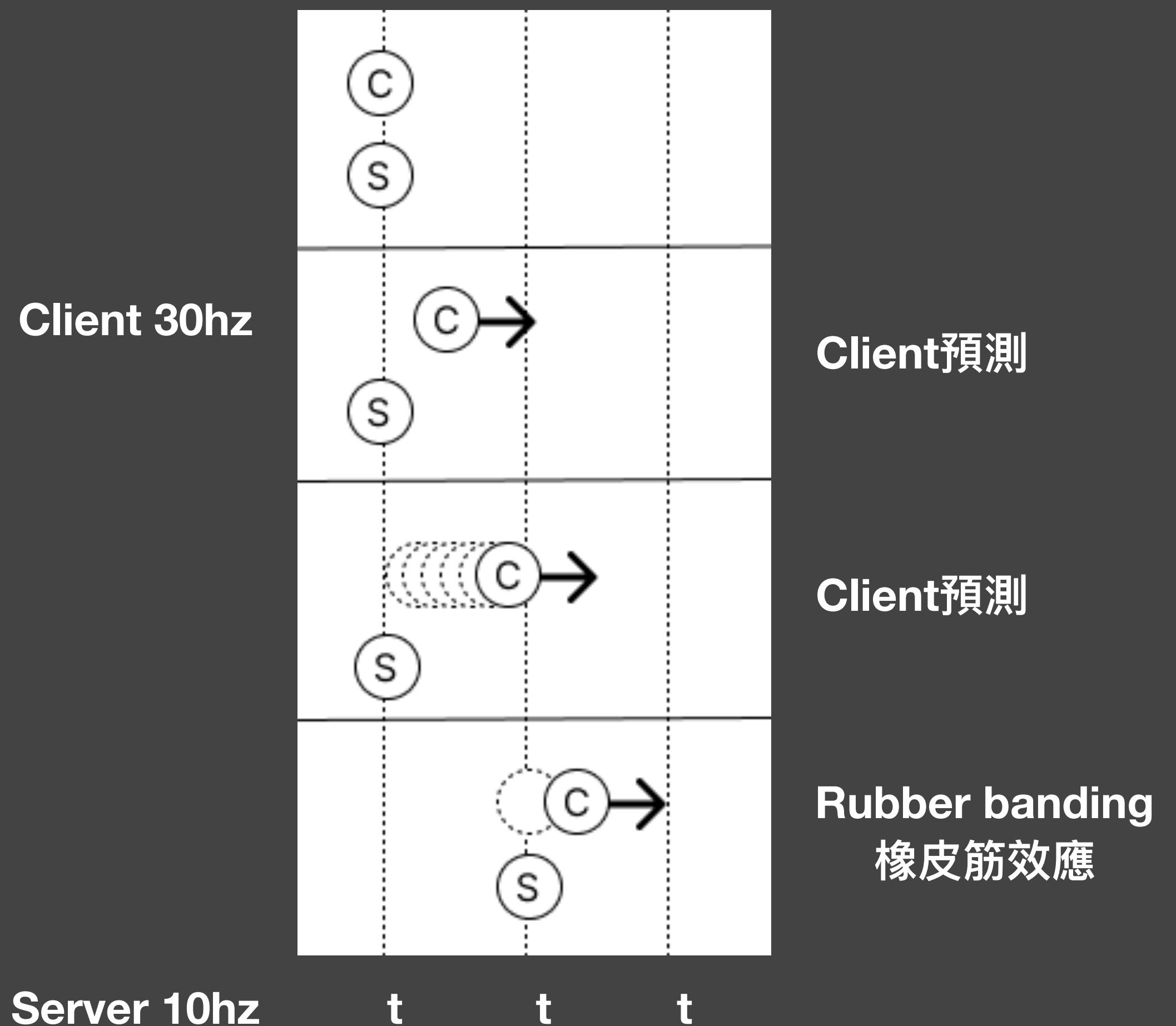
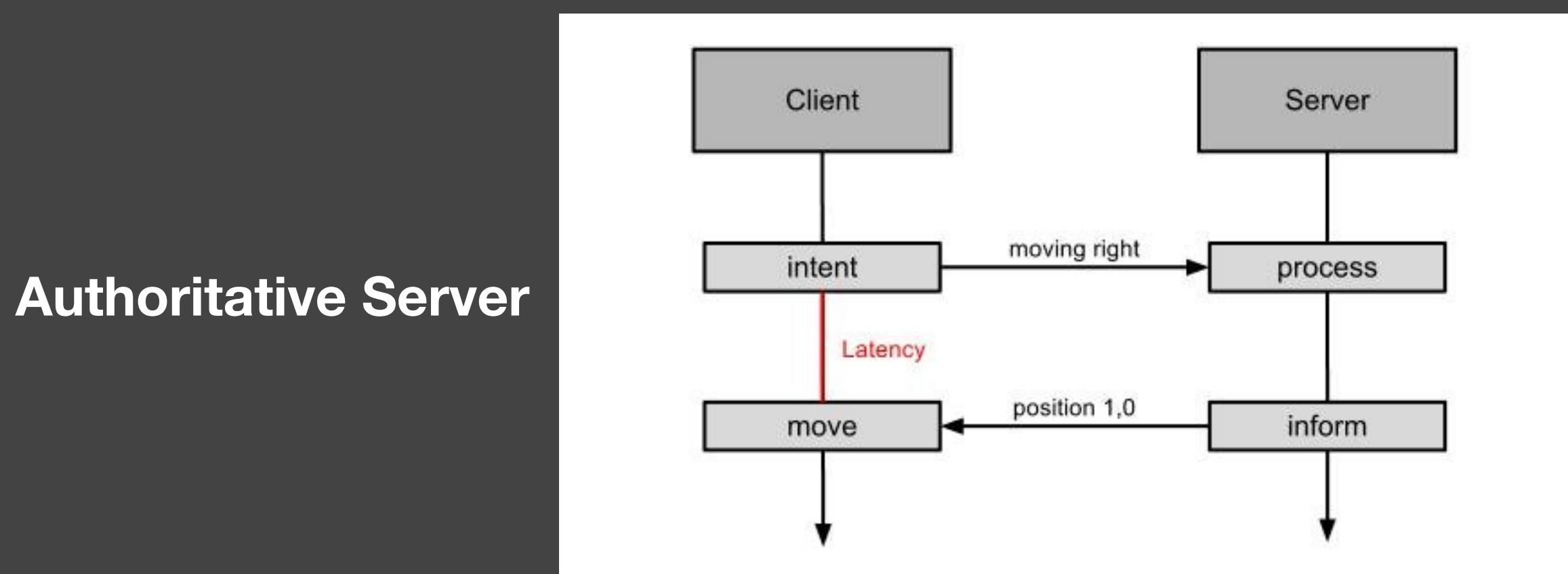
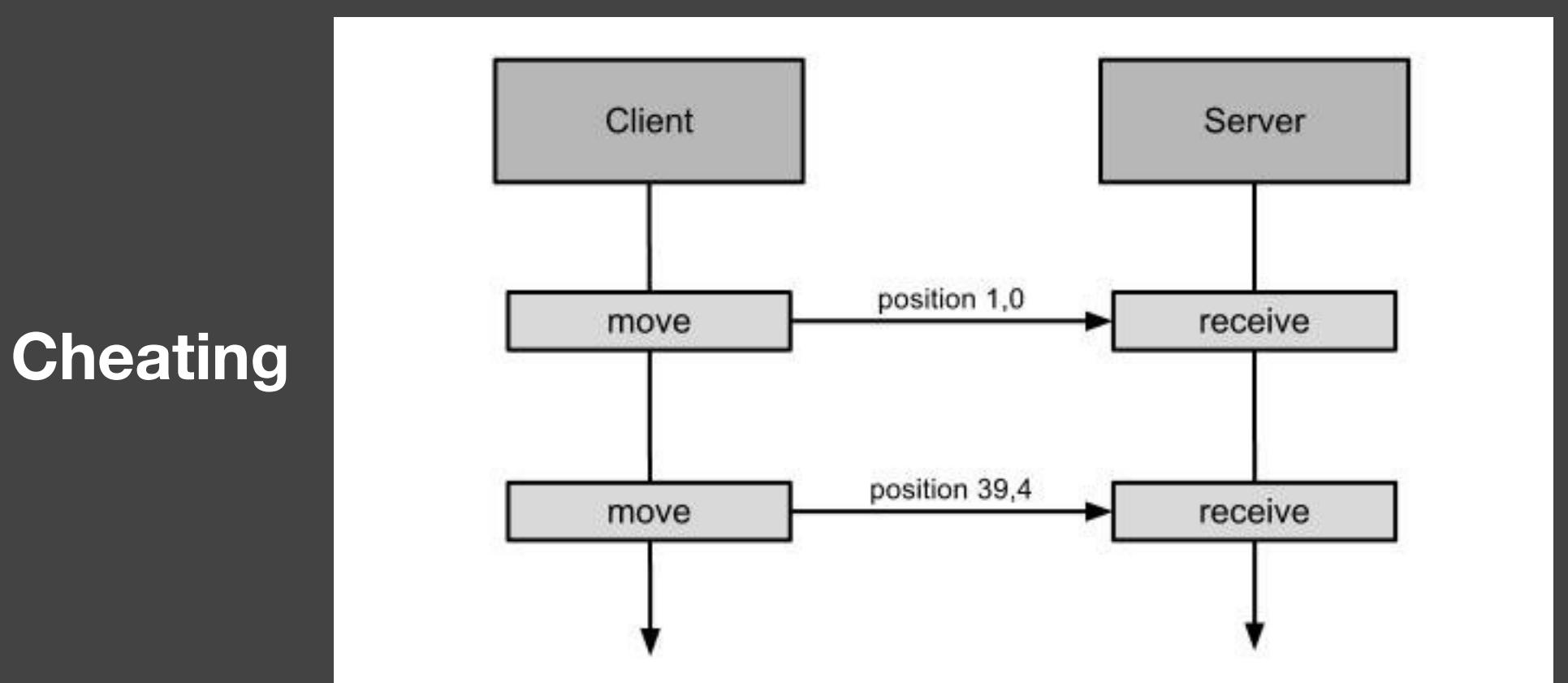


Super Bullets

客戶端預測

Client-side Prediction

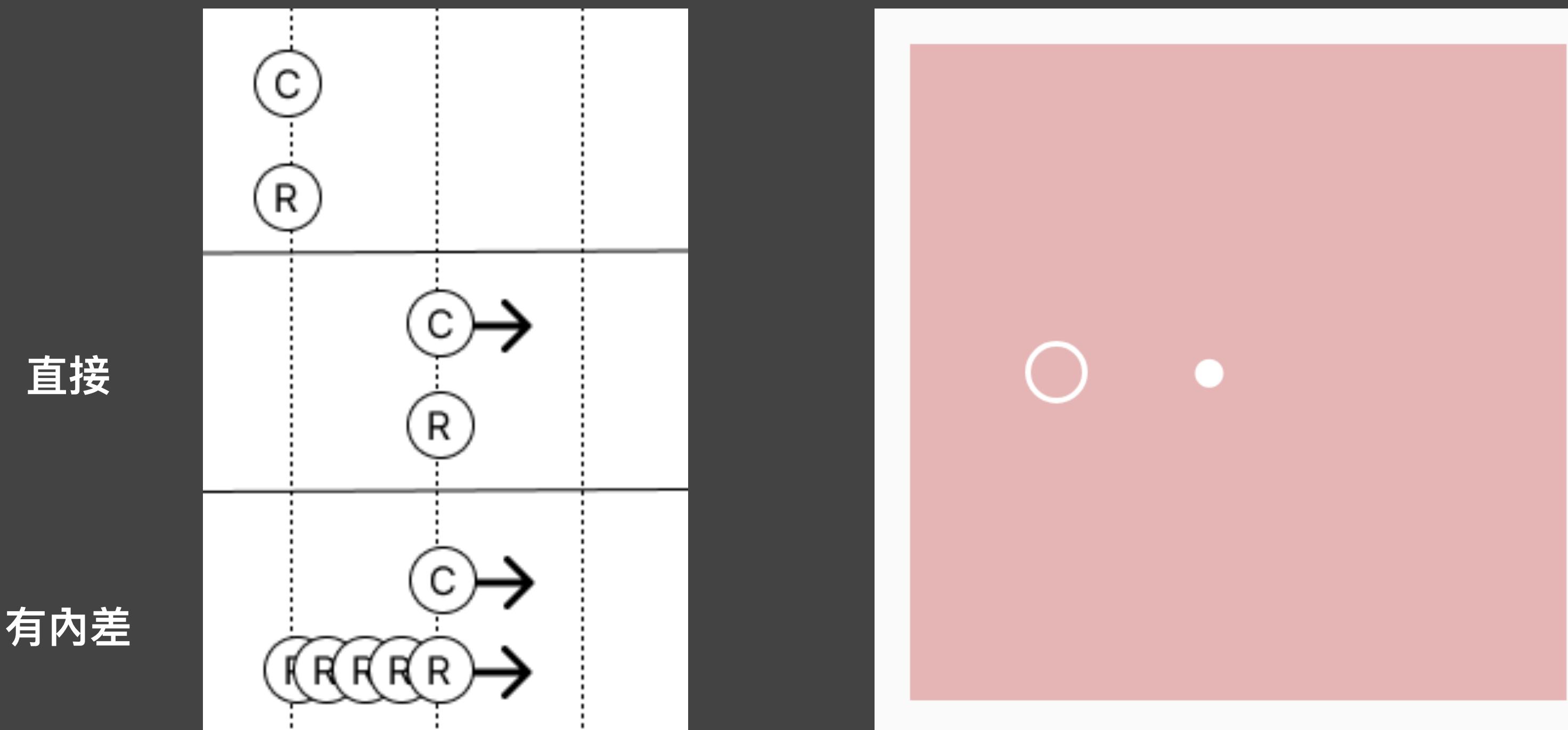
- 伺服器全權管理的遊戲（作弊）



內插

Snapshot Interpolation

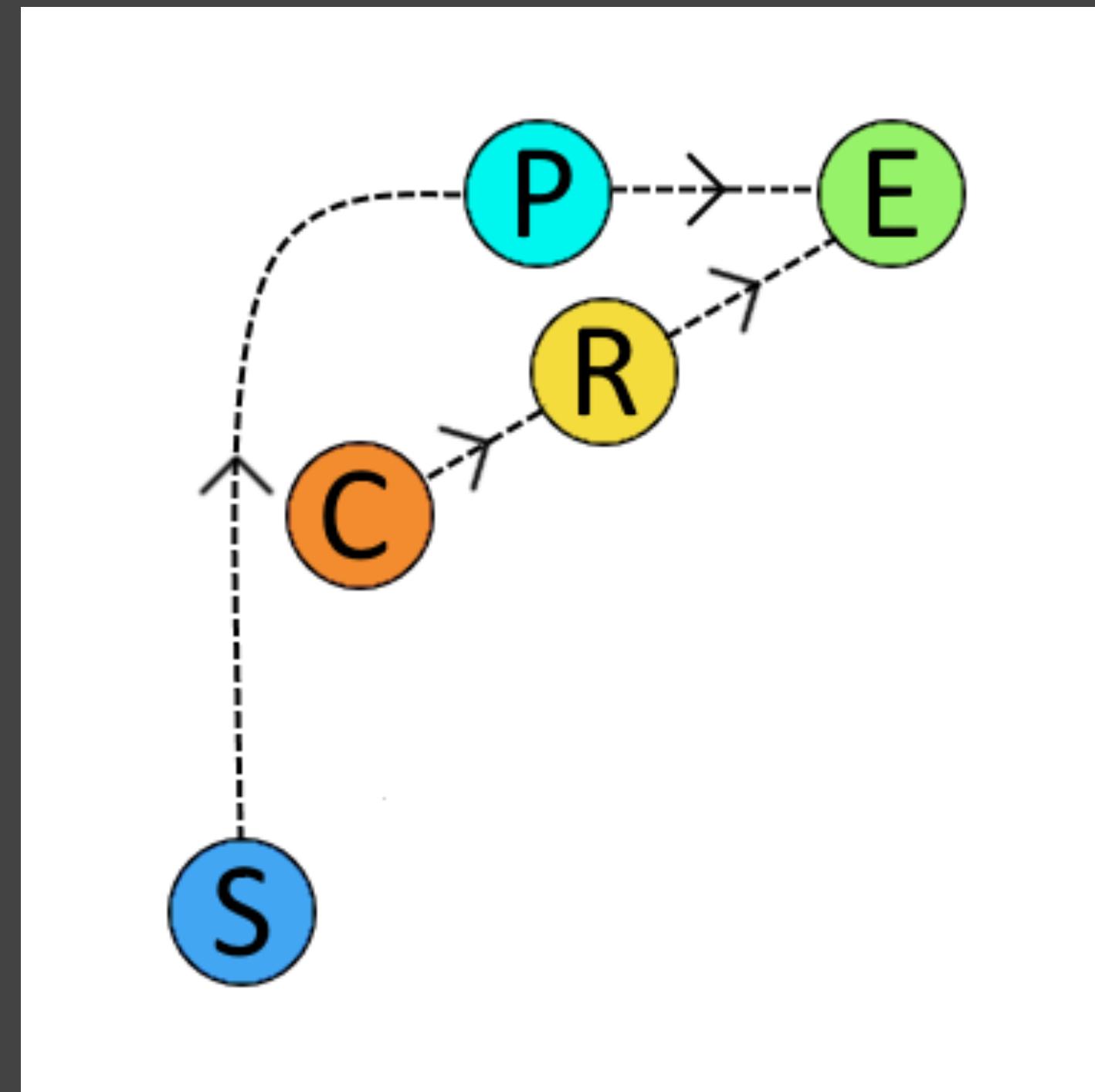
- 內插演算法會用在美化狀態的變化（視覺上）



外插

Extrapolation & Dead reckoning

- 外插演算法會用在延遲時，根據之前的狀態(速度、加速度)去預測新的位置
- Dead reckoning 則是用在有差異時，要如何同步回正確的位置



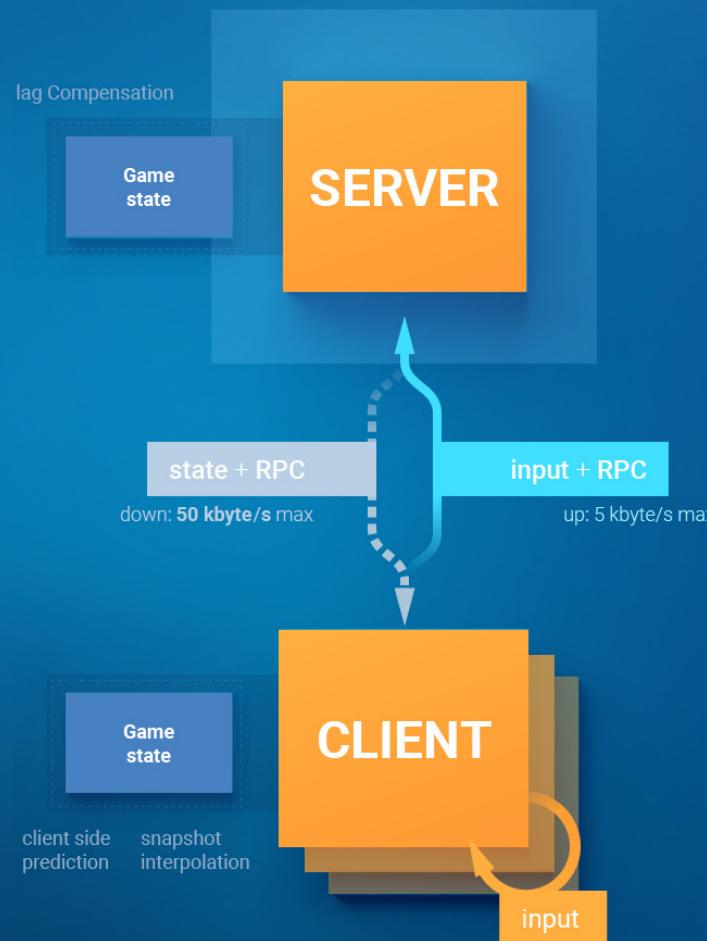
延遲補償

Lag Compensation

- Photon 說明
 - 延遲補償解決了快節奏網路遊戲中的一個基本問題：允許客戶端獲得所見即所得的體驗，但又不能相信客戶端。
 - 問題是網路上沒有一台機器在遊戲中處於完全相同的時間。一個客戶所看到的並基於他們的行動，對他們來說只是100%正確的。典型的例子是檢測對一個遠處物體的精確射擊；盡管客戶直接在十字準線上看到了目標，但實際上它已經移動了。
 - 延遲補償允許伺服器暫時從每個客戶的角度看世界，並決定他們是否真的處於一個位置來進行那個不可能的射擊。反過來說，這意味著目標可能會被擊中，盡管他們自己認為他們被安全地拉到了牆後面；然而，這不太可能被注意到。
- 例如：
 - 延遲高的玩家A，位置會像是閃現，所以延遲低的玩家B射擊A的時候，看到的是舊的位置，可是Server判斷的時候，可能已經收到玩家B的新位置(牆後面)，所以沒射中。
 - 延遲補償就會讓Server去回推當時A射擊的時候，他看到的狀況(B舊的位置)，來判斷是否擊中。

同步狀態資料的演算法 Replication Systems

Extreme Optimization: Replication Algorithms



High performance replication is essential for:

High Player Counts:

8 Team Death Match to 200 in small MMOs.

High Tick Rates:

30Hz for mobile to 120Hz in competitive FPSs.

Performance:

From low-end mobile devices to high-end PCs.

Bandwidth:

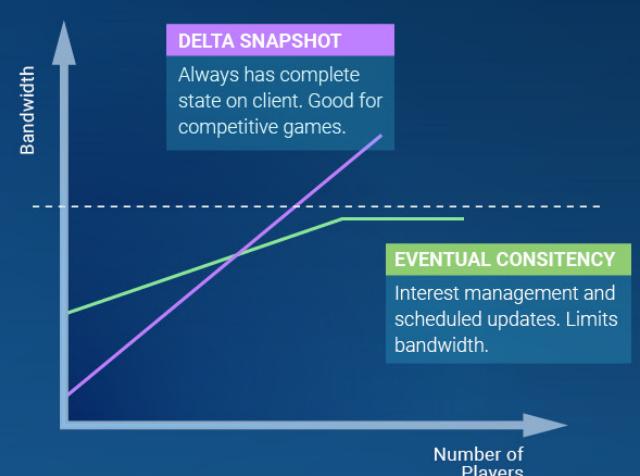
From weak mobile connections to Fiber.



Fusion implements two replication algorithms which can be switched between:

Delta Snapshots is mostly suitable for games or game modes with smaller player counts and object counts and higher tick rates but provides very fast and accurate state replication.

Eventual Consistency is suitable for games where a lot more objects or players are in the world, than a client can replicate.



極端優化： 複製算法

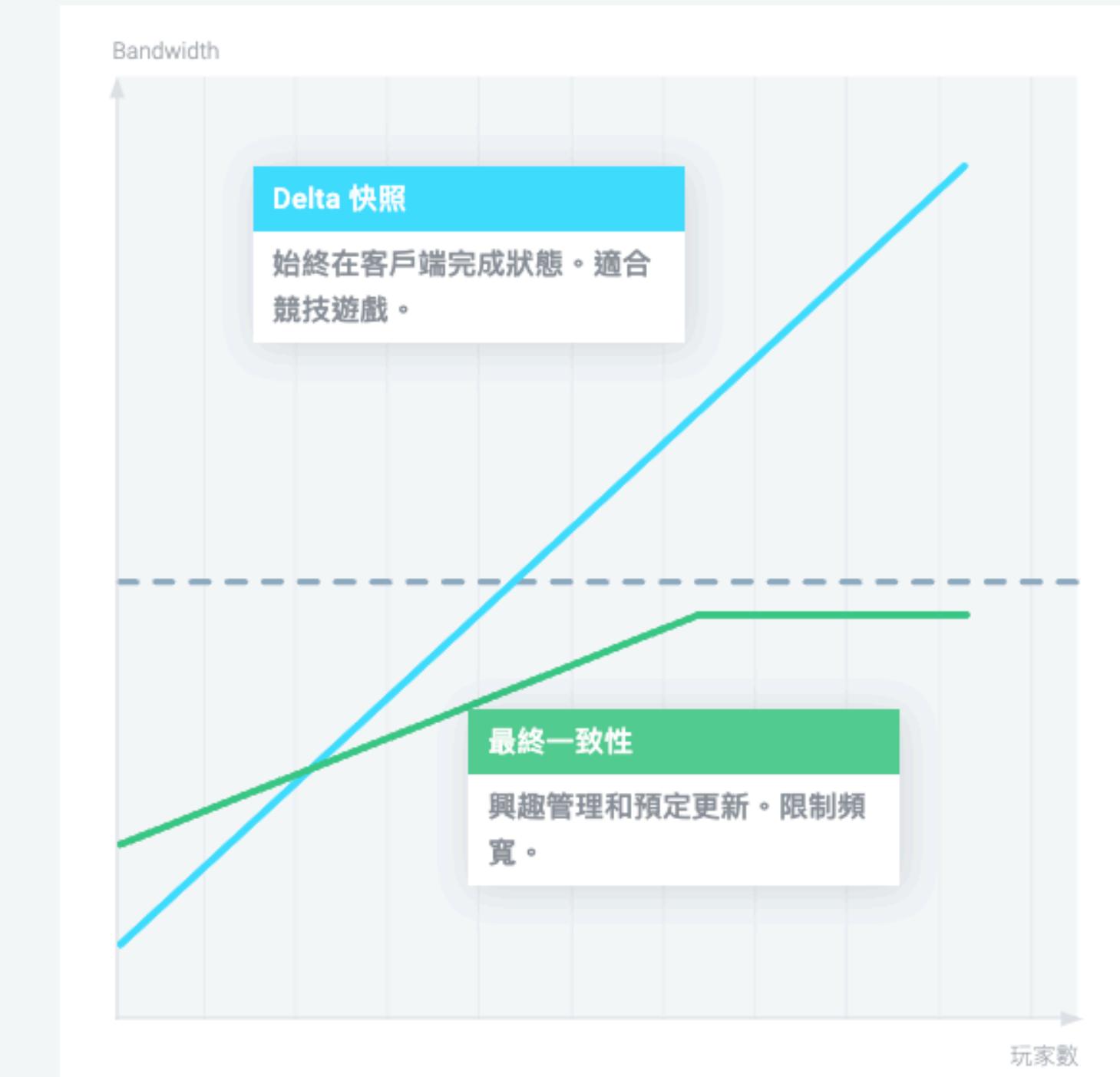
Fusion可以根據遊戲需求選擇融合複製算法。

Delta 快照

Delta快照非常適合 high-tick率準確的狀態復制，是物件數量較少的競技遊戲的理想選擇。

最終一致性

最終一致性非常適合具有大量玩家或物件數量，並頻寬很重要的遊戲。它還增加了控制的興趣管理(interest management)。



網路拓樸

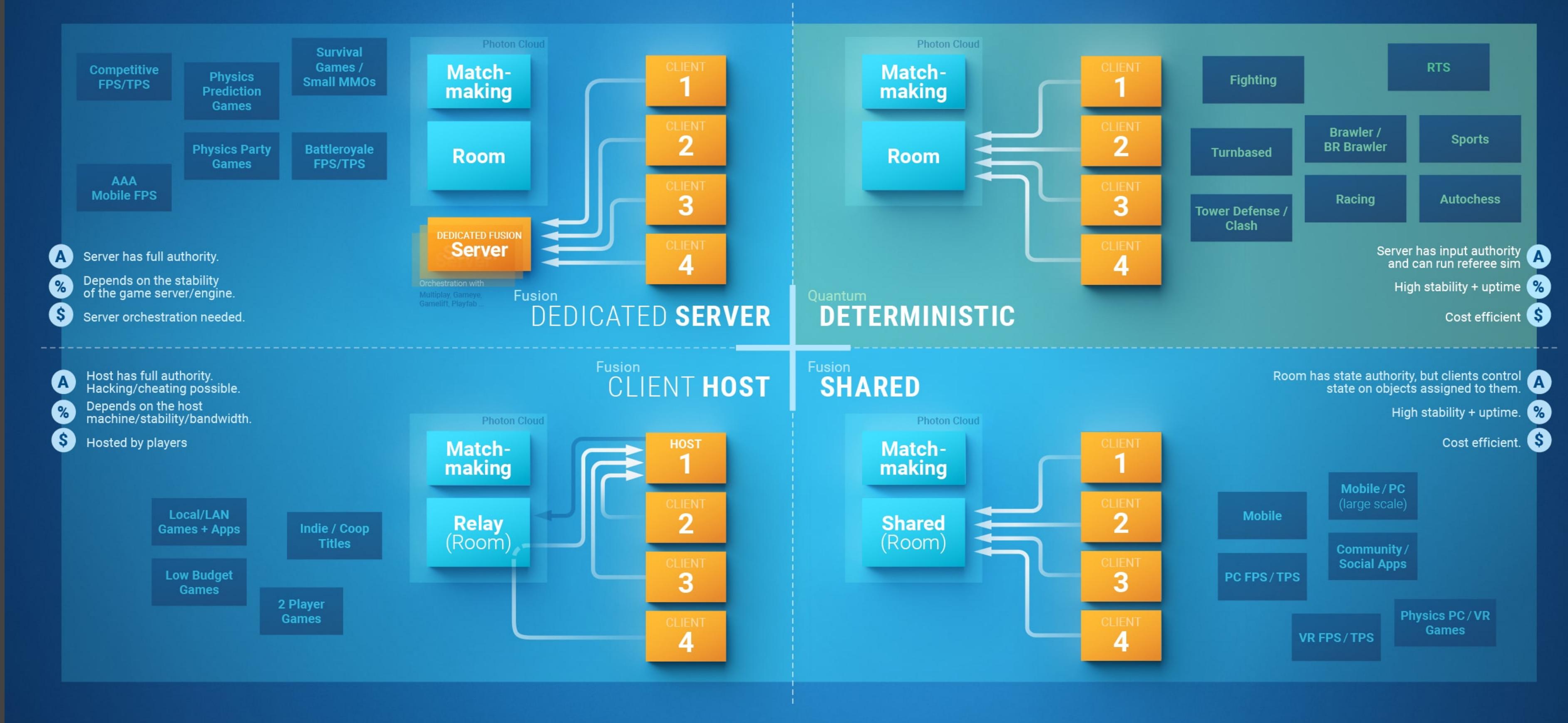
Network Topologies

Server	Host	Shared	Deterministic
具有公共 IP 的專用伺服器 (Headless Unity instance)	一名玩家當 Host，其他玩家會連到該 Host	房間有狀態權限以及可選插件(自定義代碼)	Quantum: 每個客戶端執行決定論性模擬
A 權限 ++ 伺服器擁有完全權限	-- Host擁有完全權限 可黑客/作弊	/ 房間具有狀態權限，但客戶端可控制分配到的物件狀態	++ 伺服器有輸入權限，可參考sim
S 狀態 ++ 可執行伺服器遷移	++ Turnkey Host遷移	++ 房間具有狀態權限，但客戶端可控制分配到的物件狀態	++ 房間具有狀態權限，但客戶端可控制分配到的物件狀態
L 延遲 ++ 低延遲：從客戶端直接連到伺服器	/ 延遲好(Punch)或相對差(relay)質量取決於Host	++ 直接連到Region內的房間	++ 直接連到Region內的房間
% QoS ++ 遊戲引擎的穩定性	- 取決於玩家/Host	+ 高穩定性+正常運行時間	+ 高穩定性+正常運行時間
\$ 費用 -- 專用伺服器的編排	++ 玩家+Photon Cloud託管	++ 成本效益高	++ 成本效益高

網路拓樸

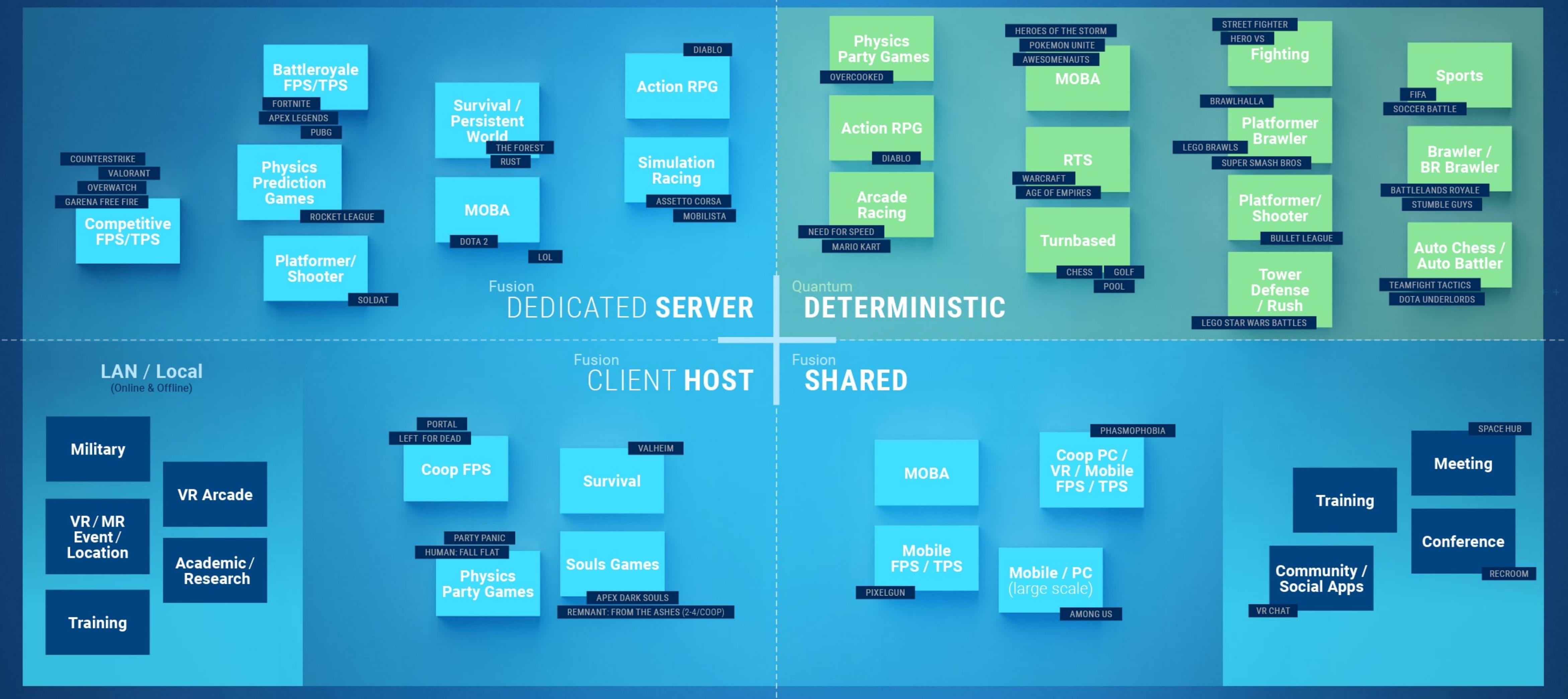
Network Topologies

Network Topologies



The Quadrant

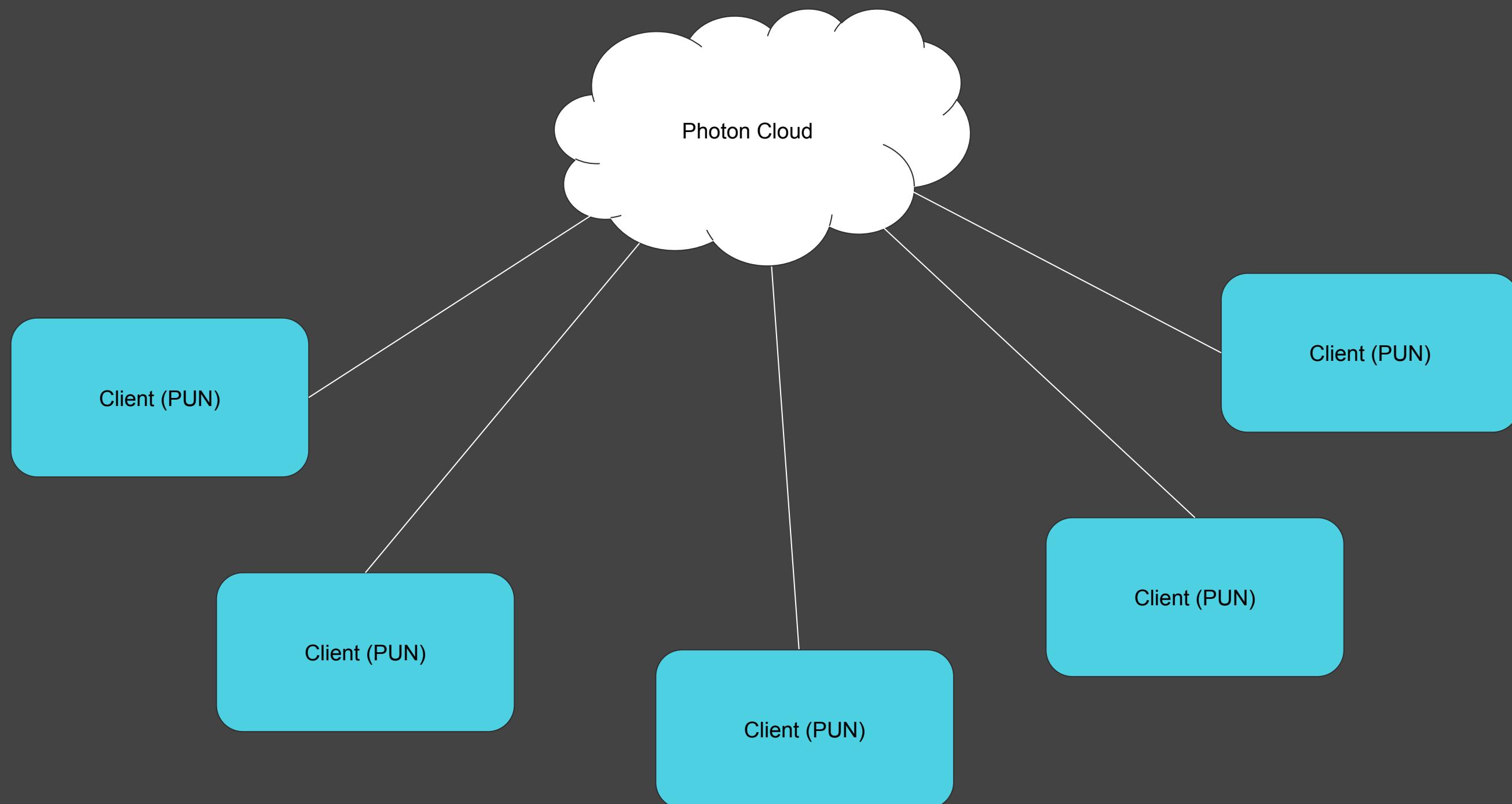
Pick the Right Product for Your Game/Genre



休息5分鐘

Photon Unity Network

一般連線, 非立即物理碰撞型, 像是任務 / 訊息 / 回合 / 解迷, 也就是 RPG, Puzzle, Turn-base 類型的遊戲.



Photon Unity Network

Home > Tools > Network > PUN 2 - FREE

The screenshot shows the Unity Asset Store page for the "PUN 2 - FREE" asset. The main visual is a large banner for "Photon Unity Networking" version 2, featuring a large blue "2" and the text "#1 Platform for UNITY - Multiplayer" and "FAST. RELIABLE. SCALABLE.". To the right of the banner, the product title "PUN 2 - FREE" is displayed, along with a "FREE" badge and a "20 CCU" badge. Below the banner, there are five smaller screenshots showing various features like "Global Low Latency" and "Unparalleled Cross Platform Support". On the right side of the page, detailed information about the asset is provided, including its rating (★★★★★ 253), views (6328 in the past week), and download links. A "Related keywords" section at the bottom lists terms like Multiplayer, PUN, Photon, Photon Realtime, UN, UFPS, Realtime, pun 2, PUN+, network engine, LAN, and Coop.

PUN 2 - FREE

FREE

20 CCU

NEW VERSION

#1 Platform for UNITY - Multiplayer

FAST. RELIABLE. SCALABLE.

1/5

Overview Package Content Releases Reviews Publisher Info

Related keywords

Multiplayer PUN Networking Photon
Photon Realtime UN UFPS Realtime
pun 2 PUN+ network engine LAN Coop

<https://assetstore.unity.com/packages/tools/network/pun-2-free-119922>

Create PUN App

The screenshot shows the Photon Cloud Apps dashboard. At the top right, there is a prominent 'CREATE A NEW APP' button with a red border. Below it, the main area displays the 'Tanks' application. The application card includes the name 'PUN', a status bar showing '20 CCU', and a summary section with 'App ID: c9f02694-b...', 'Peak CCU: 0', and 'Traffic used: 0%'. At the bottom of the card are three buttons: 'ANALYZE', 'MANAGE', and '-/+ CCU'.

Create a New Application

The application defaults to the **Free Plan**. You can change the plan at any time.

Photon Type *
Photon PUN

Name *
Tank

Description
Tank Game

Url
http://enter.your-url.here/

CREATE or [go back to the application list](#).

<https://www.photonengine.com/zh-TW/PUN>

<https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro>

Create PUN App

Your Photon Cloud Apps [+ CREATE A NEW APP](#)

Show [All Apps](#) in Status [Active](#) Sort by [Peak CCU](#)

Order [Descending](#) Display [As List](#)

[PUN](#) **20 CCU**

Tanks

App ID: [REDACTED]

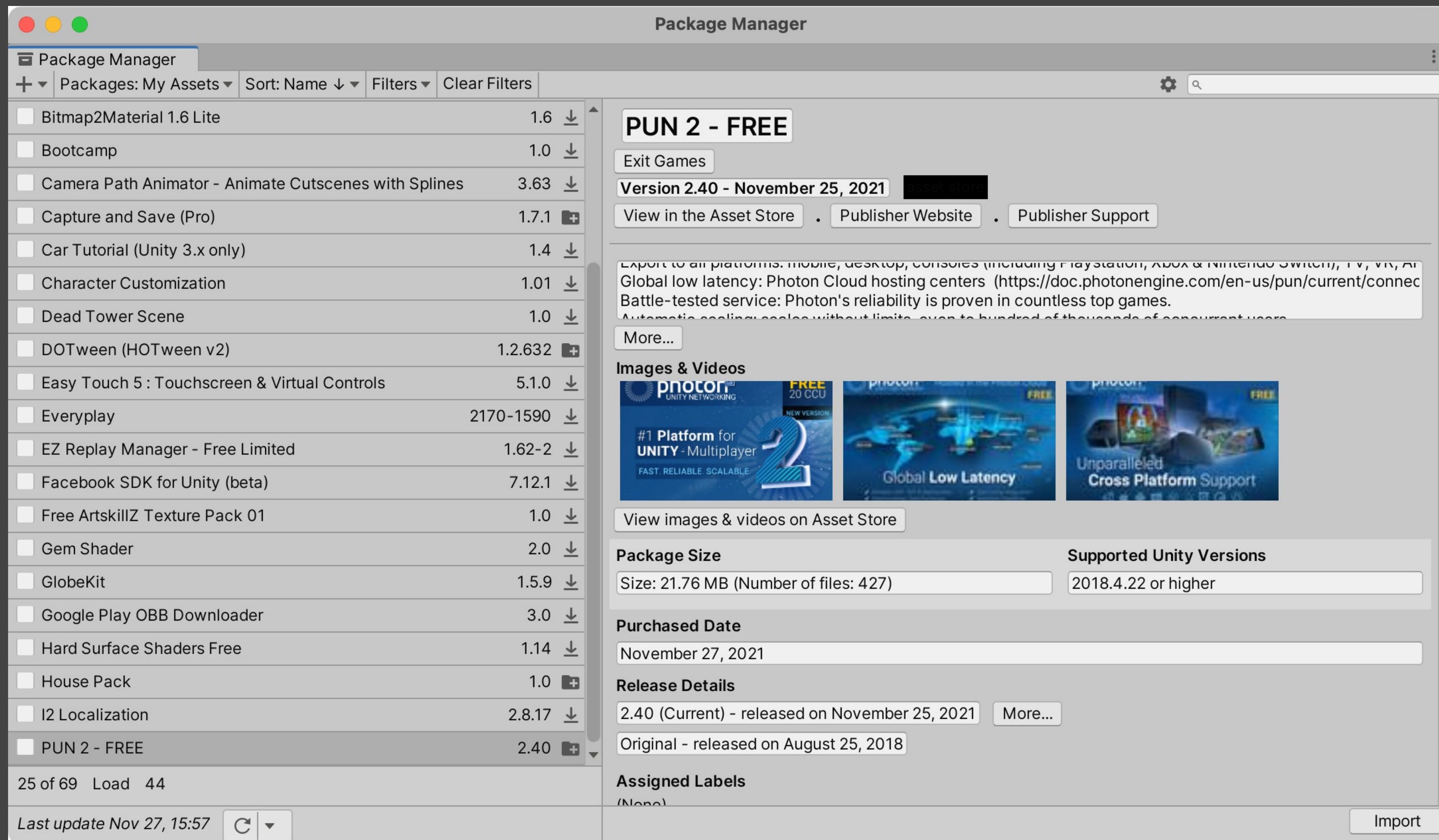
Peak CCU
0

Traffic used
0%

[ANALYZE](#) [MANAGE](#) [-/+ CCU](#)

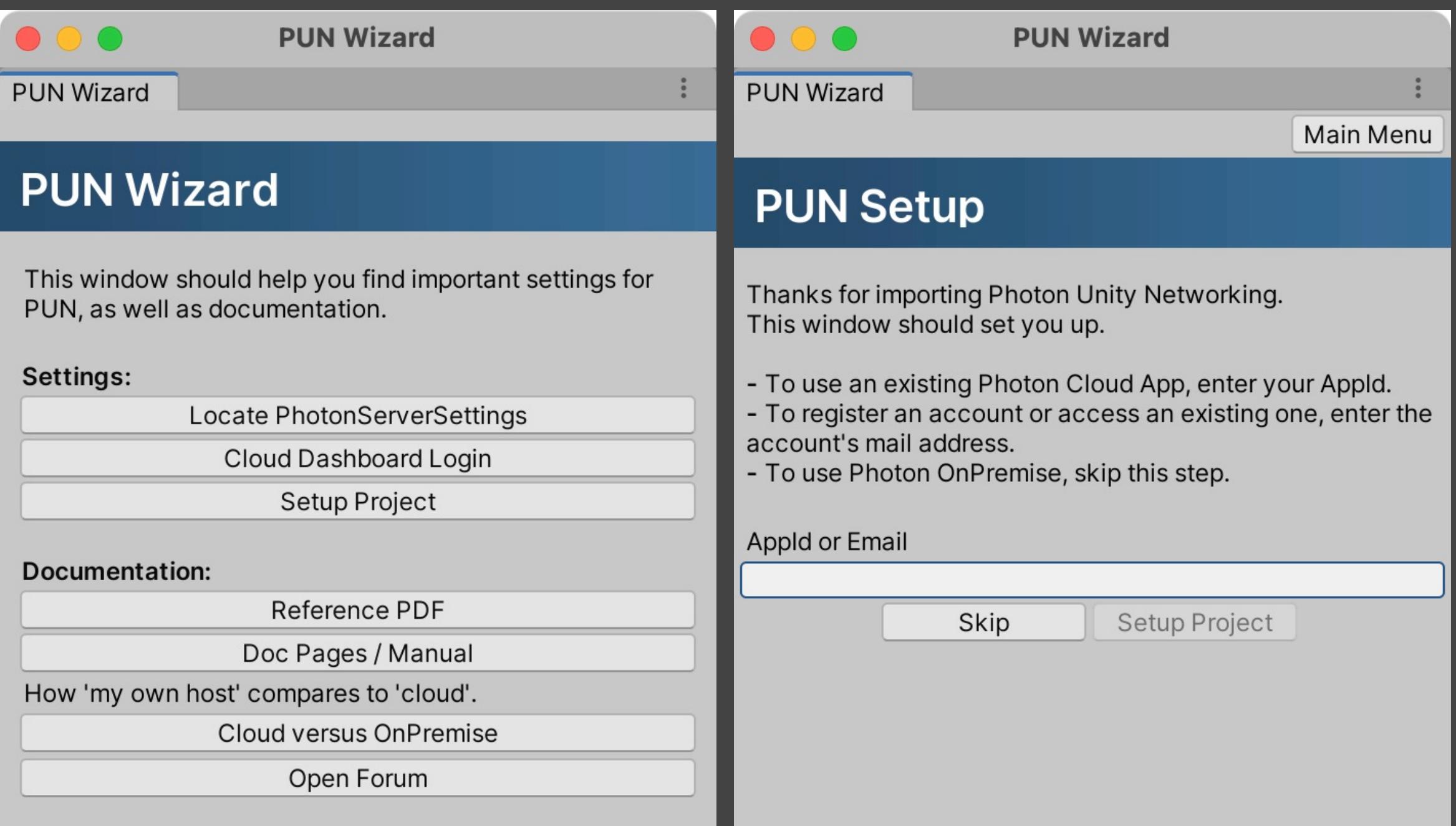
A screenshot of the Photon Cloud Apps web interface. At the top, it says "Your Photon Cloud Apps" and has a "CREATE A NEW APP" button. Below that are filters for "Show All Apps", "in Status Active", "Sort by Peak CCU", "Order Descending", and "Display As List". A prominent card for the "PUN" app is shown, featuring its name, a "20 CCU" badge, and a "Tanks" section. Within the tanks section, the "App ID" field is highlighted with a red border. Below it are two performance metrics: "Peak CCU" at 0 and "Traffic used" at 0%. At the bottom of the card are three buttons: "ANALYZE", "MANAGE", and "-/+ CCU".

Import PUN in Unity



PUN Setting

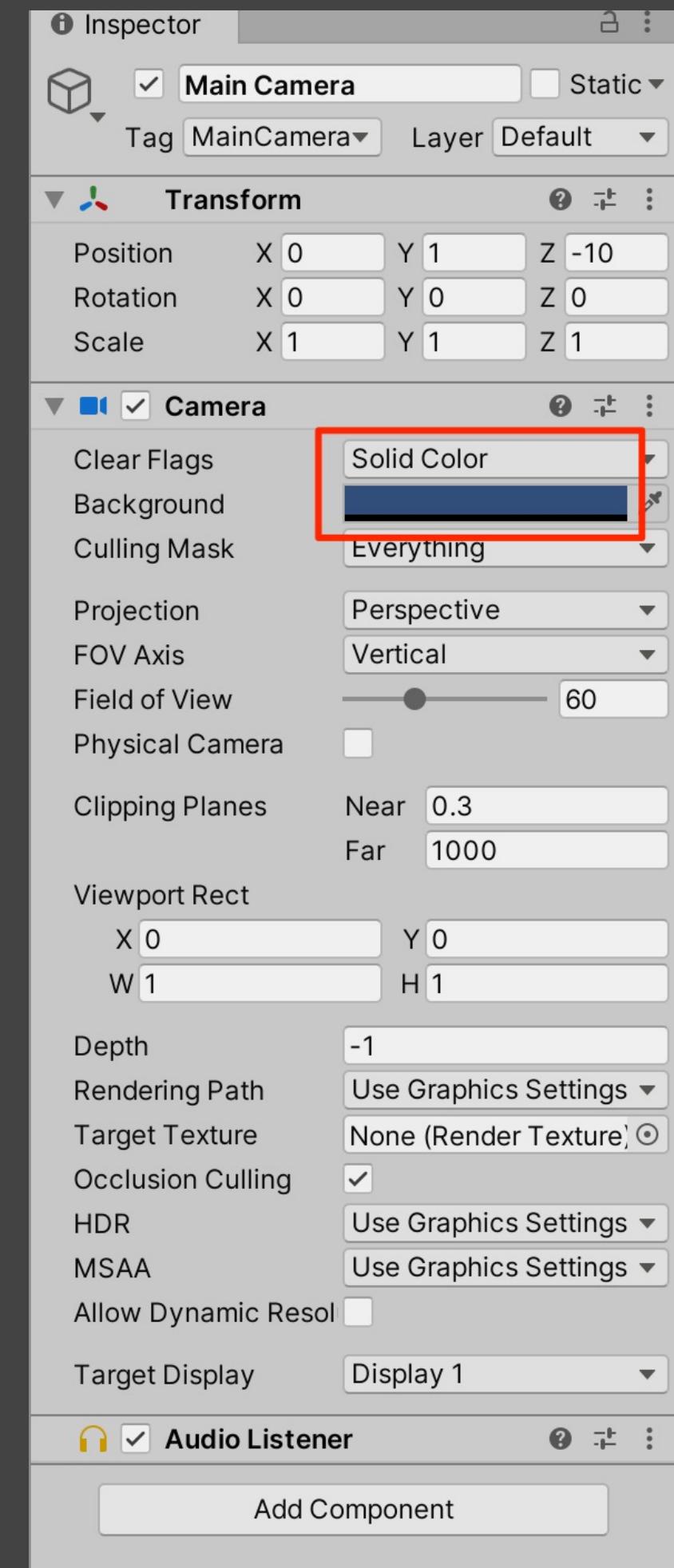
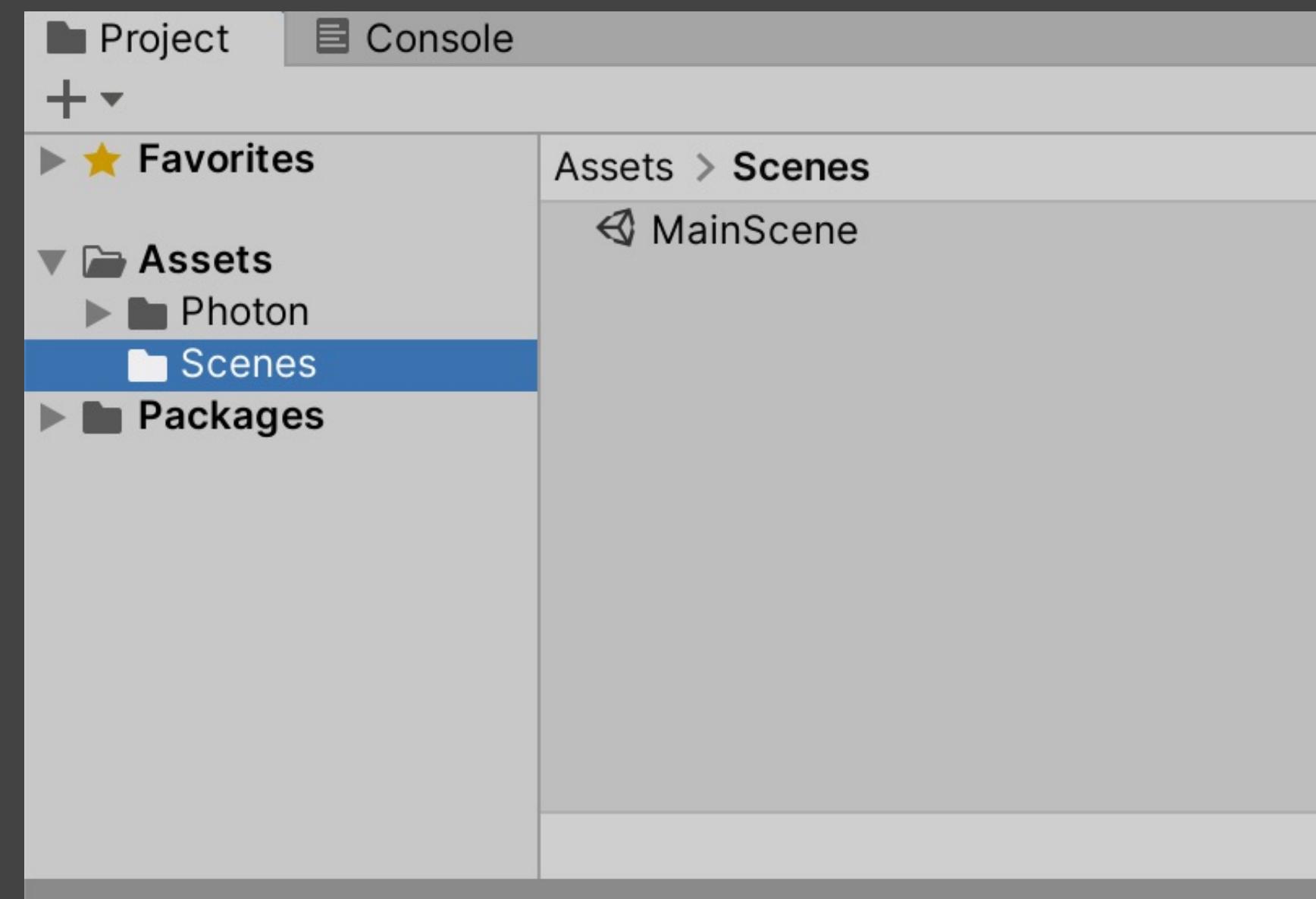
- 開啟 *Window > Photon Unity Networking > PUN Wizard* 視窗
- Select Setup Project
- Fill your AppId from your photon Dashboard



練習 5分鐘

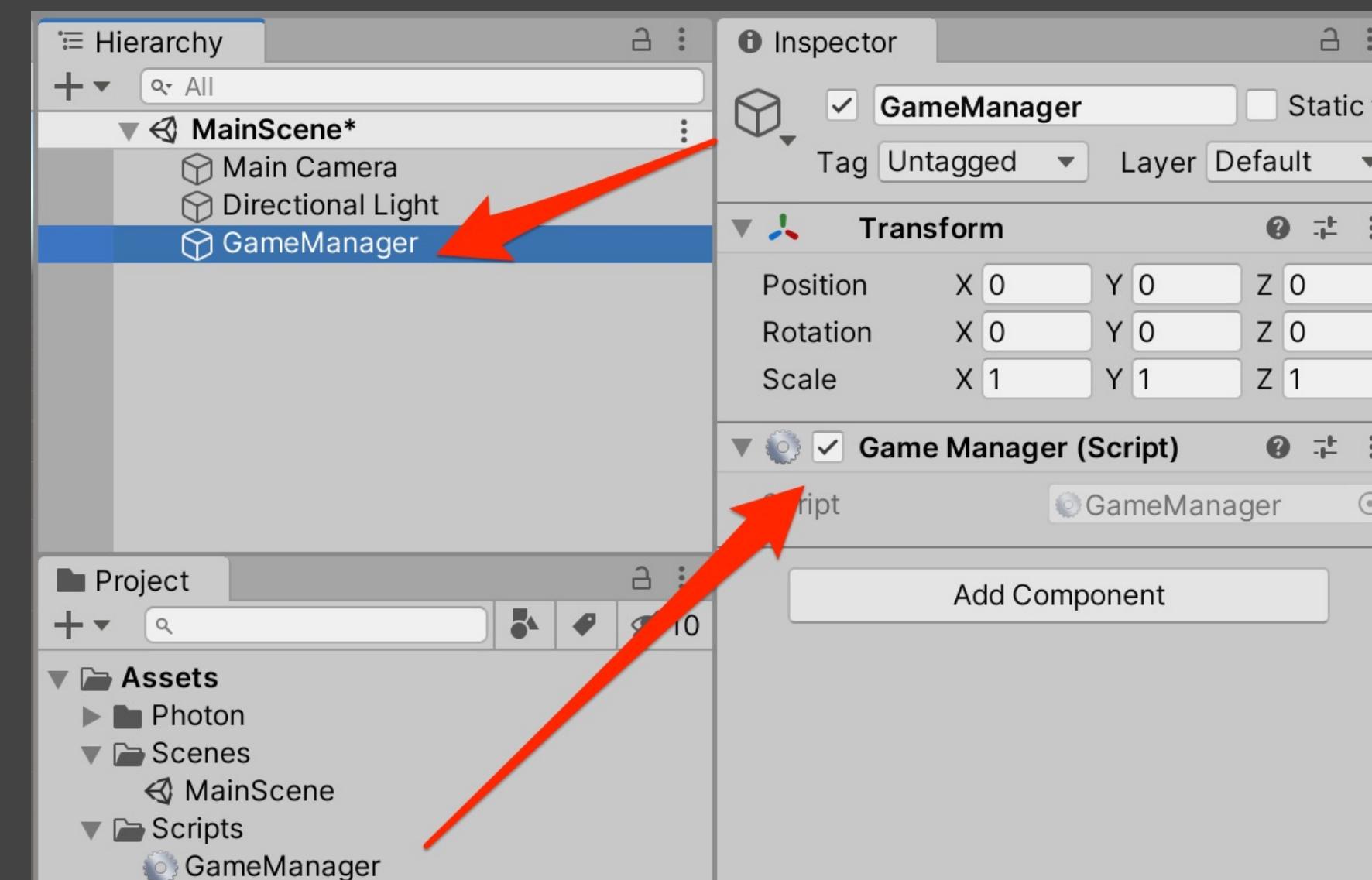
Setup Main Scene

- 儲存目前的 Scene，並取名 MainScene，並放在 Scenes 目錄下
- Camera Clear Flags 選 Solid Color



Game Manager

- 建立一個 Empty GameObject，命名為 **GameManager**
- 並建立一個 **GameManager** C# Script，放在 **Scripts** 目錄下
- 將 GameManager script 附加到 GameManager GameObject 的 behavior 上
- GameManager 會用來「管理整個場景」以及「負責與Photon Cloud Server間的連線」
- 為了讓它在整個遊戲過程中都可以被其他Scripts存取，並且只會有一個，因此我們將會把它設定成 **Singleton Object** 的型式，並且讓它不會再轉換場景時被刪除



Game Manager

```
using UnityEngine;

namespace Tanks
{
    public class GameManager : MonoBehaviour
    {
        public static GameManager instance;
        string gameVersion = "1";

        void Awake()
        {
            if (instance != null)
            {
                Debug.LogErrorFormat(gameObject,
                    "Multiple instances of {0} is not allow", GetType().Name);
                DestroyImmediate(gameObject);
                return;
            }

            DontDestroyOnLoad(gameObject);
            instance = this;
        }
    }
}
```

練習 5分鐘

MonoBehaviourPunCallbacks

- 這個類別會提供 photonView 物件，以及
- 各種 Callback 及 Event 事件，讓 PUN 可以呼叫
 - OnConnected / OnDisconnected
 - OnConnectedToMaster
 - OnJoinedLobby / OnLeftLobby
 - OnCreatedRoom / OnCreateRoomFailed
 - OnJoinedRoom / OnJoinRoomFailed / OnLeftRoom
 - OnPlayerEnteredRoom / OnPlayerLeftRoom

https://doc-api.photonengine.com/en/pun/v2/class_photon_1_1_pun_1_1_mono_behaviour_pun_callbacks.html

Game Manager

```
using Photon.Pun;
using UnityEngine;

public class GameManager : MonoBehaviourPunCallbacks

{
    public static GameManager instance;
    string gameVersion = "1";

    void Awake()
    {
        if (instance != null)
        {
            Debug.LogErrorFormat(gameObject,
                "Multiple instances of {0} is not allow", GetType().Name);
            DestroyImmediate(gameObject);
            return;
        }

        PhotonNetwork.AutomaticallySyncScene = true;
        DontDestroyOnLoad(gameObject);
        instance = this;
    }
}
```

PhotonNetwork

- PhotonNetwork.AutomaticallySyncScene
 - 設定是否讓 Room 裏面所有的 Client 自動跟著 MasterClient 更換相同的場景(Scene)
 - 如果設定為 true，MasterClient 可以呼叫 `PhotonNetwork.LoadLevel()` 函式時，讓其他 Client 同時切換場景

Master Client

- 在 P2P 遊戲中，通常其中一個客戶端(Client)會作為 Game Logic Server
- 通常最早進去或是建立這個房間的客戶端
- 在 Photon 裡，負責 Game Logic Server 的客戶端稱為 Master Client

練習 5分鐘

Connect to Photon Cloud

```
void Start()
{
    PhotonNetwork.GameVersion = gameVersion;
    PhotonNetwork.ConnectUsingSettings();
}

public override void OnConnected()
{
    Debug.Log("PUN Connected");
}

public override void OnConnectedToMaster()
{
    Debug.Log("PUN Connected to Master");
}

public override void OnDisconnected(DisconnectCause cause)
{
    Debug.LogWarningFormat("PUN Disconnected was called by PUN with reason {0}", cause);
}
```

PhotonNetwork

- ConnectUsingSettings
 - 會自動使用之前已設定好的 Photon Services Setting 的值 (AppID, Hosting, Protocol...) 作為資訊傳到 Photon Cloud
- PhotonNetwork.GameVersion
 - 主要是用來讓 Photon Server 知道我們雖然連上的是同款遊戲(AppID 是一樣的), 但可能是不同版本, 這樣我們以後就可以處理新舊程式版本的相容性問題, 對不同版本可以有不同的連線或是資料處理方式

<https://doc.photonengine.com/en-us/pun/current/reference/glossary>

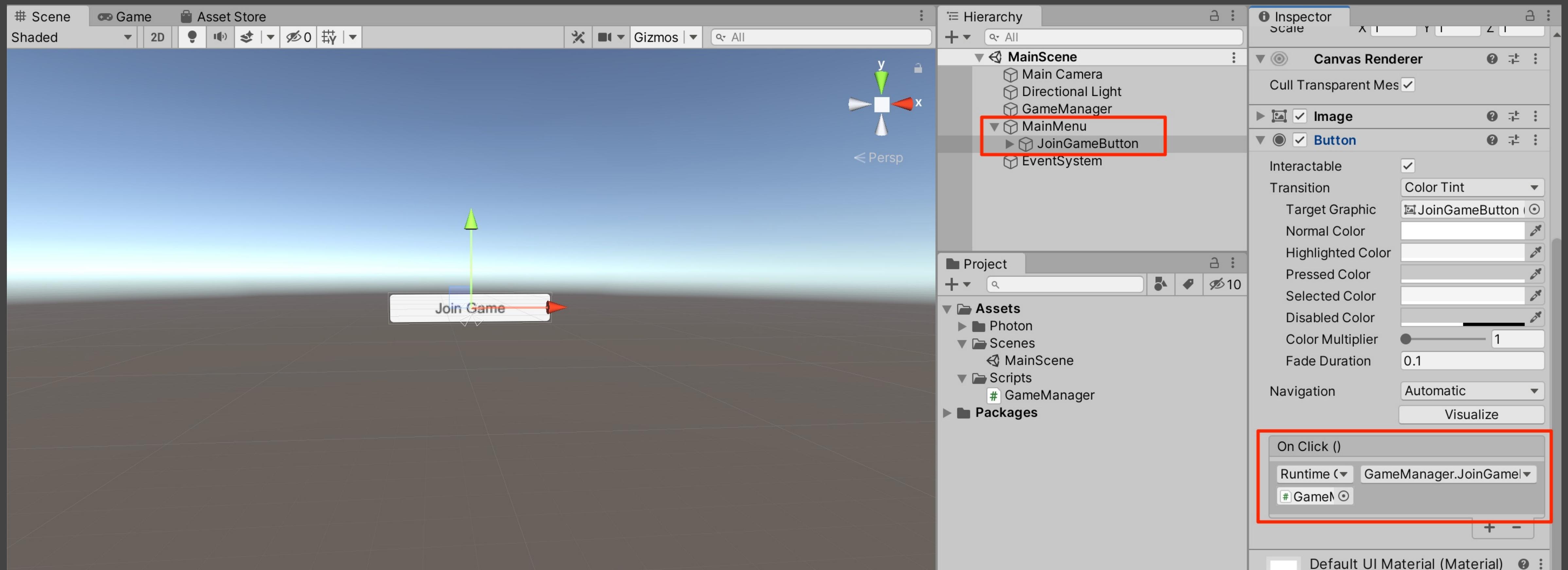
<https://doc.photonengine.com/zh-tw/pun/current/connection-and-authentication/regions>

練習 5分鐘

Create & Join Room

- 新增一個2D UI按鈕，命名為 JoinGameButton
- 按鈕上文字設為 Join Game
- 並且將新的 Canvas 命名為 MainMenu
- 接下來要實作建立新房間或加入現有房間的 Script
- 建立 Game Room 需要做一些設定
 - 最大人數
 - 是不是隱藏
 - 是不是開啟 / 關閉

Create & Join Room



Create & Join Room

- 建立 或是 加入 Room
 - 在 PUN 中，是使用 ***JoinOrCreateRoom*** 函式來加入 Room，
 - 如果房間還沒建立，PUN會幫我們建立，其他人只要用同樣的函式，並以相同房間名稱就可以加入同一個房間

```
public void JoinGameRoom()
{
    var options = new RoomOptions
    {
        MaxPlayers = 6
    };

    PhotonNetwork.JoinOrCreateRoom("Kingdom", options, null);
}
```

- 然後將 JoinGameButton 的 OnClick 設定為按下時，執行此 Script

Create & Join Room

- 最後複寫 OnJoinedRoom 來獲取加入成功的通知

```
public override void OnJoinedRoom()
{
    Debug.Log("Joined room!!");
}
```

- 現在我們只印出Debug訊息，之後我們會在這邊載入遊戲場景以及玩家的角色

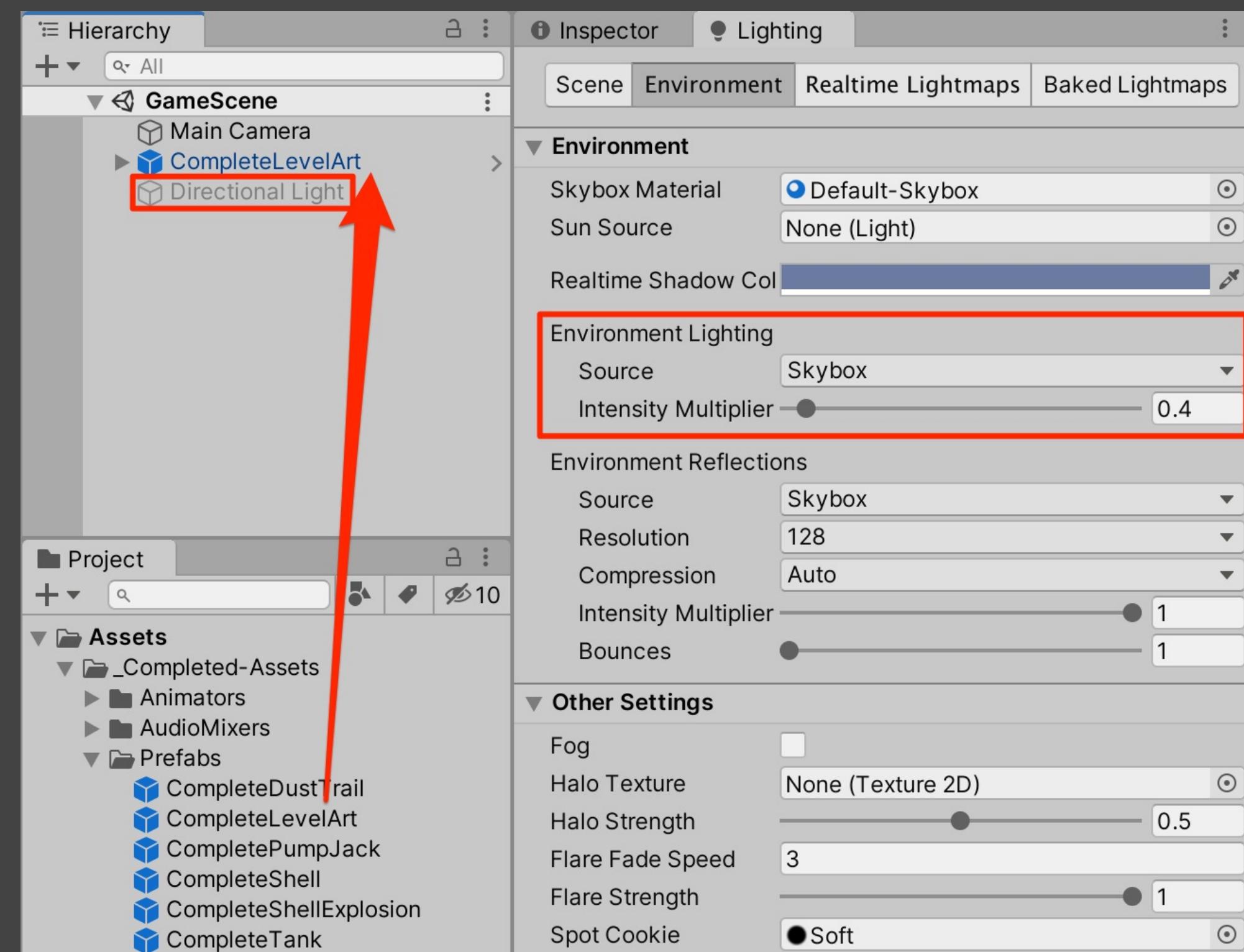
練習 5分鐘

Create Game Scene

- 汇入坦克遊戲資源
 - 下載 UnityTanksAssets.unitypackage & Import to your Game
 - <https://reurl.cc/GxQdNx>
- 注意：
 - 請使用上面的連結的資源，已針對新版的*Unity*處理過
 - 不要使用官方教學提供的

Create Game Scene

- 建立一個新的 Scene **GameScene**
- 並將 **_Completed-Assets\Prefabs\CompleteLevelArt.prefab** 拖到 GameScene 場景中
- 將 **Directional Light** 關掉
- 設定環境光
 - 選單 Window > Rendering > Lighting/Environment



練習 5分鐘

載入 Game Scene

- 自動同步場景機制

```
PhotonNetwork.AutomaticallySyncScene = true;
```

- 載入 Game Scene

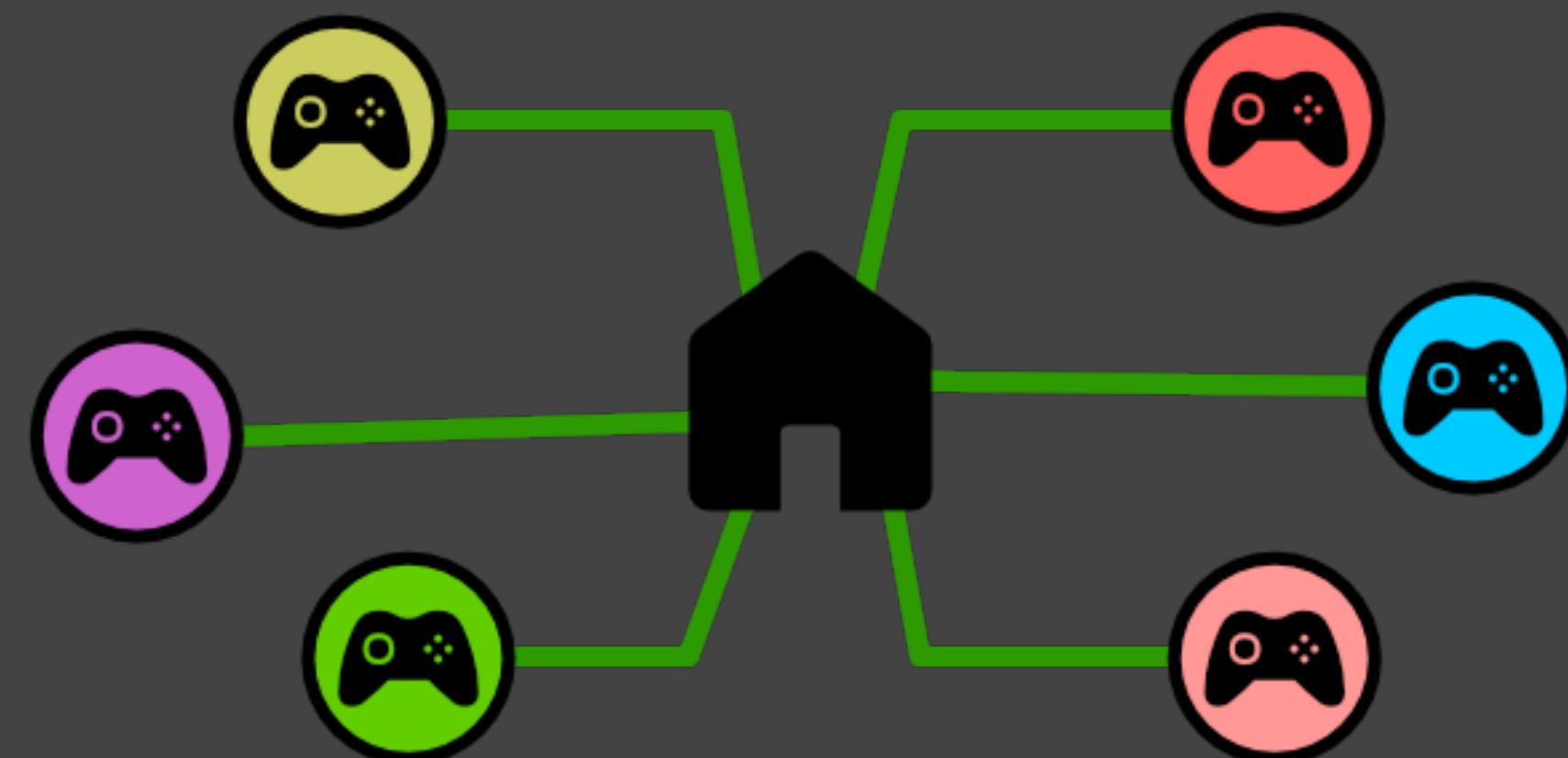
```
public override void OnJoinedRoom()
{
    if (PhotonNetwork.IsMasterClient)
    {
        Debug.Log("Created room!!");
        PhotonNetwork.LoadLevel("GameScene");
    }
    else
    {
        Debug.Log("Joined room!!");
    }
}
```

Master Client

- 在 P2P 遊戲中，通常其中一個客戶端(Client)會作為 Game Logic Server
- 通常最早進去或是建立這個房間的客戶端
- 在 Photon 裡，負責 Game Logic Server 的客戶端稱為 Master Client

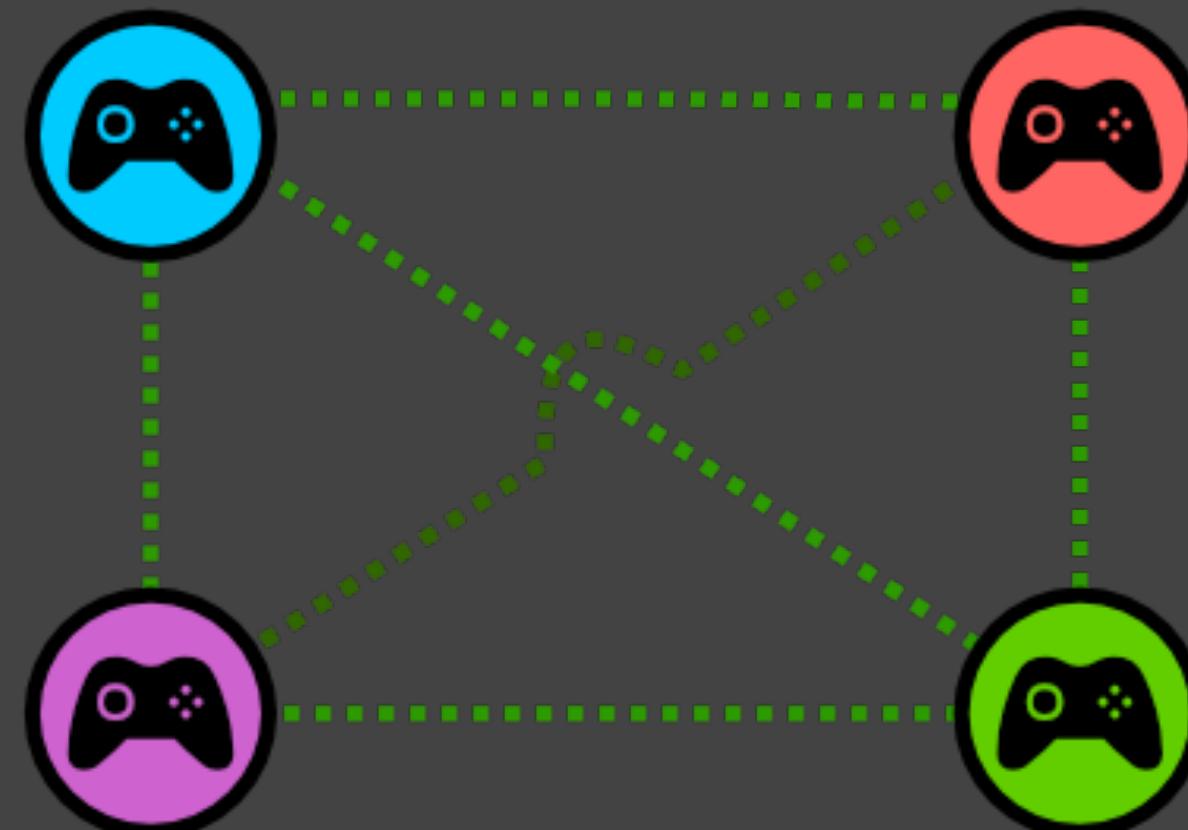
Client-Server 架構

- 主從式架構
- 所有的 Client 都連上同一台 Server 主機
- Server 負責傳送玩家之間的狀態，記錄所有玩家狀態
- Server 管理整個遊戲的資源及狀態，遊戲邏輯由 Server 控管



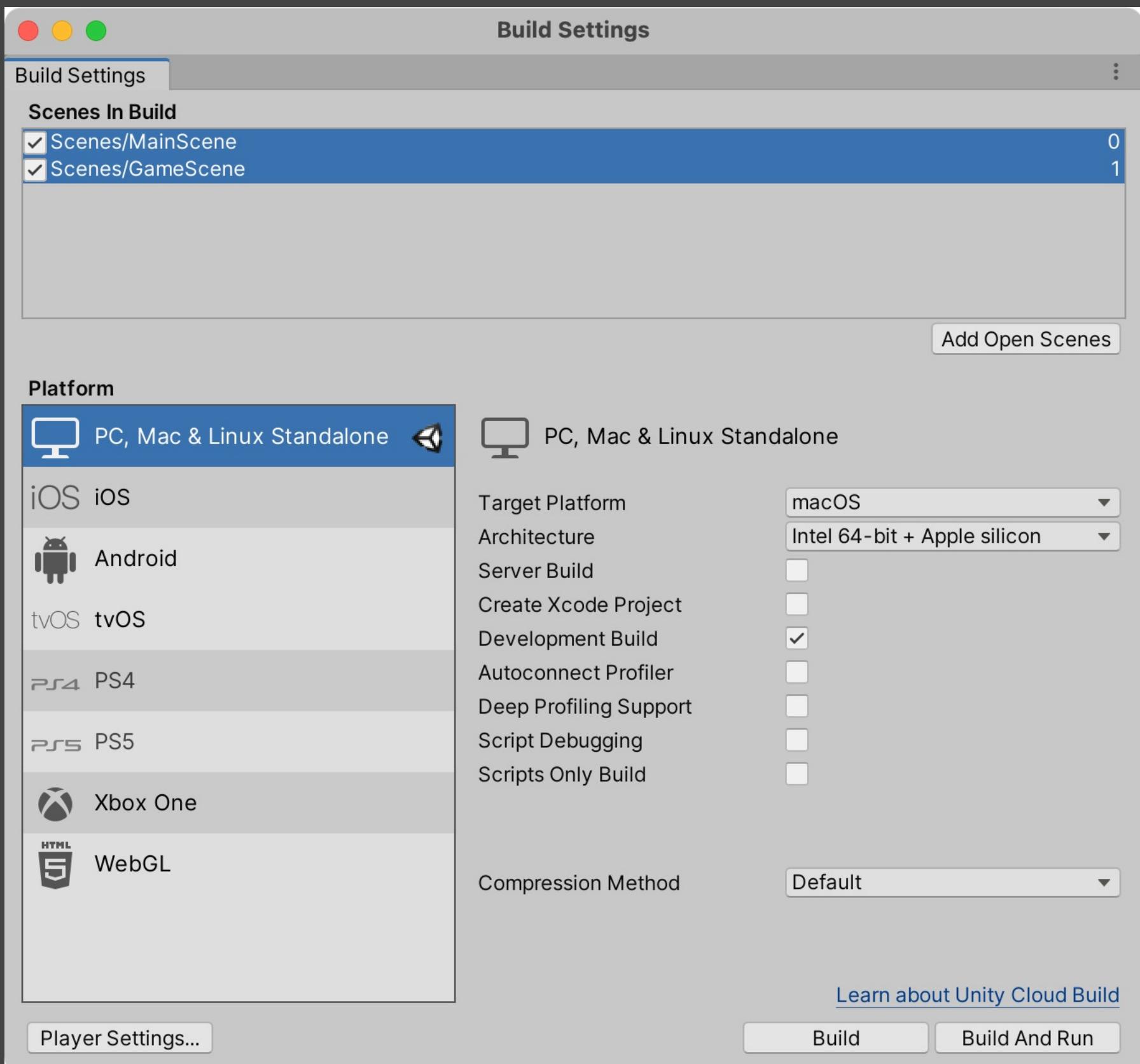
Peer-To-Peer (P2P)

- 點對點模式
- 不需要獨立主機做訊息的派送
- 每個Client (Peer, Player) 會負責傳送自己資料給所有 Client
- 比較常見是區域網路遊戲(LAN Game, Local Network Game)，大概 4~8 人的遊戲



載入 Game Scene

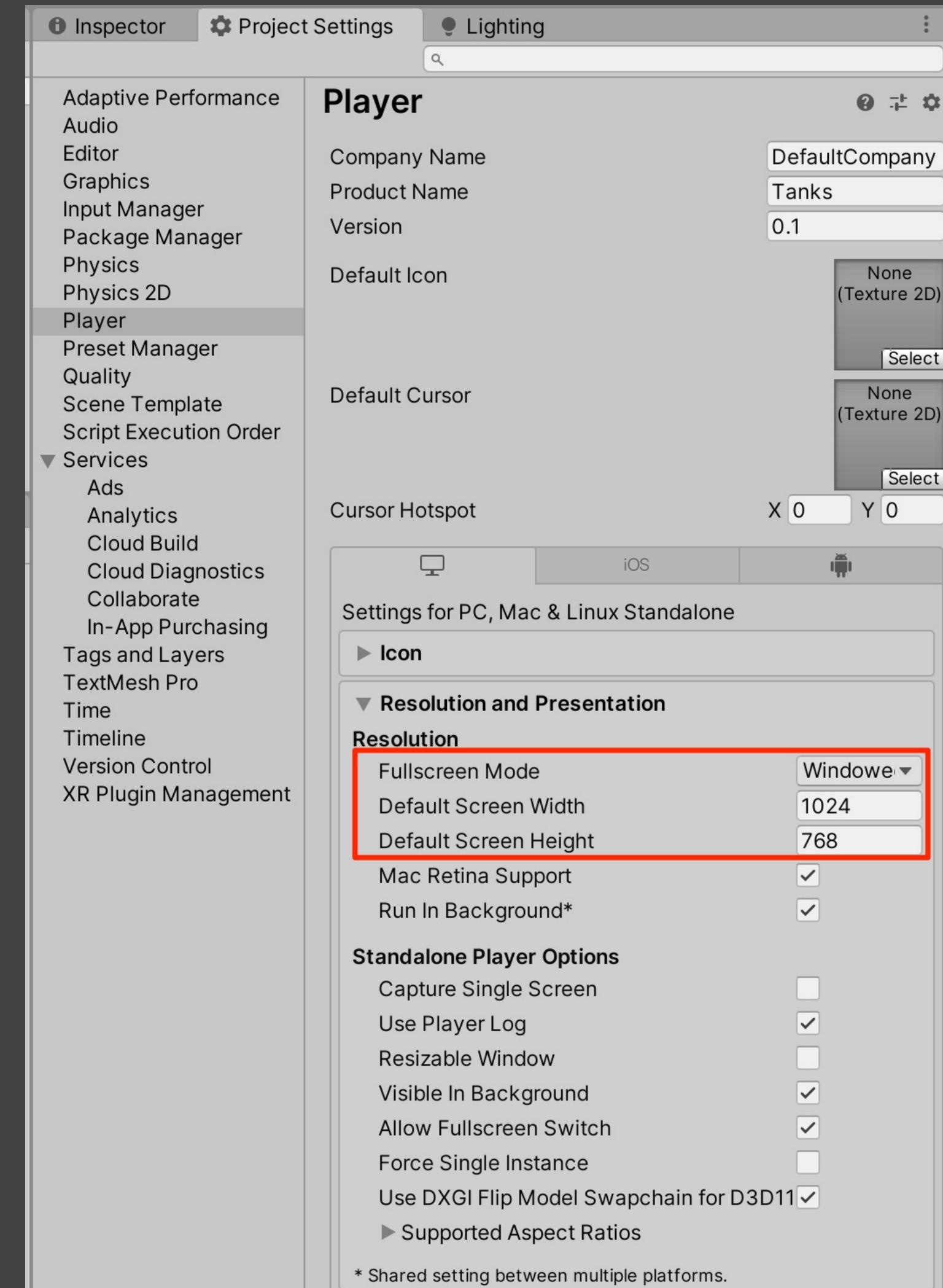
- 開啟 File > Build Settings 視窗，將兩個Scene加到Build中
- 切換回 MainGame，再 Play



練習 5分鐘

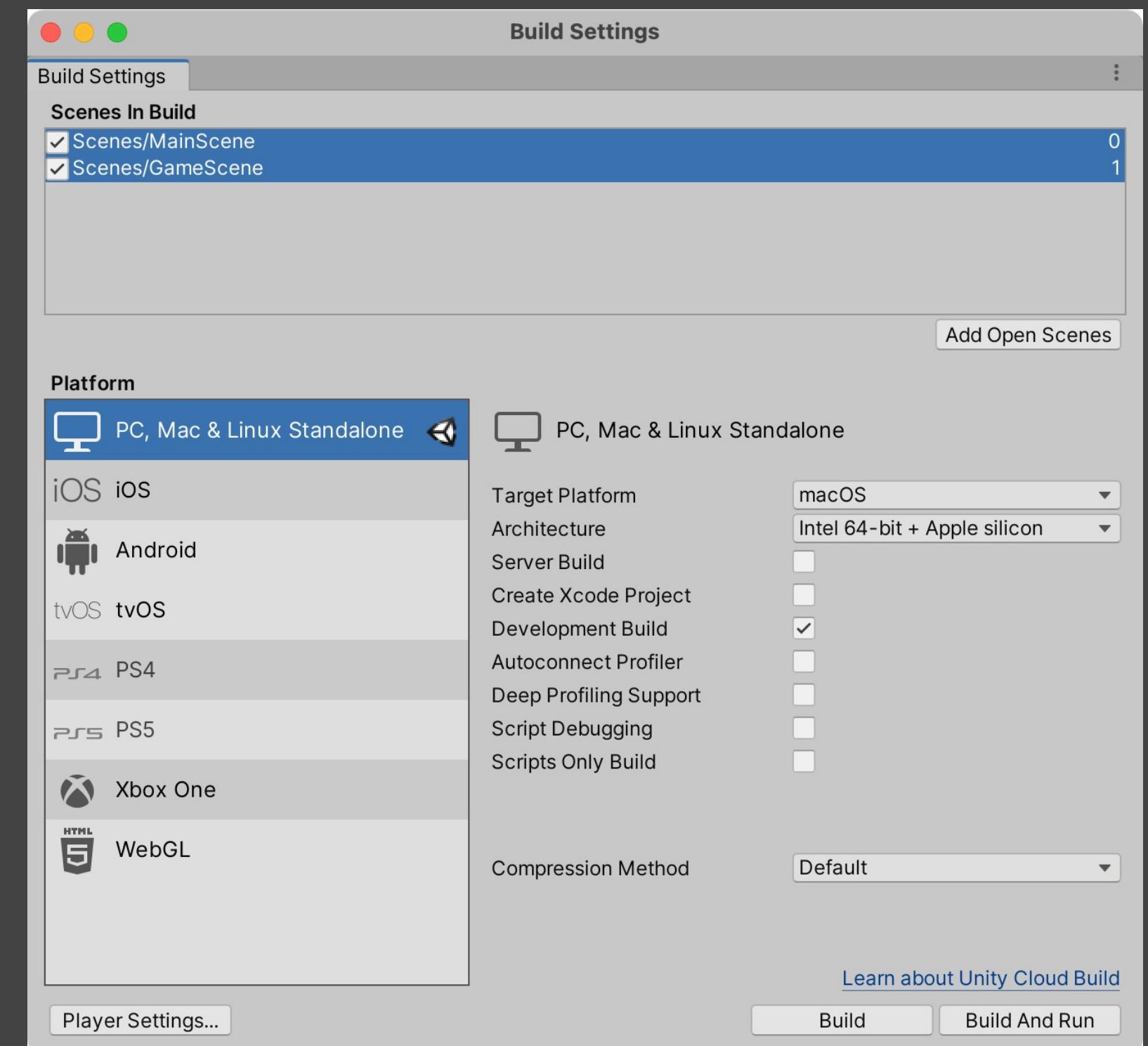
輸出視窗程式

- 開啟 Edit > Project Settings
 - 進入Player頁面，設定為視窗模式



Play!

- 執行2個遊戲，觀察 Created/Joined Room 訊息
 - 一個用 Unity Play
 - 一個使用 Build And Run 按鈕
- 如果發現較晚 Join Room的Client，會被當成 Master Client的話，可以去 Photon 的 settings 設定 fixed region，強制兩個 Client 進入同一個 Region。



練習 5分鐘

Create Player

- 進入 遊戲 Room 後，我們需要隨時可以更新玩家的資訊，因此我們必須建立一個**玩家**的參考物件

```
public class GameManager : MonoBehaviourPunCallbacks
{
    public static GameManager instance;
    public static GameObject localPlayer;
    ...
}
```

Create Player

- 我們使用 Unity 的 sceneLoaded 事件，來觀察是否 GameScene 被載入。
- 但是 MainScene 被載入時也會被呼叫，所以這邊加上判斷 PhotonNetwork.InRoom 來檢查現在是否已進入遊戲

```
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviourPunCallbacks
{
    ...

    void Start()
    {
        SceneManager.sceneLoaded += OnSceneLoaded;
        ...
    }

    ...

    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        if (!PhotonNetwork.InRoom)
        {
            return;
        }
    }
}
```

Create Player

- 然後使用 **TankPlayer** prefab 來產生玩家的坦克

```
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviourPunCallbacks
{
    ...

    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        if (!PhotonNetwork.InRoom)
        {
            return;
        }

        localPlayer = PhotonNetwork.Instantiate(
            "TankPlayer", new Vector3(0, 0, 0), Quaternion.identity, 0);
        Debug.Log("Player Instance ID: " + localPlayer.GetInstanceID());
    }
}
```

Network Object

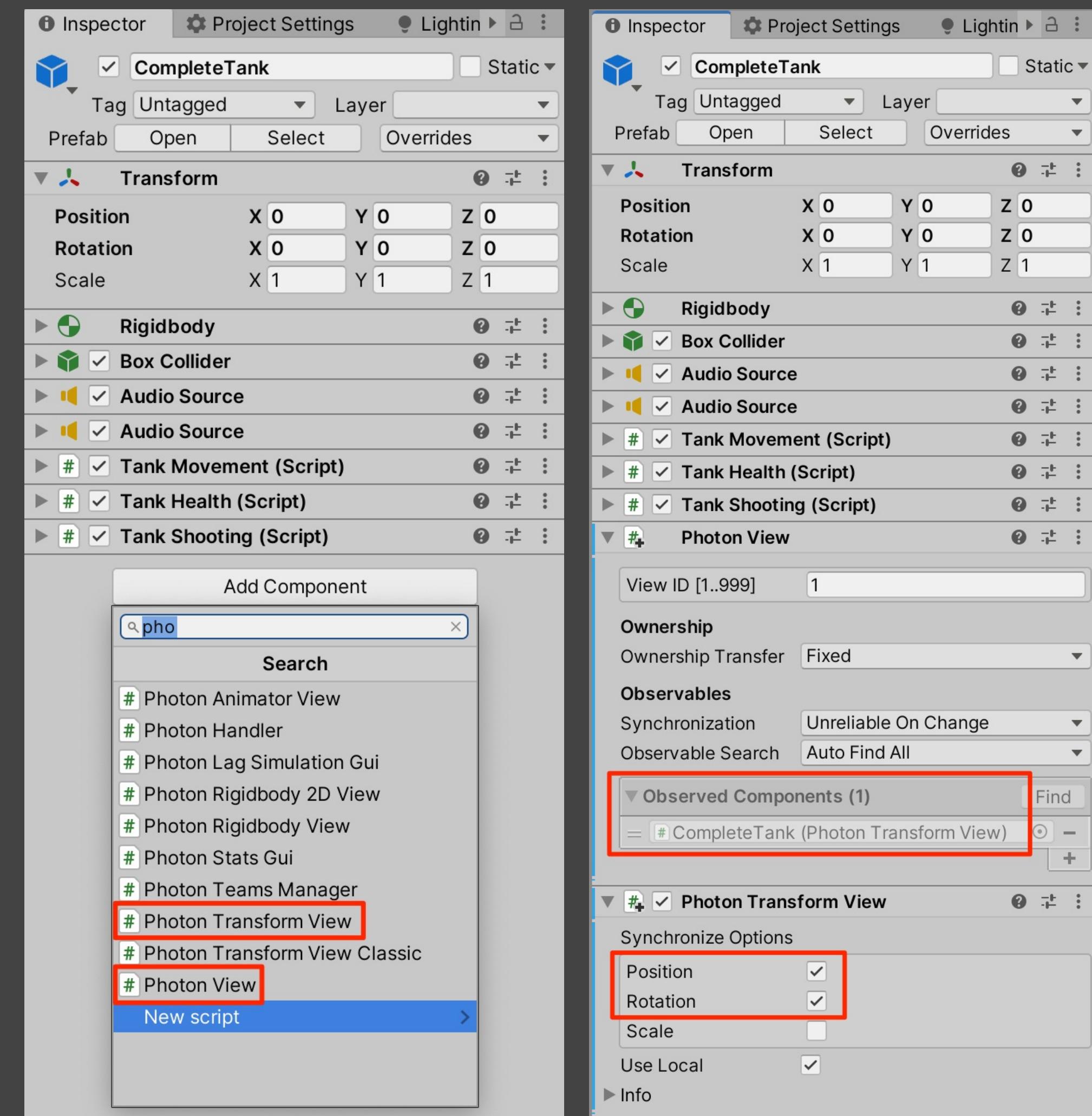
```
PhotonNetwork.Instantiate("PrefabName", position, quaternion, 0);
```

- Prefab
 - 必須有 PhotonView component
 - Photon 會從 Resources 目錄 尋找 Prefabs
- Owner：建立 Network Object 的Client，就是物件的 Owner (擁有者)
- Lifetime：建立者離開 Room 的話就會一起刪除
- Room Object：由 Master Client 使用
`PhotonNetwork.InstantiateRoomObject()` 建立

練習 5分鐘

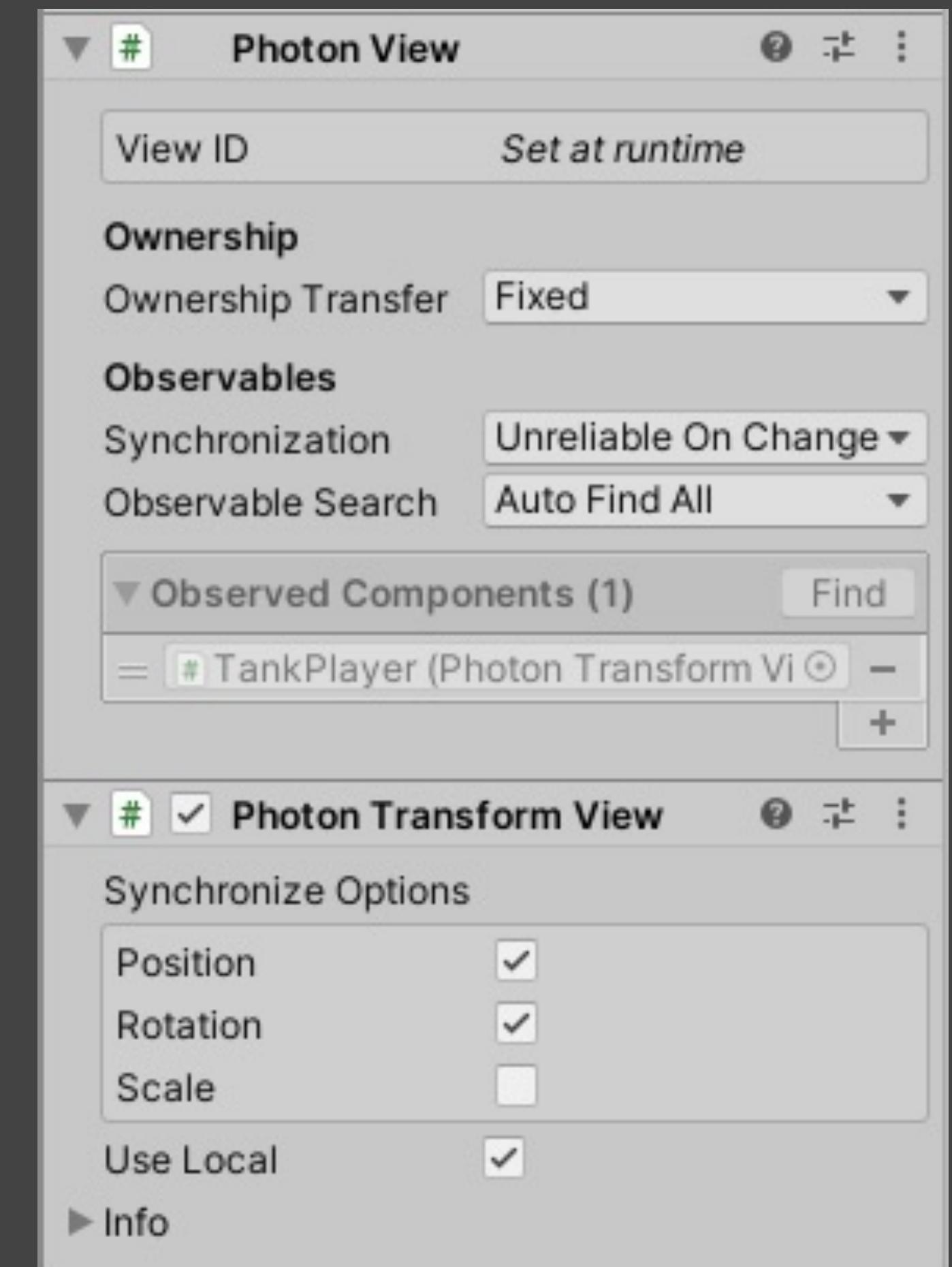
Create Tank Player Prefab

- 設定 Tank player prefab
 - 拖移 _Completed-Assets\Prefabs\CompleteTank.prefab 到場景
 - 加入 PhotonView 來自動同步玩家坦克的產生
 - 加入 PhotonTransformView 來自動同步坦克的位置及方向



Photon View

- A PhotonView identifies an object across the network ([viewID](#))
- Configures how controlling client updates remote instances
- Ownership Transfer Options
 - [Fixed](#): 不可轉換
 - [Request](#): 可以用 `PhotonView.RequestOwnership()` 來跟Owner 要求
 - [Takeover](#): 沒有Owner，可以用 `PhotonView.TransferOwnership()` 直接取得



https://doc-api.photonengine.com/en/pun/v2/class_photon_1_1_pun_1_1_photon_view.html

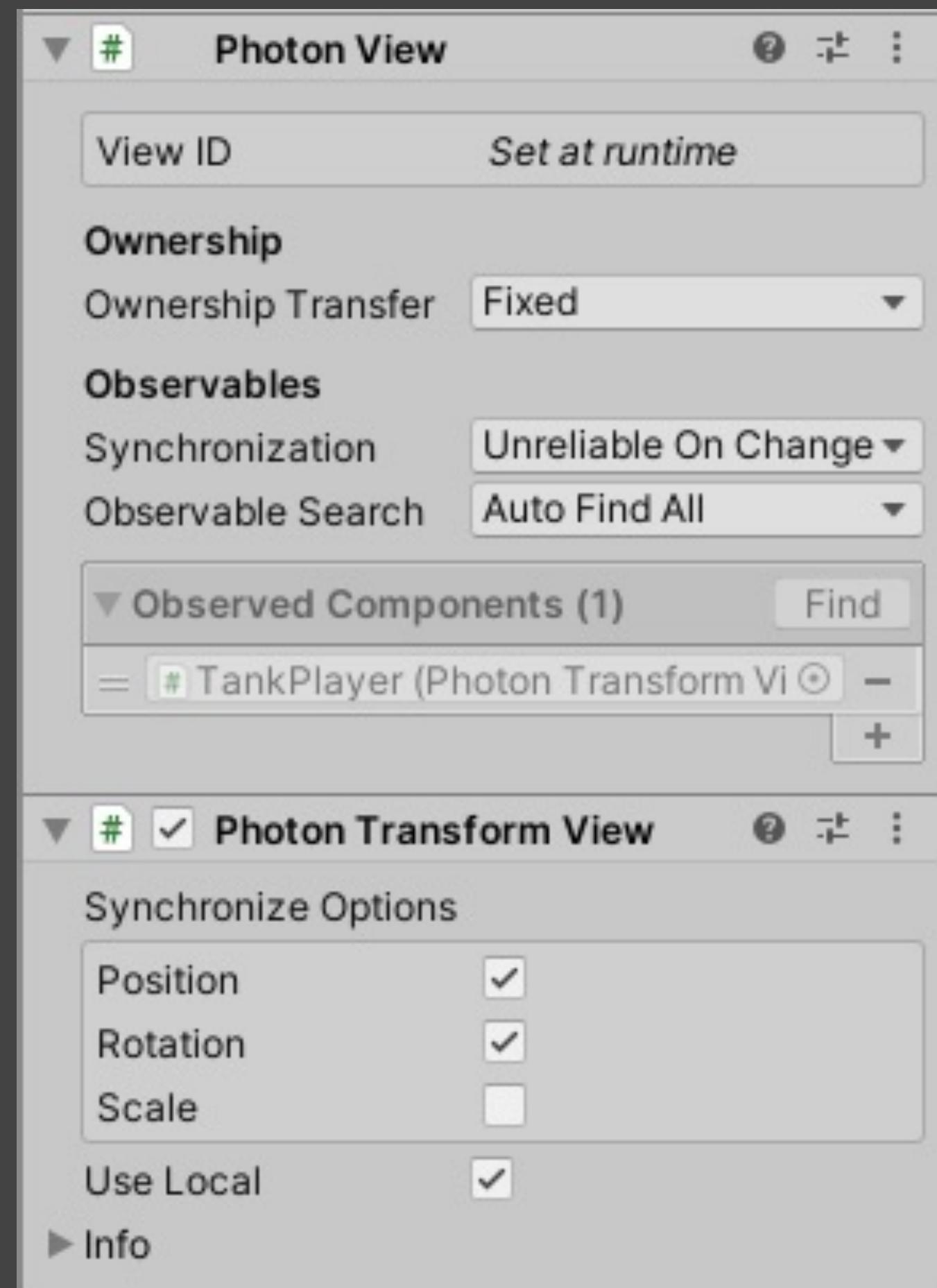
<https://doc.photonengine.com/zh-cn/pun/current/gameplay/ownershipandcontrol>

Photon View

- Observables / Synchronization：設定更新如何發送以及何時被發送，也會影響到 OnPhotonSerializeView 被調用的頻率。
 - Off：
 - 不同步。
 - 如果是使用 RPC 的話，建議可將Observe Option設定為Off，使物件從被觀察物件中除名，如此將能讓收發訊息的量再更輕減。
 - Unreliable：
 - 收送會按照順序，但是不可靠，更新有可能會丟失。
 - 通常用在更新頻率很快，適合每次更新都是最後的值，像是位置。但是像切換武器這樣觸發器就不適合。
 - 缺點是，如果同步的遊戲物體的位置沒有更改，也會發送更新，會浪費頻寬。
 - Unreliable on Change：
 - 與 Unreliable 一樣，收送會按照順序，但是不可靠，更新有可能會丟失。
 - 但是會檢查每一次更新的變化。如果數值與上一次一樣，就會停止傳送後續的更新，直到數值再次發生變化。
 - Reliable Delta Compressed：
 - 更新保證送到。
 - 將更新的每個值與它之前的值進行比較，未更改的值將跳過不傳送，降低傳送頻率，以保持低流量。
 - 對於可能會停止運動以及暫時不會更新的遊戲物體來說非常有用。例如玩家推的箱子，不會常常移動，只有被推時才會改變位置

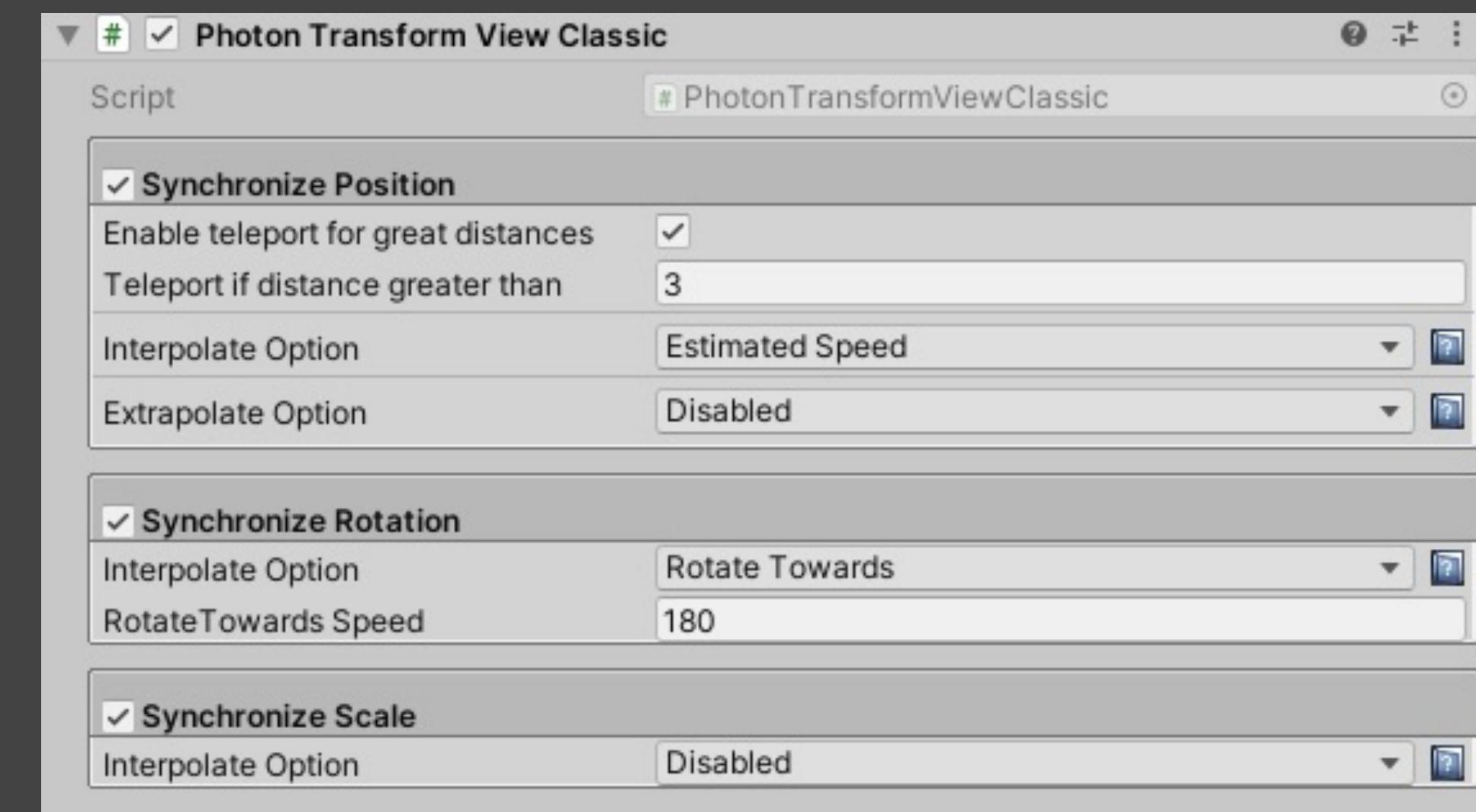
Photon Transform View

- 控制 Network Object 的位置、旋轉、縮放的同步
- 必須要設定到 PhotoView 的 **Observed Components** 列表中，才會有作用
- 會根據 PhotonView 的 Synchronization 設定決定同步方式



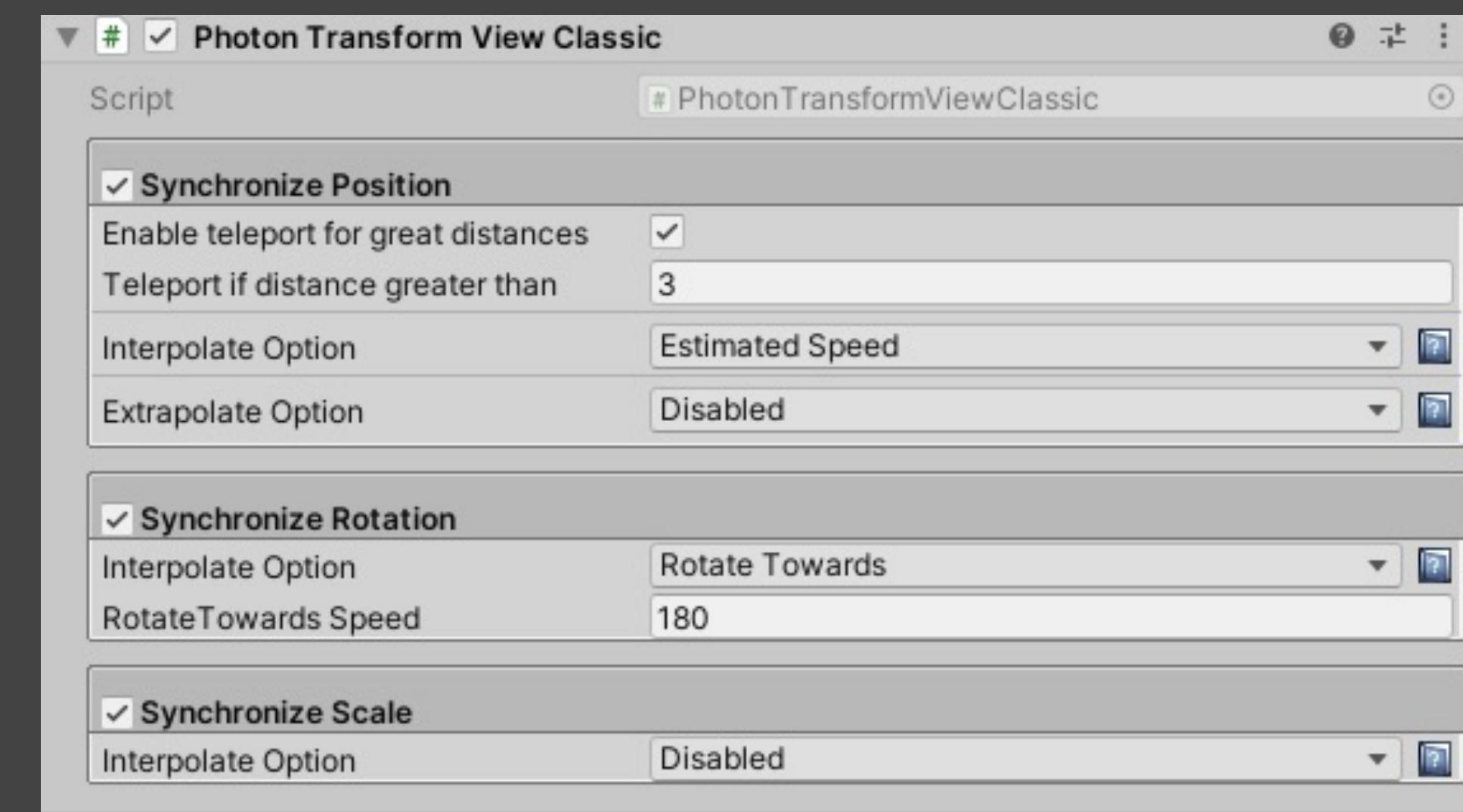
Photon Transform View Class

- 因為網路延遲 (lag)，會造成物件像抖動 (jitter) 或瞬移 (teleport) 的感覺，Photon 提供了下列方式來減緩這種感覺
- Synchronize Position
 - Enable teleport for great distances: 同步時，發現位置相差太遠的話，是否要直接跳過去
 - Teleport if distance grage then: 大於此距離就跳躍



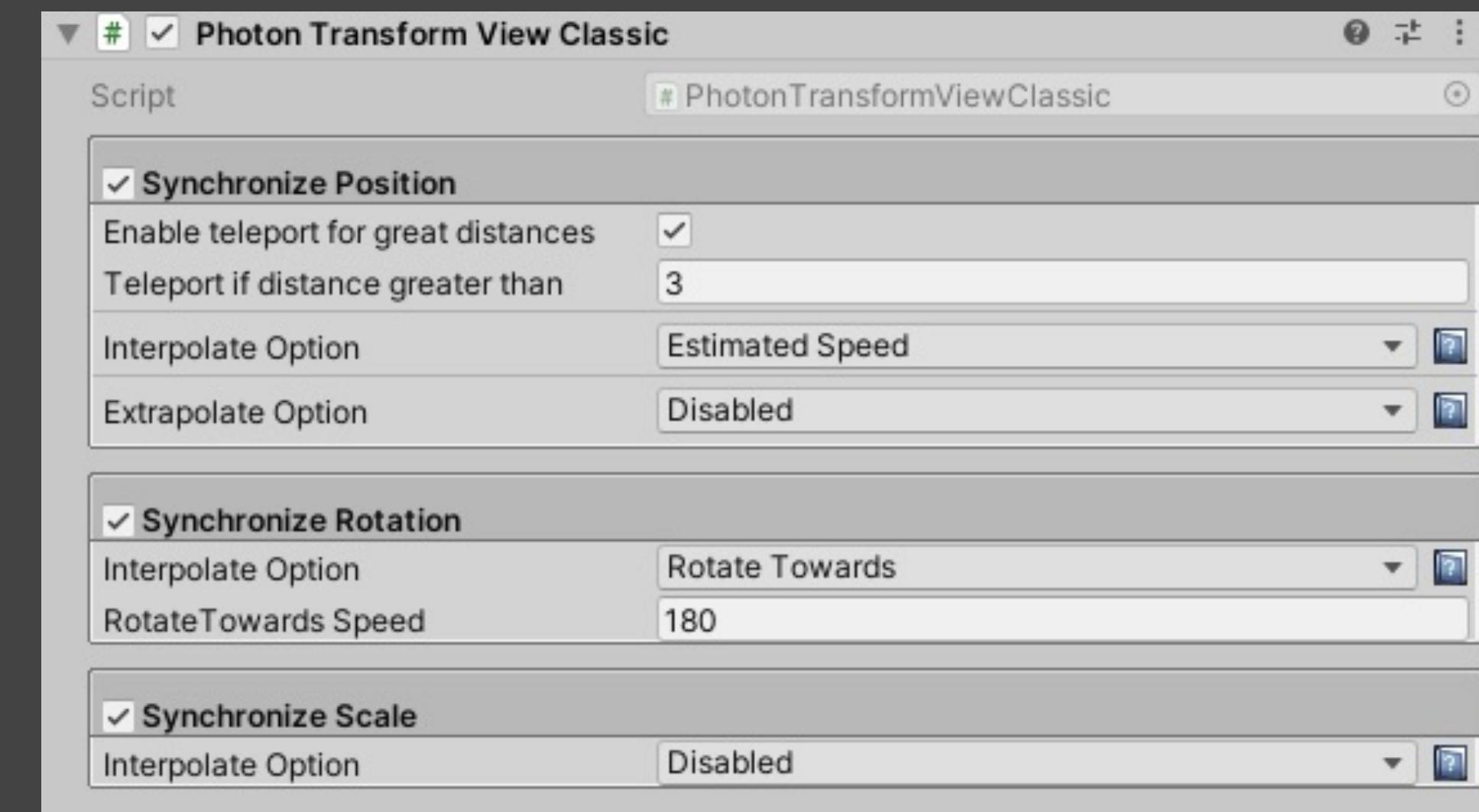
Photon Transform View Class

- Interpolate Option: 位置同步的內插方式
 - Disable : 不使用，直接跳到新的位置
 - Fixed Speed : 固定速度，適合移動速度不會變動的物體
 - Estimated Speed : 利用之前的座標與現在座標來評估速度，適合移動速度變化不會劇烈的物體，例如車輛
 - Synchronize Values : 在Script中直接給定目前的速度及迴轉速度，透過 PhotonTransformView.SetSynchronizedValues函式來指定給Photon，適合速度變化劇烈的物體，像是動作遊戲
 - Lerp : 線性插值，效果會像是先快速逼近，然後減速到新的位置



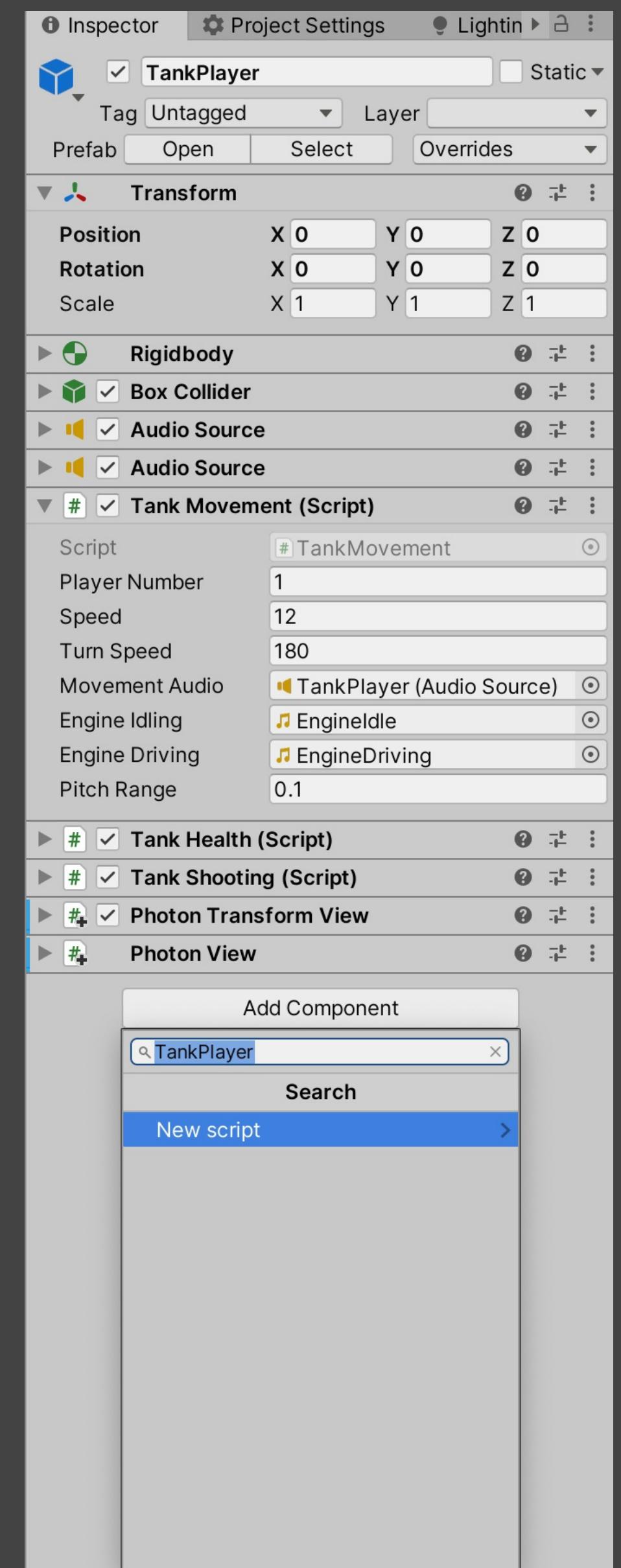
Photon Transform View Class

- Extrapolate Option: 位置同步的外插方式 (Dead Recoking)
 - 預設是關閉的。
 - 當網路太慢，或是更新不夠快時使用。當沒收到新位置時，用預測的方式來模擬新的位置，可以減少延遲的現象。
 - 不要用在速度變化太大，或是無法預測位置的物件上。



Tank Player Prefab

- 將 CompleteTank 改名為 TankPlayer
- 建立 Resources 目錄，再將 TankPlayer 移動到此目錄產生 TankPlayer 的 prefab
- 建立 TankPlayer script 用來管理玩家狀態
- 最後把 TankPlayer game object 從 GameScene 刪除，因為進入 Room 時，GameManager 會自動透過 prefab 建立玩家的坦克



練習 10分鐘

Tank Player Script

```
using Photon.Pun;

namespace Tanks
{
    public class TankPlayer : MonoBehaviourPunCallbacks
    {
        // Reference to tank's movement script, used to disable and enable control.
        private Complete.TankMovement m_Movement;
        // Reference to tank's shooting script, used to disable and enable control.
        private Complete.TankShooting m_Shooting;

        private void Awake()
        {
            m_Movement = GetComponent<Complete.TankMovement>();
            m_Shooting = GetComponent<Complete.TankShooting>();

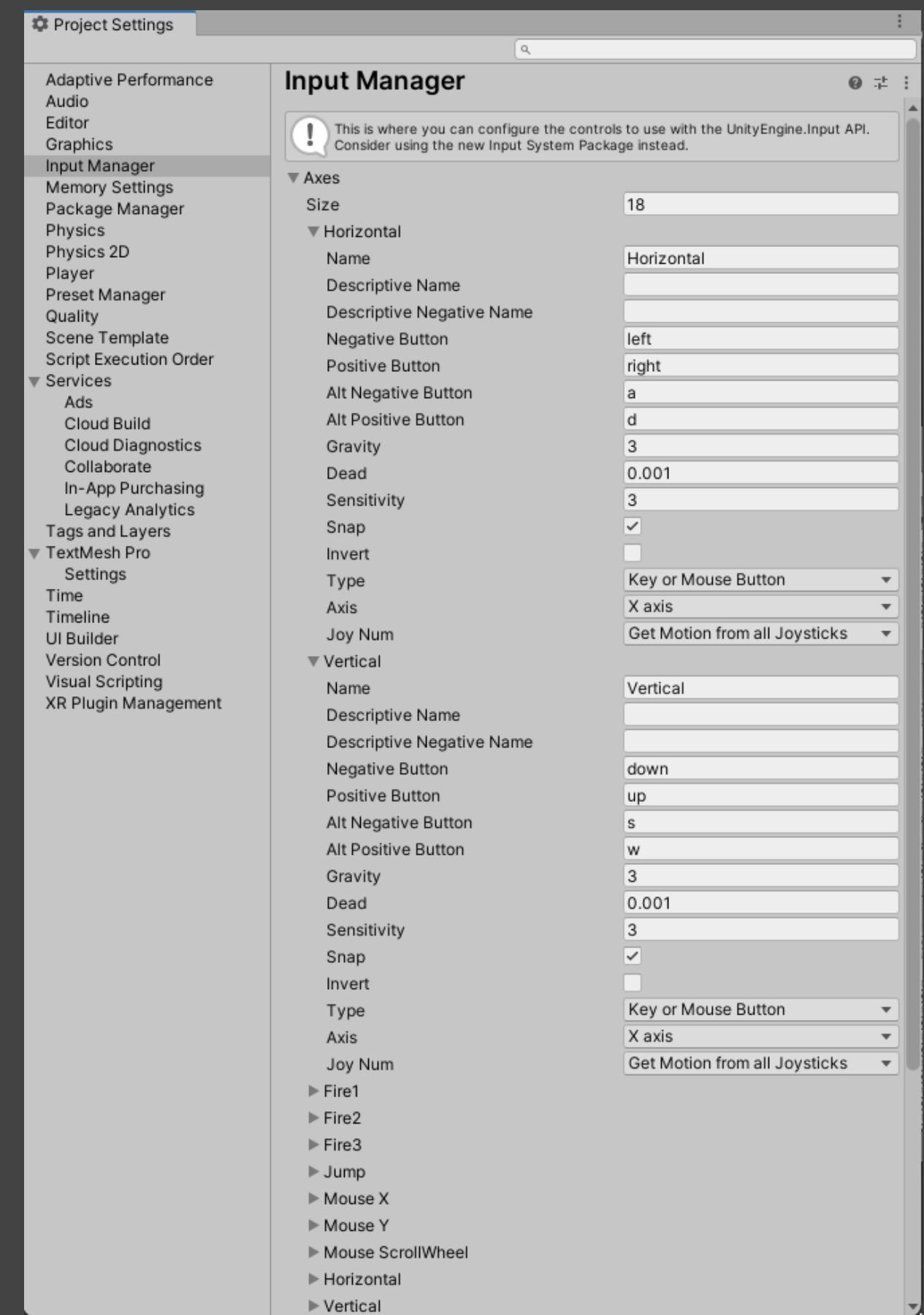
            if (!photonView.IsMine)
            {
                m_Movement.enabled = false;
                m_Shooting.enabled = false;

                enabled = false;
            }
        }
    }
}
```

Control Player

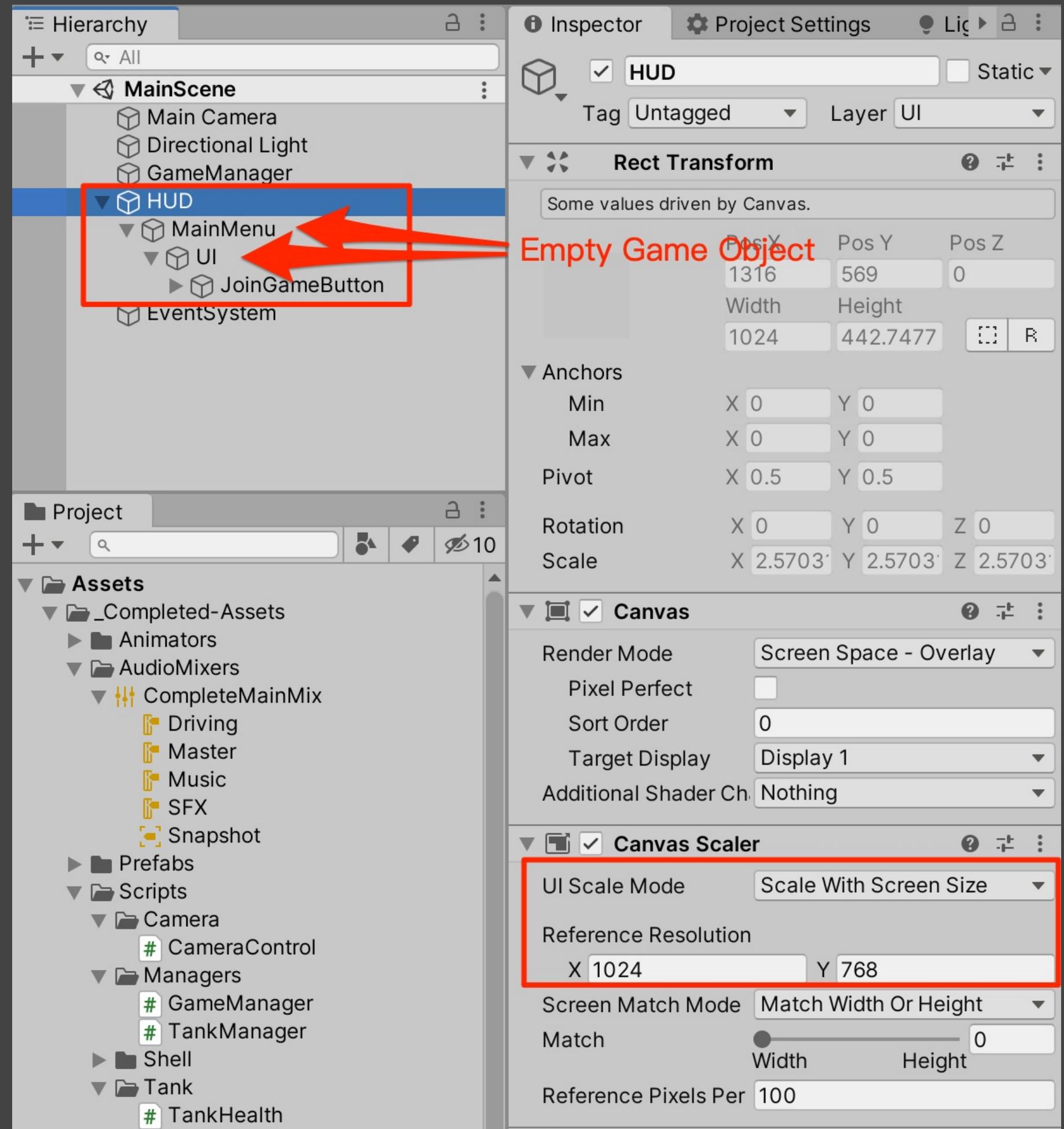
```
namespace Complete
```

```
{  
    public class TankMovement : MonoBehaviour  
    {  
        ...  
        private void Start ()  
        {  
            // The axes names.  
            m_MovementAxisName = "Vertical";  
            m_TurnAxisName = "Horizontal";  
  
            // Store the original pitch of the audio source.  
            m_OriginalPitch = m_MovementAudio.pitch;  
        }  
    }  
}
```



練習 5分鐘

Main Menu



Main Menu

- 接下來在 Scripts 目錄下新增一個 ExtensionMethods Script，用來擴展 Transform 類別，讓它可以根据名稱及類型來找尋子物件

```
using UnityEngine;

namespace Tanks
{
    public static class ExtensionMethods
    {
        public static T FindAnyChild<T>(this Transform trans, string name) where T : Component
        {
            for (var n = 0; n < trans.childCount; n++)
            {
                if (trans.GetChild(n).childCount > 0)
                {
                    var child = trans.GetChild(n).FindAnyChild<Transform>(name);
                    if (child != null)
                        return child.GetComponent<T>();
                }

                if (trans.GetChild(n).name == name)
                {
                    return trans.GetChild(n).GetComponent<T>();
                }
            }

            return default;
        }
    }
}
```

Main Menu

- 在 Scripts 目錄下新增 HUD Script ，並且設定為 Singleton 型式，然後讓主介面不會因為切換場景而消失，然後將此 Script 放到 HUD 物件上

```
using UnityEngine;

namespace Tanks
{
    public class HUD : MonoBehaviour
    {
        static HUD instance;

        void Awake()
        {
            if (instance != null)
            {
                DestroyImmediate(gameObject);
                return;
            }

            instance = this;
            DontDestroyOnLoad(gameObject);
        }
    }
}
```

Main Menu

- 在 Scripts 目錄下新增 **MainMenu** Script，並且設定為 Singleton 型式，以及繼承 **MonoBehaviourPunCallbacks**，然後將此 Script 放到 **MainMenu** 物件上

```
using Photon.Pun;

namespace Tanks
{
    public class MainMenu : MonoBehaviourPunCallbacks
    {
        static MainMenu instance;

        void Awake()
        {
            if (instance != null)
            {
                DestroyImmediate(gameObject);
                return;
            }

            instance = this;
            DontDestroyOnLoad(gameObject);
        }
    }
}
```

練習 5分鐘

Main Menu

- 修改 MainMenu Script，並且將 UI 物件 deactivate

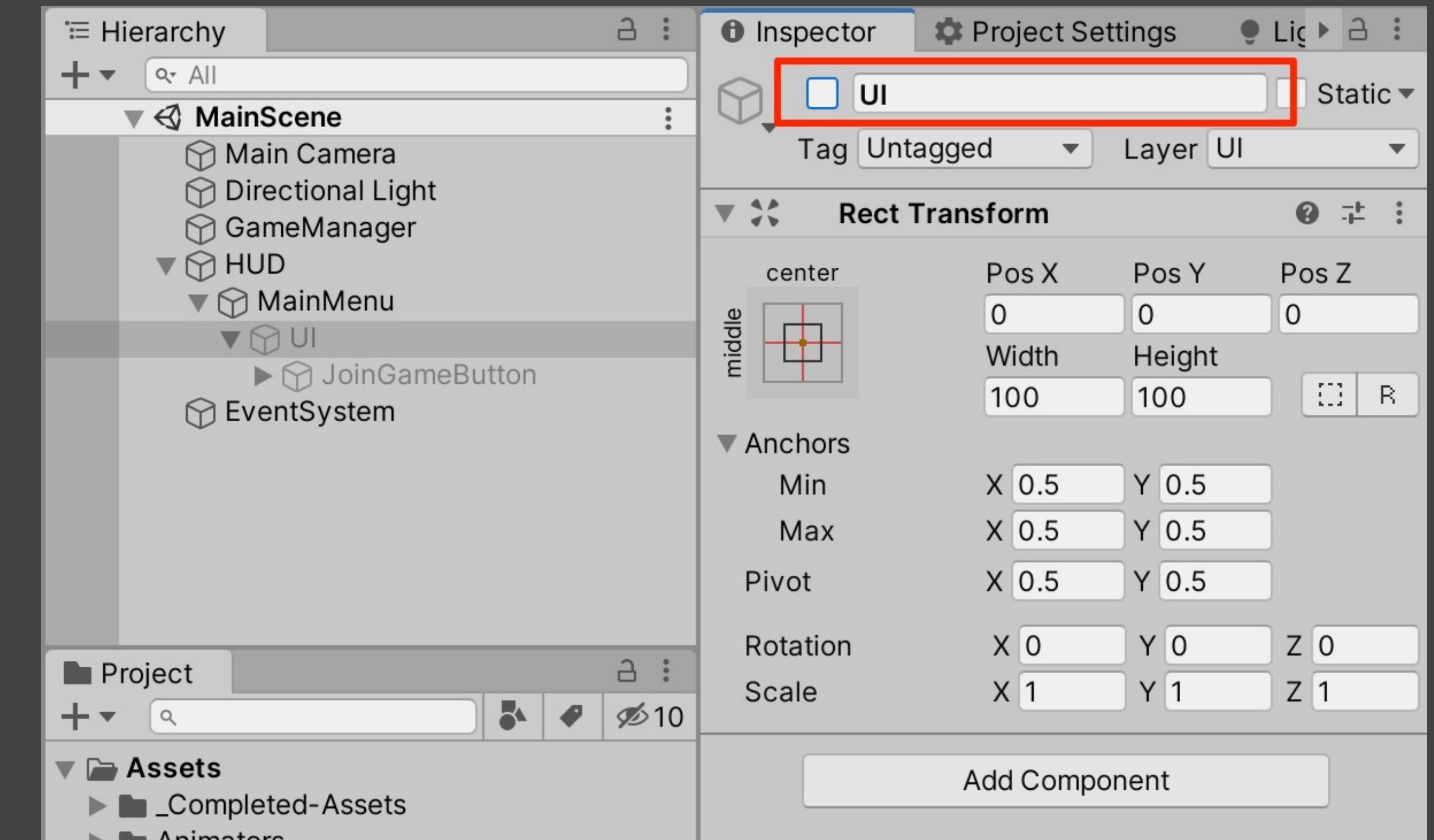
```
using Photon.Pun;
using UnityEngine;
using UnityEngine.UI;

namespace Tanks
{
    public class MainMenu : MonoBehaviourPunCallbacks
    {
        static MainMenu instance;
        private GameObject m_ui;
        private Button m_joinGameButton;

        void Awake()
        {
            if (instance != null) {
                DestroyImmediate(gameObject);
                return;
            }
            instance = this;

            m_ui = transform.FindAnyChild<Transform>("UI").gameObject;
            m_joinGameButton = transform.FindAnyChild<Button>("JoinGameButton");

            m_ui.SetActive(true);
            m_joinGameButton.interactable = false;
        }
    }
}
```



Main Menu

- 替 `MainMenu` Script 加上下面函式，讓 `Join Game` 的按鈕在連接上 Photon Master Server 後才顯示

```
public override void OnConnectedToMaster()
{
    m_joinGameButton.interactable = true;
}
```

Main Menu

- 替 MainMenu Script 加上下面函式，讓主選單一進入遊戲後就自動關閉

```
public override void OnEnable()
{
    // Always call the base to add callbacks
    base.OnEnable();

    SceneManager.sceneLoaded += OnSceneLoaded;
}

public override void OnDisable()
{
    // Always call the base to remove callbacks
    base.OnDisable();

    SceneManager.sceneLoaded -= OnSceneLoaded;
}

private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
{
    m_ui.SetActive(!PhotonNetwork.InRoom);
}
```

練習 5分鐘

Spawn Points

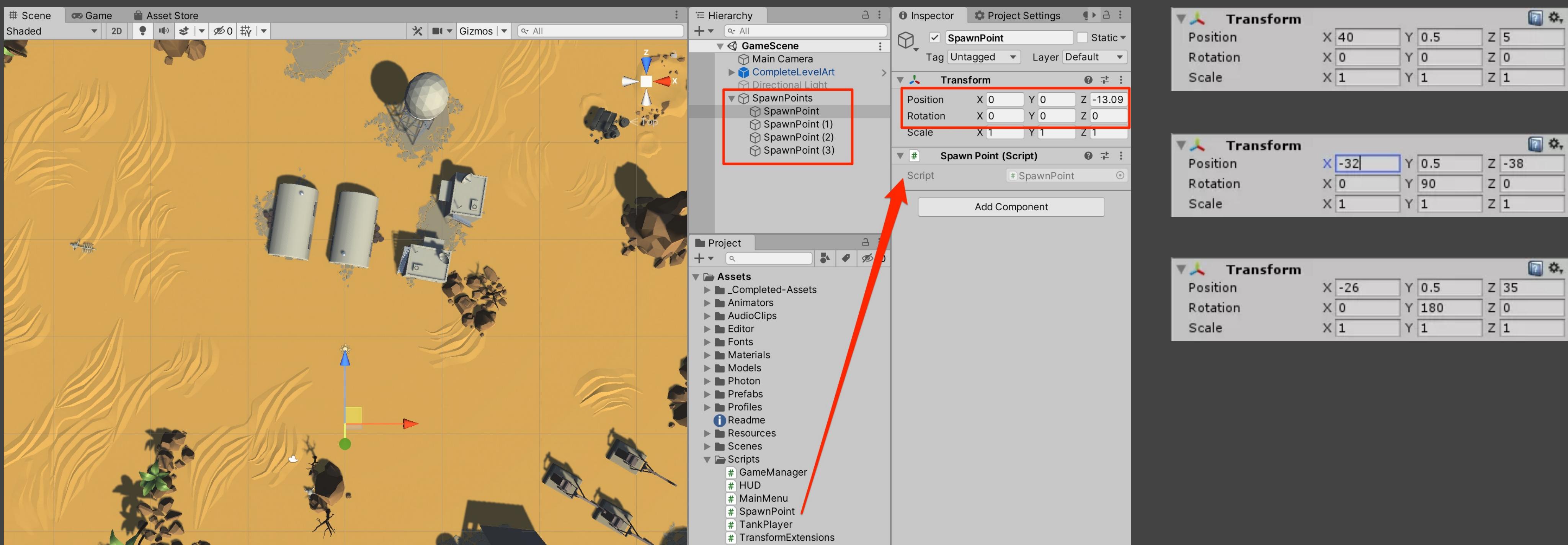
- 在 Scripts 目錄下新增 SpawnPoint Script

```
using UnityEngine;

namespace Tanks
{
    public class SpawnPoint : MonoBehaviour
    {
        void Awake()
        {
            gameObject.SetActive(false);
        }
    }
}
```

Spawn Points

- 在 Game Scene 中建立一個 empty Game Object，並命名為 **SpawnPoint**，並把 **SpawnPoint Script** 附加上去，然後複製出多個，並設定其位置



Spawn Points

- 在 `GameManager` 中加上下面函式，用來抓取所有重生點

```
public static List<GameObject> GetAllObjectsOfTypeInScene<T>()
{
    var objectsInScene = new List<GameObject>();

    foreach (var go in (GameObject[])Resources.FindObjectsOfTypeAll(typeof(GameObject))) {
        if (go.hideFlags == HideFlags.NotEditable ||
            go.hideFlags == HideFlags.HideAndDontSave)
            continue;

        if (go.GetComponent<T>() != null)
            objectsInScene.Add(go);
    }

    return objectsInScene;
}
```

Spawn Points

```
public class GameManager : MonoBehaviourPunCallbacks
{
    ...
    private GameObject defaultSpawnPoint;
    ...
    void Awake()
    {
        ...
        defaultSpawnPoint = new GameObject("Default SpawnPoint");
        defaultSpawnPoint.transform.position = new Vector3(0, 0, 0);
        defaultSpawnPoint.transform.SetParent(transform, false);
    }
    ...
    private Transform GetRandomSpawnPoint()
    {
        var spawnPoints = GetAllObjectsOfTypeInScene<SpawnPoint>();
        return spawnPoints.Count == 0
            ? defaultSpawnPoint.transform
            : spawnPoints[Random.Range(0, spawnPoints.Count)].transform;
    }
}
```

Spawn Points

- 修改 `OnSceneLoaded` 函式，讓坦克可以在隨機位置出現

```
private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
{
    if (!PhotonNetwork.InRoom)
    {
        return;
    }

    var spawnPoint = GetRandomSpawnPoint();

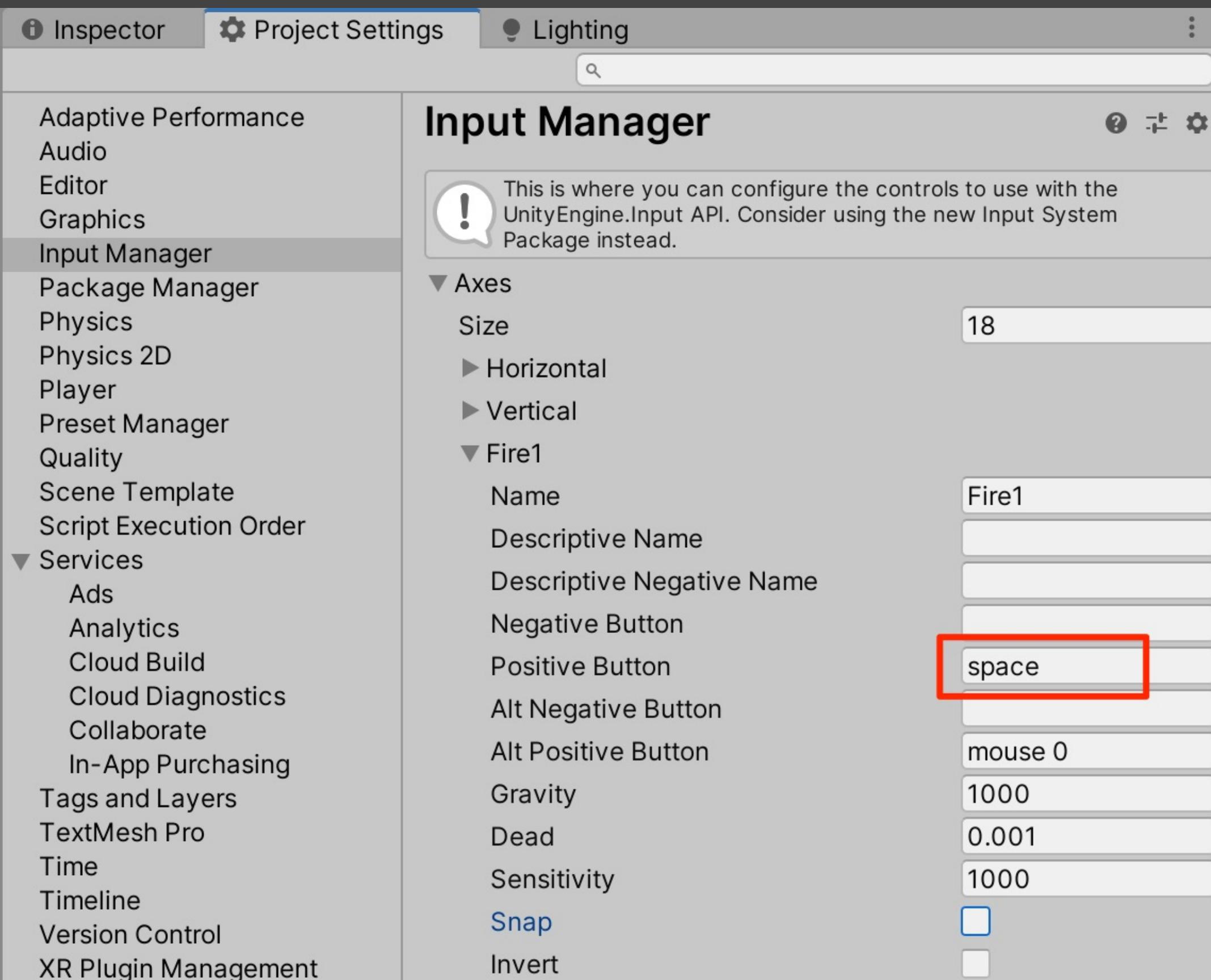
    localPlayer = PhotonNetwork.Instantiate(
        "TankPlayer",
        spawnPoint.position,
        spawnPoint.rotation,
        0);

    Debug.Log("Player Instance ID: " + localPlayer.GetInstanceID());
}
```

練習 10分鐘

Shoot

- 修改Project Settings > Input Manager 設定，使用空白鍵發射砲彈



Shoot Method 1
PhotonNetowrk.Instantiate

Shoot Method 1

PhotonNetowrk.Instantiate

- 將 CompleteShell prefab 移至 Resources 目錄下
- 新增 PhotonView 及 PhotonTransformView 到 CompleteShell prefab 上
- 修改 TankShooting script 如下，並且繼承 MonoBehaviourPunCallbacks

```
public class TankShooting : MonoBehaviourPunCallbacks
{
    ...
    public override void OnEnable()
    {
        base.OnEnable();
        ...
    }
    ...
}

}
```

Shoot Method 1

PhotonNetowrk.Instantiate

```
public class TankShooting : MonoBehaviourPunCallbacks
{
    ...
    private void Fire ()
    {
        // Set the fired flag so only Fire is only called once.
        m_Fired = true;

        // Create an instance of the shell and store a reference to it's rigidbody.
        GameObject shellInstance = PhotonNetwork.Instantiate(
            "CompleteShell",
            m_FireTransform.position,
            m_FireTransform.rotation,
            0);

        Rigidbody body = shellInstance.GetComponent<Rigidbody>();

        // Set the shell's velocity to the launch force in the fire position's forward direction.
        body.velocity = m_CurrentLaunchForce * m_FireTransform.forward;

        // Change the clip to the firing clip and play it.
        m_ShootingAudio.clip = m_FireClip;
        m_ShootingAudio.Play ();

        // Reset the launch force. This is a precaution in case of missing button events.
        m_CurrentLaunchForce = m_MinLaunchForce;
    }
}
```

練習 10分鐘

Shoot Method 2

PUN Remote Procedure Call (RPC)

Remote Procedure Calls

- 遠端程序呼叫：呼叫同一個房間裡的其他遠端客戶的 method
- 要啟用 method 的遠端程序呼叫，必須使用 PunRPC attribute

```
[PunRPC]  
void ChatMessage(string a, string b)  
{  
    Debug.Log(string.Format("ChatMessage {0} {1}", a, b));  
}
```

- 要呼叫被標記為 PunRPC 的函示，需要透過 PhotonView 物件

```
PhotonView photonView = PhotonView.Get(this);  
photonView.RPC("ChatMessage", RpcTarget.All, "jup", "and jup.");
```

- Photon 會呼叫遠端對應的同一個網路物件(PhotonView) 的 Method

Remote Procedure Calls

- 注意
 - RPC method 的 script 的 GameObject 必須要擁有 PhotonView Component
 - RPC method 不能是 static method
 - RPC method 不能是 Generic methods
 - RPC method 可以有回傳值，但是沒有用處
 - 如果繼承時 Override 的 method 也要加上 PunRPC attribute
 - 不要添加相同的 script(有 RPC method) 在同一個 GameObject，會無法確認哪個 script 的 method 被呼叫
 - 不要有 overload 的 RPC method
 - 不要有 optional parameters

Remote Procedure Calls

- 注意

- 如果要傳遞 object array，先轉成 object type

```
object[ ] objectArray = GetDataToSend();
photonView.RPC("RpcWithObjectArray", target, objectArray as object);
```

```
[PunRPC]
void RpcWithObjectArray(object[ ] objectArray)
{
    // ...
}
```

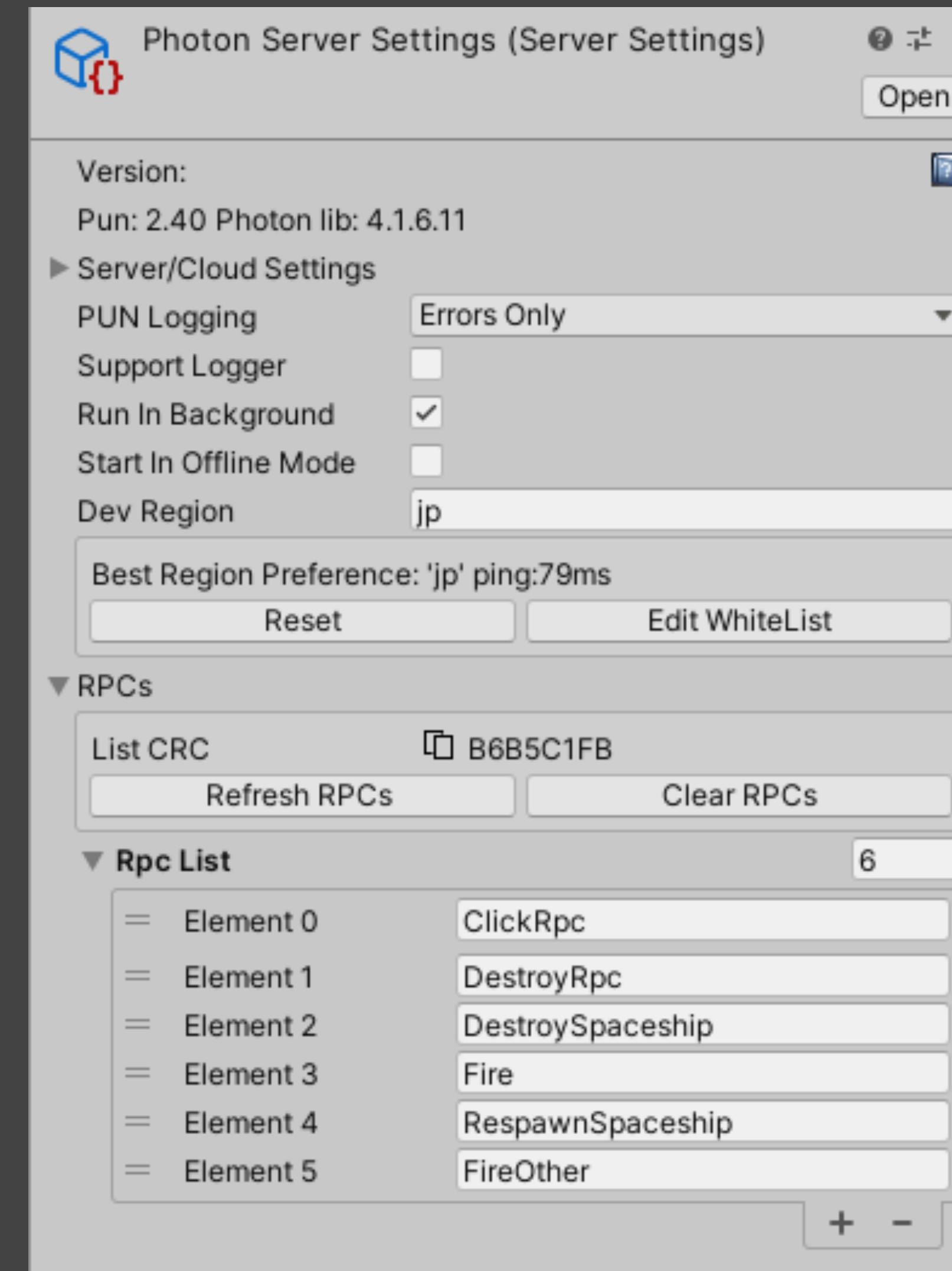
Remote Procedure Calls

- Targets (傳送對象)

```
public enum RpcTarget
{
    All,           // 送給所有人，自己會立刻執行
    Others,        // 送給所有人，自己不會執行
    MasterClient,
    AllBuffered,   // 送給所有人，自己會立刻執行，後來進來的也會收到
    OthersBuffered, // 送給所有人，自己不會執行，後來進來的也會收到
    AllViaServer,  // 送給所有人，會透過Server決定順序
    AllBufferedViaServer
    // 送給所有人，會透過Server決定順序，後來進來的也會收到
}
```

Remote Procedure Calls

- Shortcuts For RPC Names



Shoot Method 2

PUN Remote Procedure Call (RPC)

- 修改 TankShooting script 如下，並且繼承 MonoBehaviourPunCallbacks

```
public class TankShooting : MonoBehaviourPunCallbacks
{
    ...
    public override void OnEnable()
    {
        base.OnEnable();
        ...
    }
    ...
}
```

Shoot Method 2

PUN Remote Procedure Call (RPC)

```
public class TankShooting : MonoBehaviourPunCallbacks
{
    ...
    private void Fire ()
    {
        m_Fired = true;

        Rigidbody shellInstance = Instantiate (m_Shell, m_FireTransform.position, m_FireTransform.rotation) as Rigidbody;
        photonView.RPC ("FireOther", RpcTarget.Others, m_FireTransform.position);

        shellInstance.velocity = m_CurrentLaunchForce * m_FireTransform.forward;

        m_ShootingAudio.clip = m_FireClip;
        m_ShootingAudio.Play ();

        m_CurrentLaunchForce = m_MinLaunchForce;
    }

    [PunRPC]
    private void FireOther (Vector3 pos)
    {
        m_Fired = true;

        Rigidbody shellInstance = Instantiate (m_Shell, pos, m_FireTransform.rotation) as Rigidbody;
        shellInstance.velocity = m_CurrentLaunchForce * m_FireTransform.forward;
        m_CurrentLaunchForce = m_MinLaunchForce;
    }
}
```

練習 10分鐘

作業2

- 使用”Q”及“E”按鍵旋轉砲台
- 同步砲台的方向
- 讓砲彈掉落點一樣
- Tip
 - 發射點 (`FireTransform`) 要如何跟著砲台 (`TankTurret`) ?
 - 取得砲台的 Transform : 使用之前寫的 `transform.FindAnyChild<Transform>()` 函式
 - 旋轉砲台可以使用 `Transform.Rotate()` 函式
 - 旋轉砲台的數值 可以參考 Tank 轉彎的計算方式 (`TankMovement.cs`)
 - 請思考為何砲彈掉落點不一樣 ?
 - `photonView.RPC` 可以傳送多個參數

參考資料

References

- Photon Fusion 架構及技術介紹
- Photon 技術講座- 三大連線產品比較(PUN2, Photon Bolt 與 Quantum)
- Unity 官方教學的Tanks!
 - <https://unity3d.com/learn/tutorials/projects/tanks-tutorial>