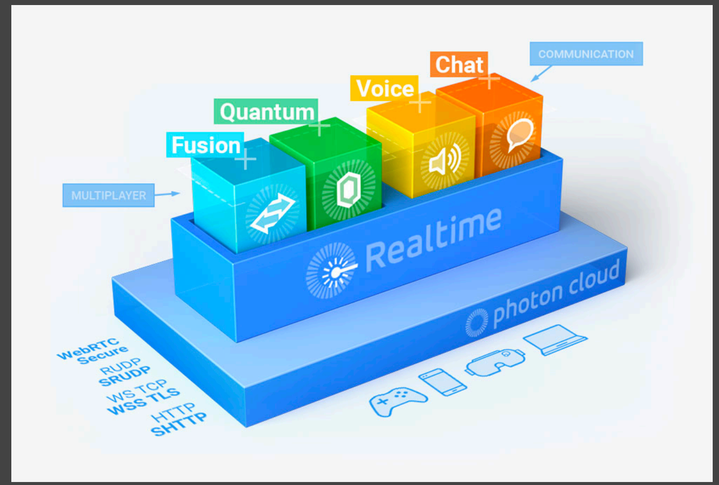
Commercial Server Solutions
Photon Unity Network

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



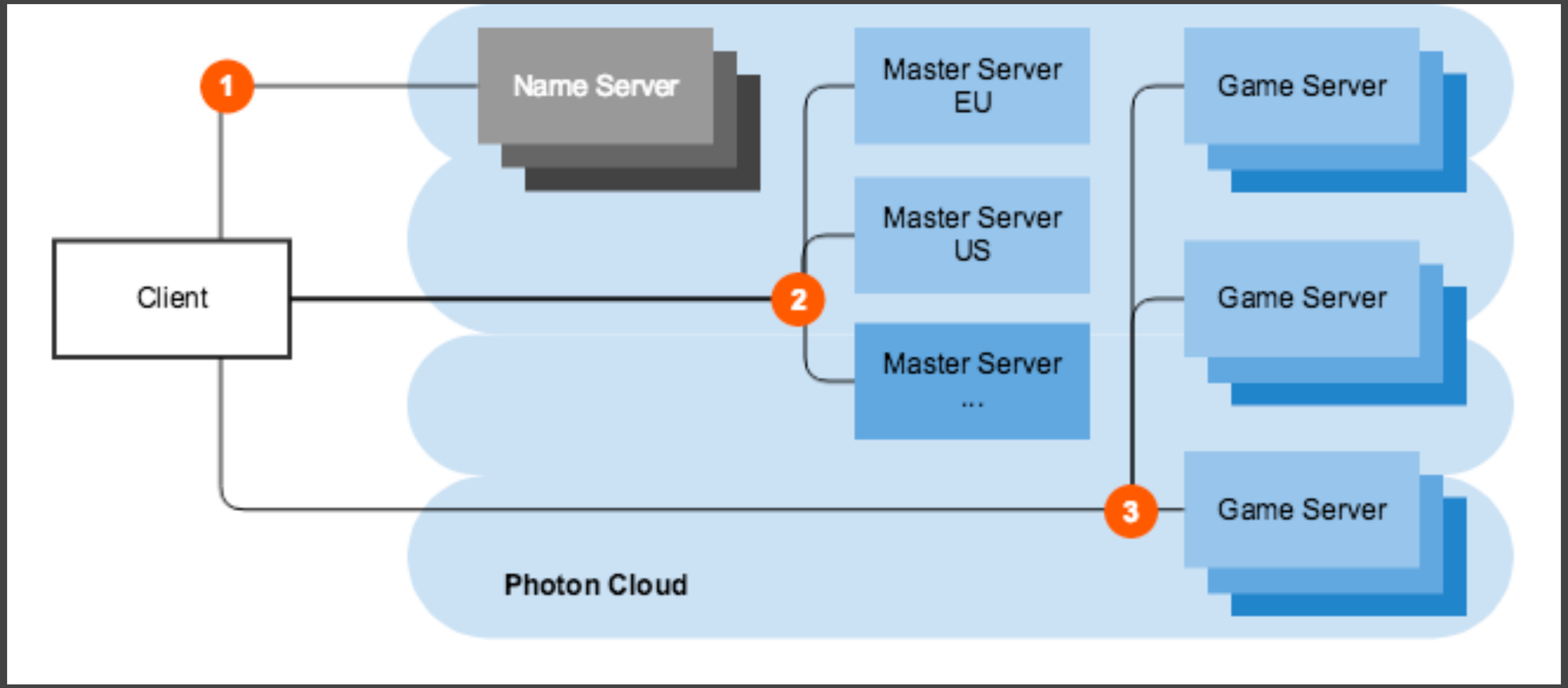
https://www.photonengine.com/en-US/Photon

Photon

	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#攥寫需要的遊戲邏輯
Setup	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

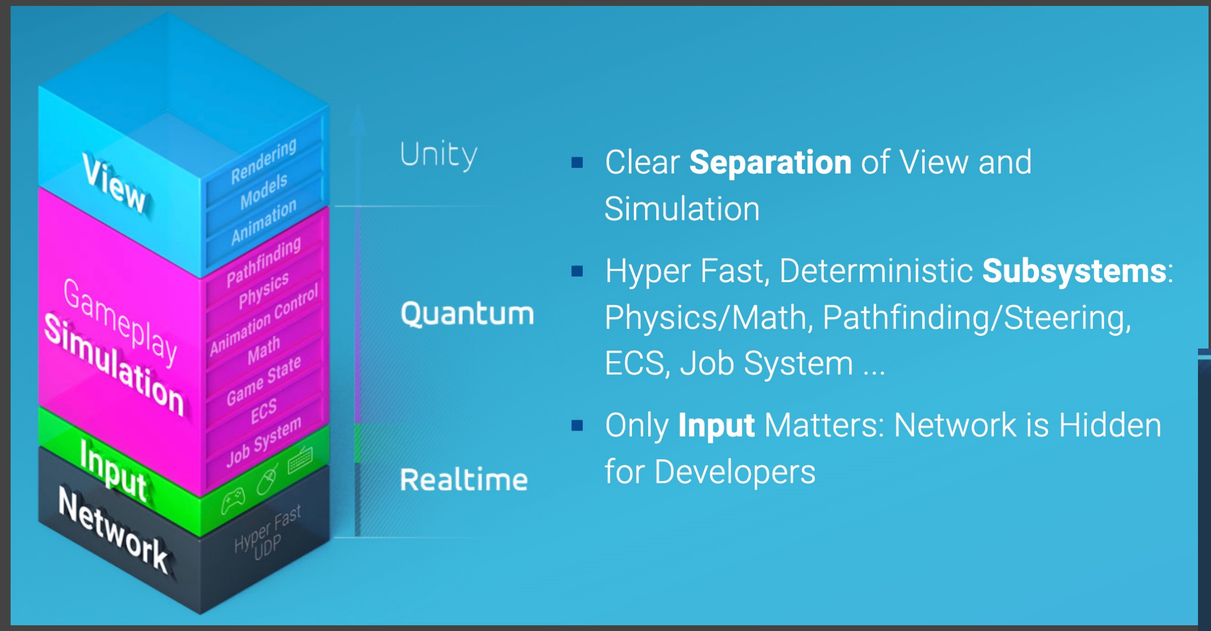
Photon Realtime



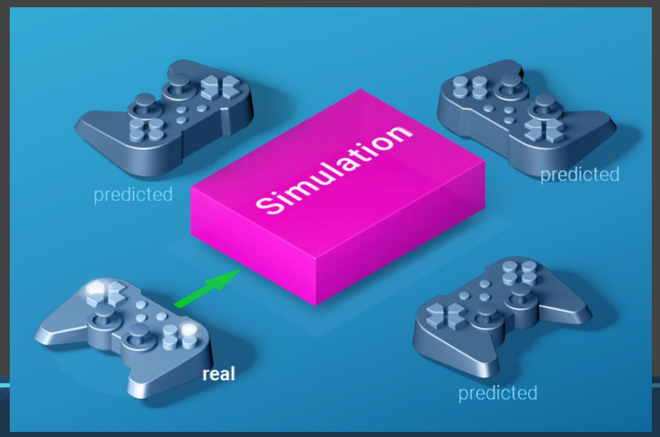
https://doc.photonengine.com/zh-tw/pun/current/connection-and-authentication/regions

Photon Quantum

適合快節奏多人即時連線動作遊戲



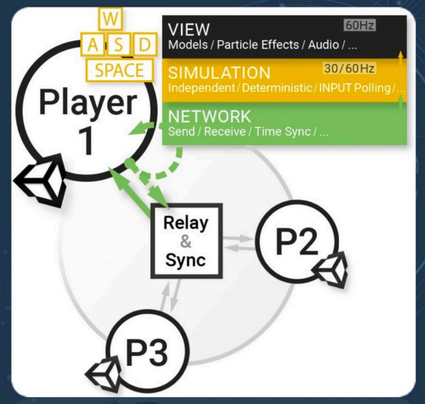
- Clear **Separation** of View and Simulation
- Hyper Fast, Deterministic **Subsystems**: Physics/Math, Pathfinding/Steering, ECS, Job System ...
- Only **Input** Matters: Network is Hidden for Developers



photon

Deterministic Architecture

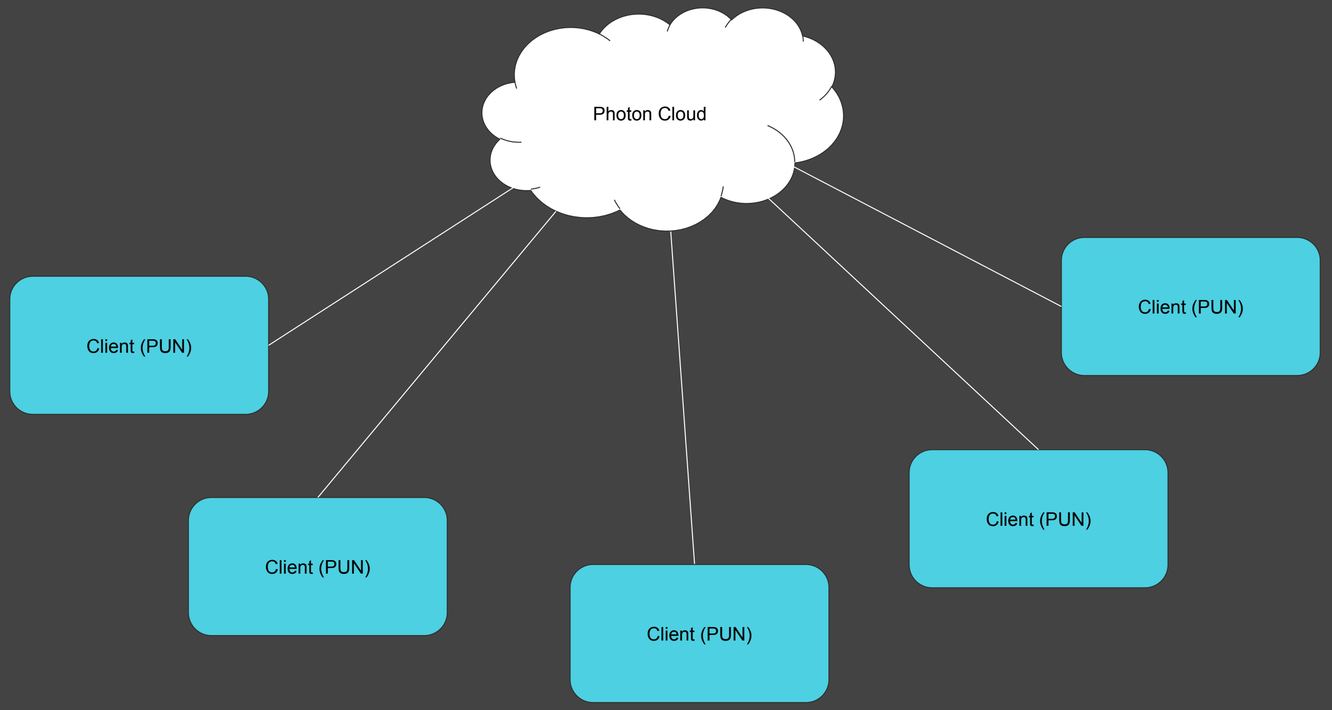
- 在玩家之間只會傳送輸入的操作 (key/mouse/touch Inputs), 網路頻寬的消耗非常低~
- 當前最棒的預測 **prediction**、回溯 **rollback** 的 Deterministic 演算法, 而不再須要 LockStepping 的作法.
- 不須要計算網路延遲的內插(interpolation)補償, 所以不會有假的、人為的、計算式的內插遊戲狀態. —> **玩起來, 非常絕對的公平！**



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

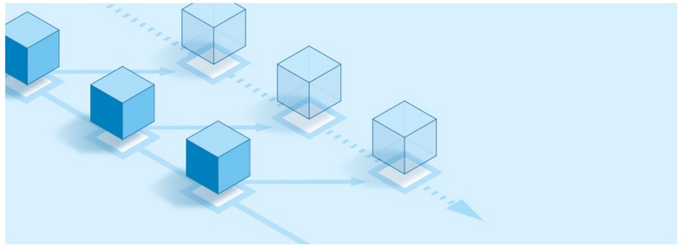
Photon Unity Network

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.



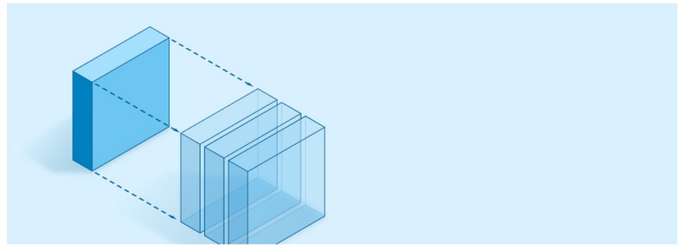
https://www.photonengine.com/zh-TW/PUN

Photon Fusion



Tick-based Simulation

Core feature of stable and accurate networking. Foundation for client side prediction and snapshot interpolation.



Client-side Prediction

Give players an instant response to their own inputs without giving up server authority, even under high latency and network loss.

Full Physics Prediction

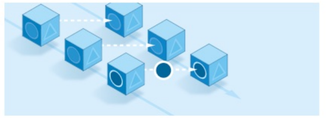
Get the best experience out of complex physics interactions between players and objects by using Full Physics Prediction.

Snapshot Interpolation

Use automated or custom interpolation for smooth visual rendering even with bad internet conditions.

Lag Compensation

Built in Lag Compensated Hitboxes with easy to use API for Esport grade game mechanics.

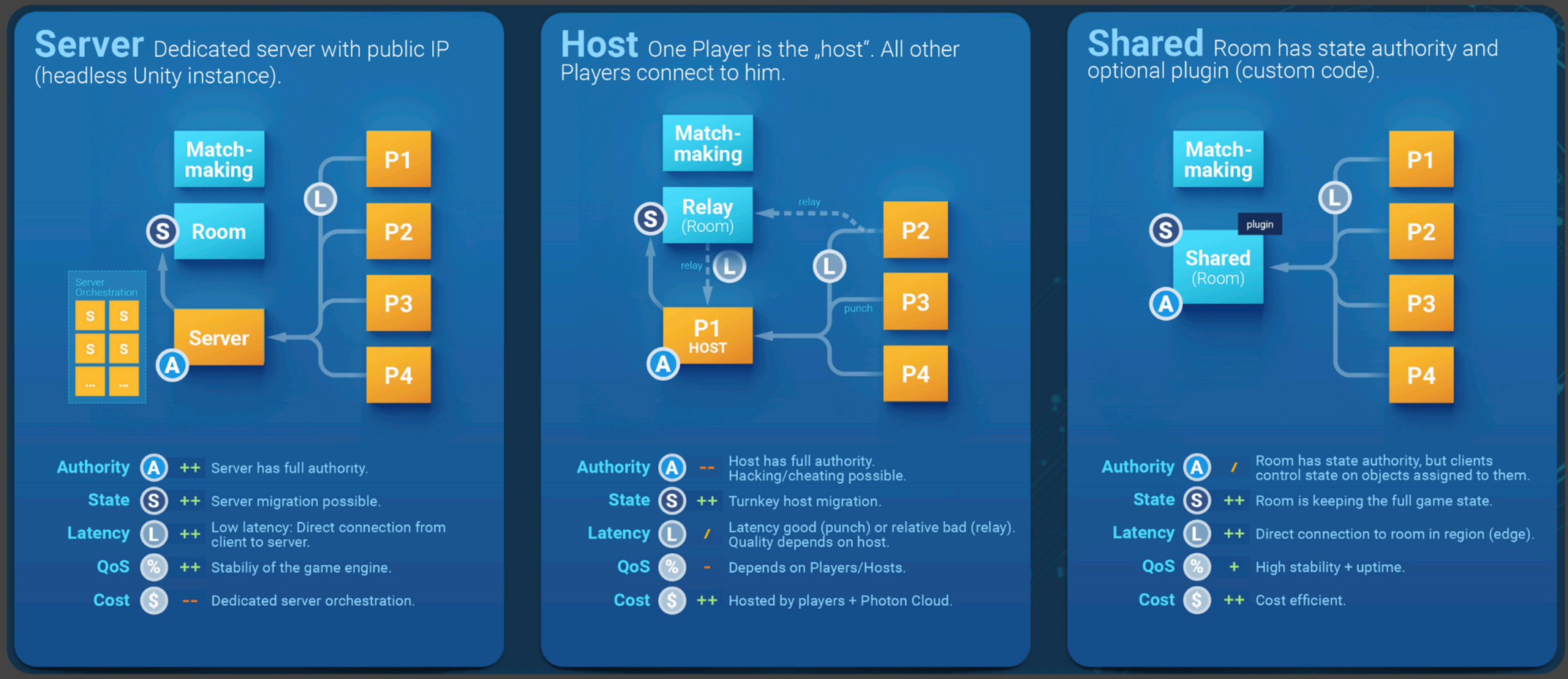


Replication Systems

Best in class implementations of replication algorithms for Delta Snapshots and Eventual Consistency with Interest Management.

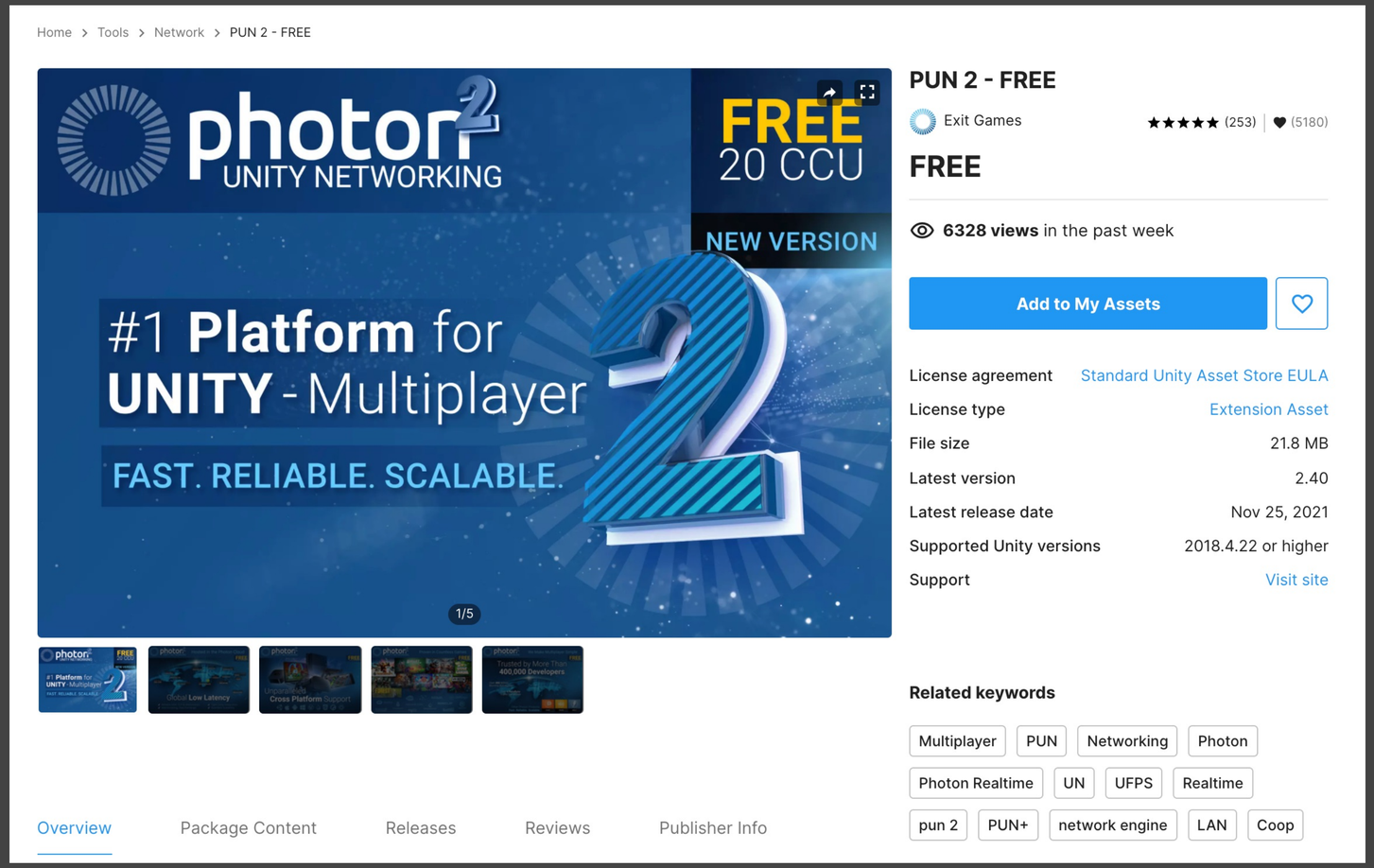
https://www.photonengine.com/en-US/Fusion

Photon Fusion



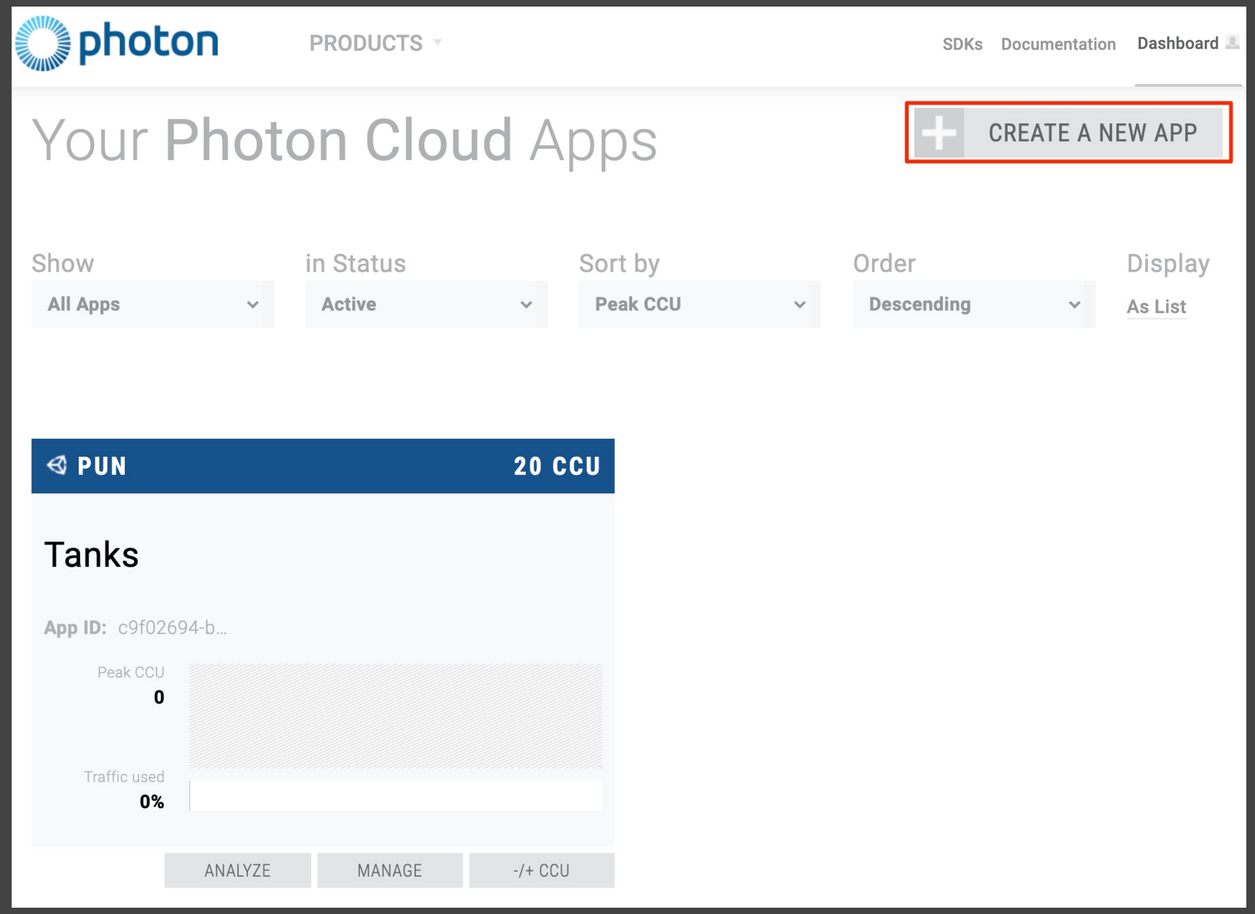
https://doc.photonengine.com/en-us/fusion/current/getting-started/fusion-intro

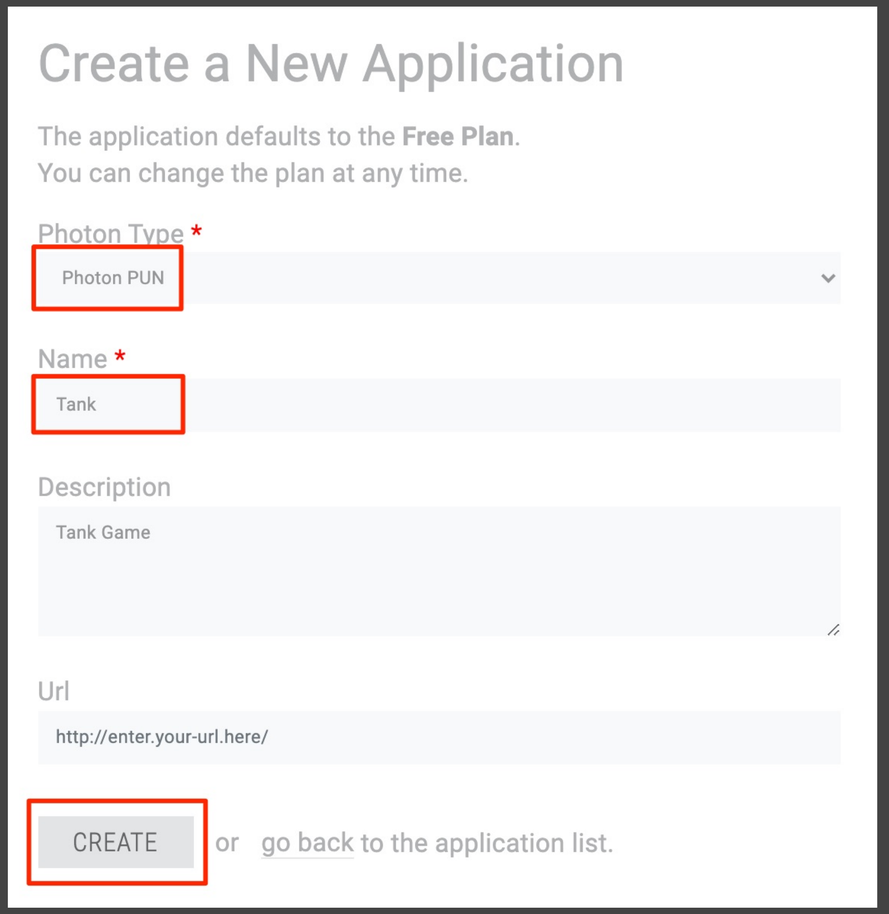
Photon Unity Network



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

Create PUN App

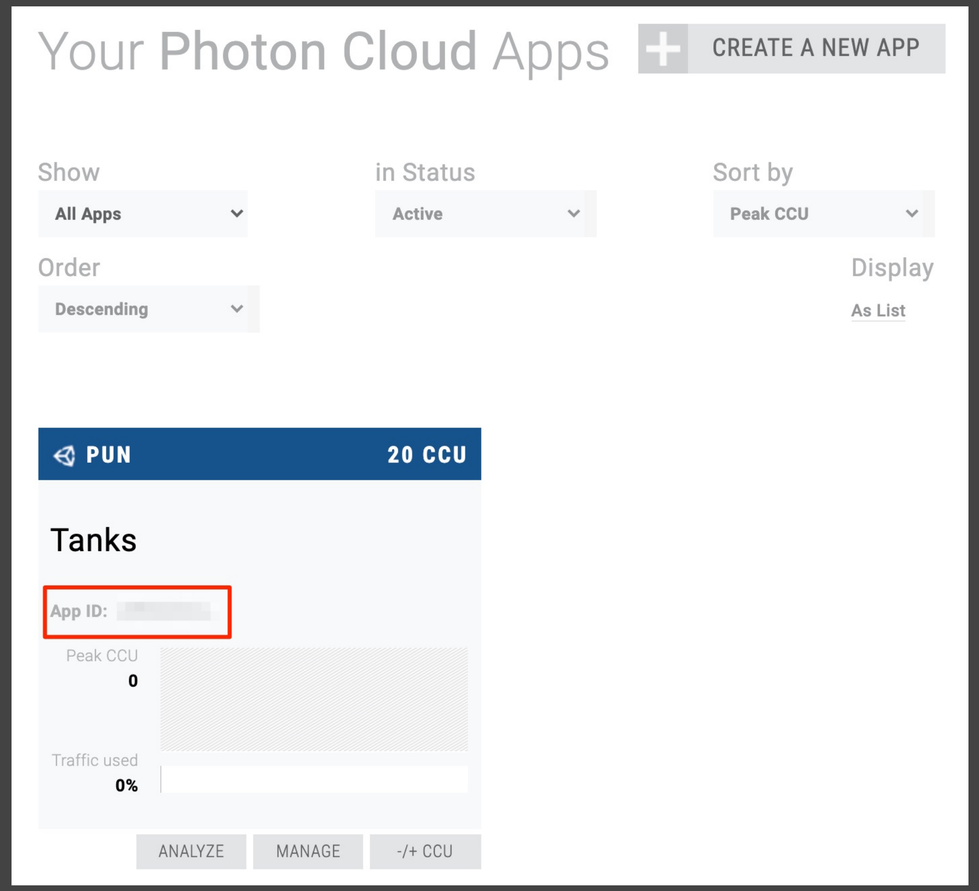




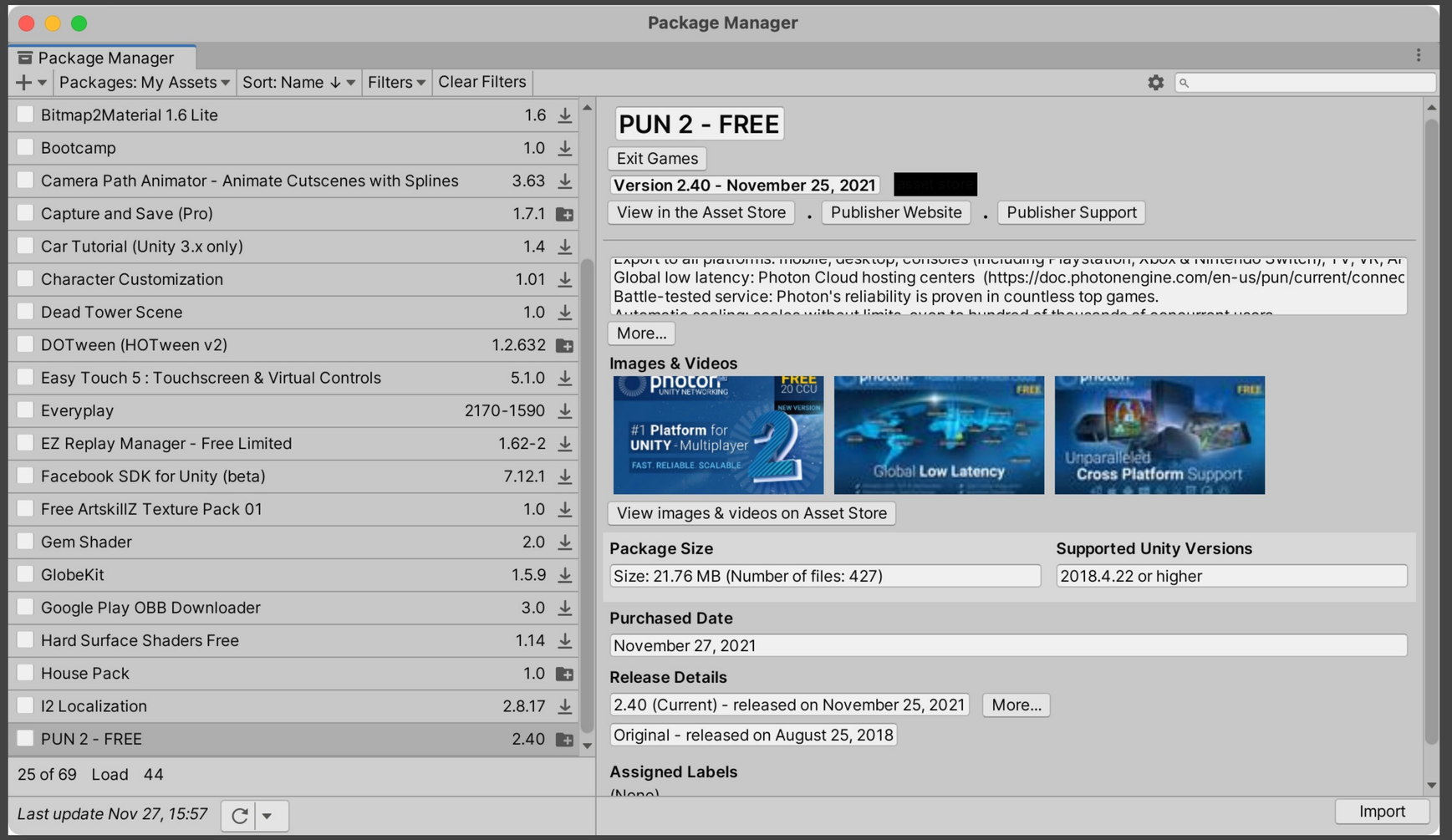
https://www.photonengine.com/zh-TW/PUN

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

Create PUN App

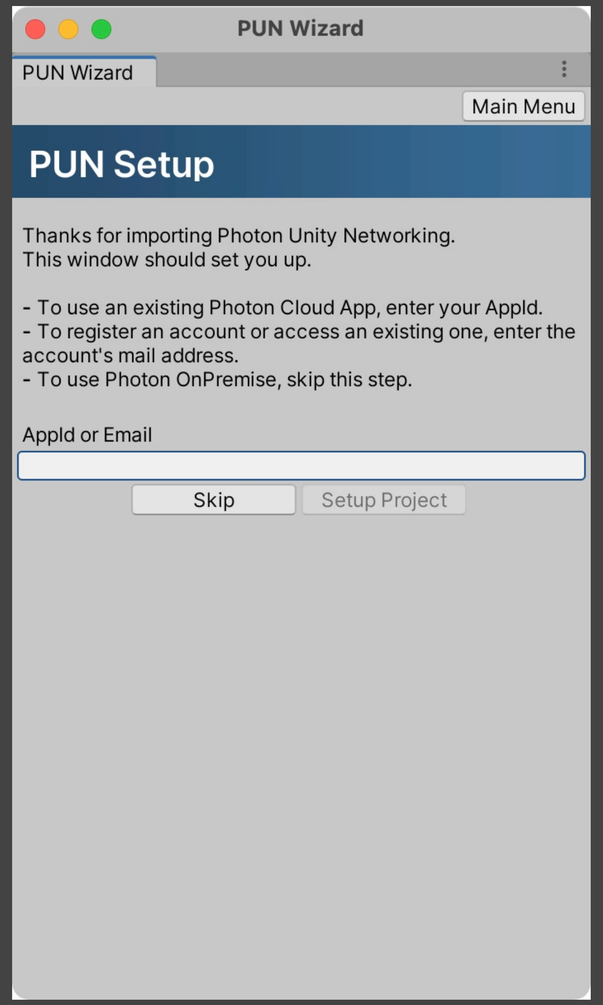


Import PUN in Unity



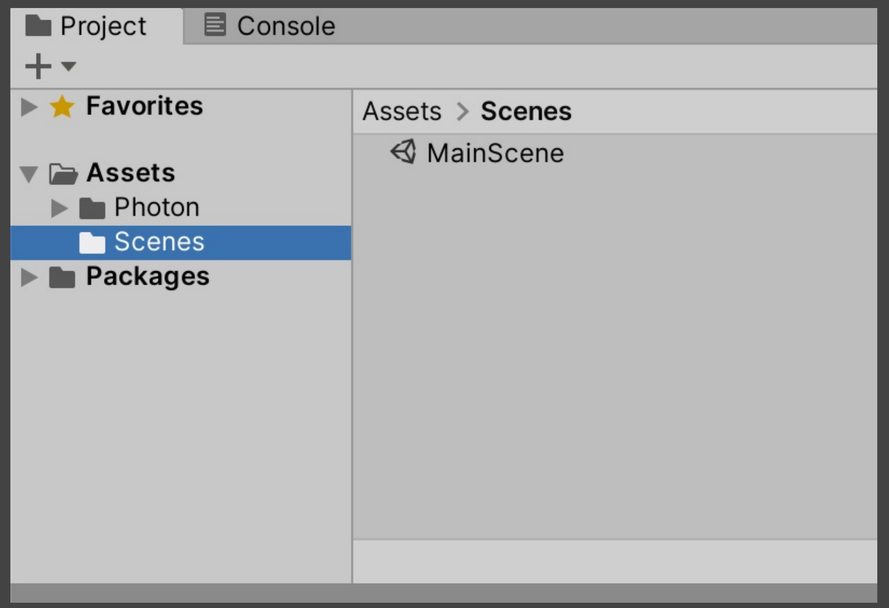
PUN Setting

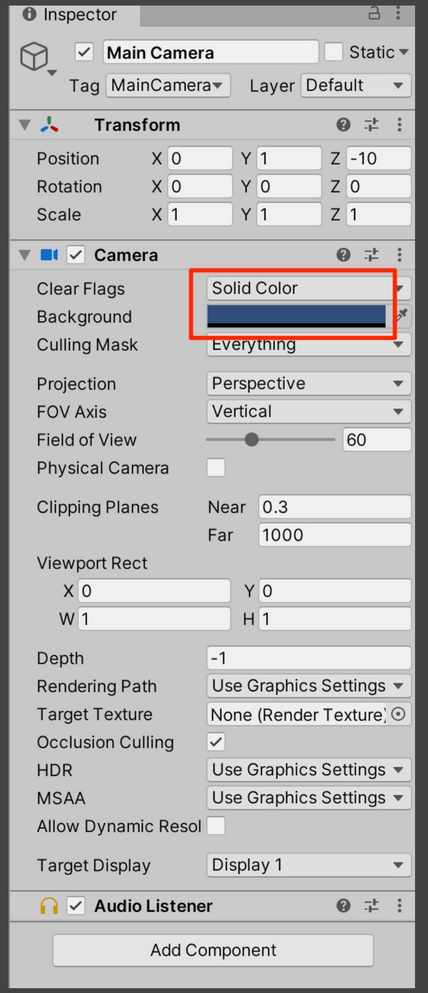
- 開啟 *Window* > *Photon Unity Networking* > *PUN Wizard* 視窗
- Select Setup Project
- Fill your AppId from your photon Dashboard



Setup Main Scene

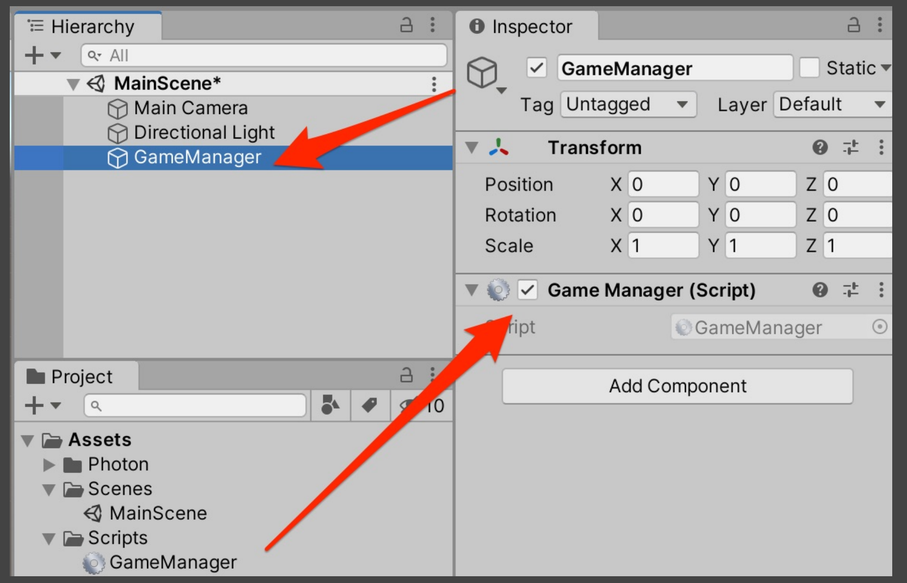
- 儲存目前的 Scene，並取名 MainScene，並放在 Scenes 目錄下
- Camera Clear Flags 選 Solid Color





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 behavior上
- GameManager會用來”管理整個場景“跟”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 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
  - Defines if all clients in a room should automatically load the same level as the Master Client.
  - When this is true, the MasterClient can call PhotonNetwork.LoadLevel() and all connected players will automatically load that same level.

MonoBehaviourPunCallbacks

- 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

Connect to Photon Cloud

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

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

練習

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

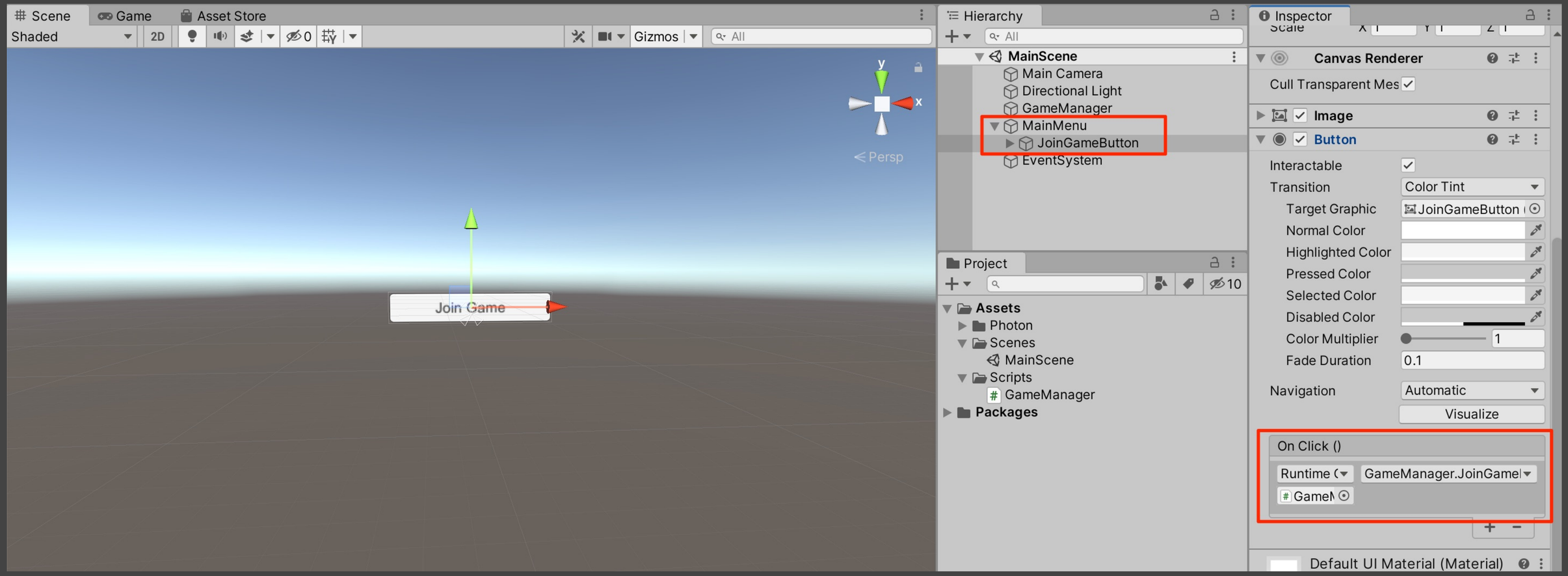
Create & Join Room

- 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



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

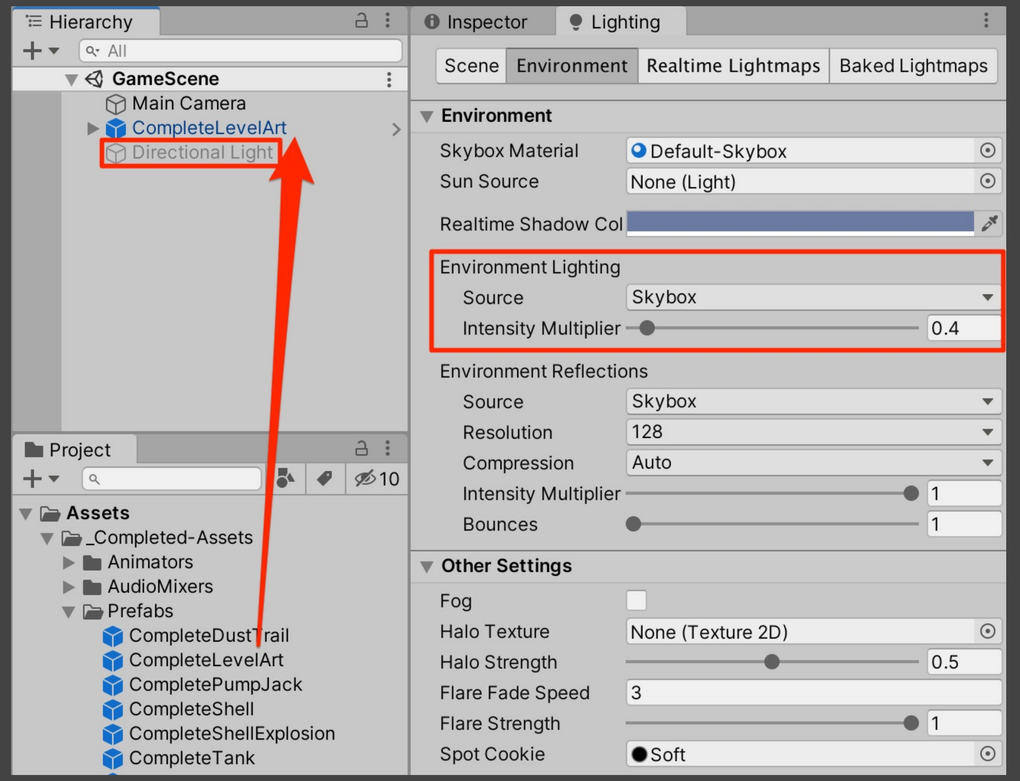
Create Game Scene

- Unity 官方教學的Tanks!
  - https://unity3d.com/learn/tutorials/projects/tanks-tutorial

- 下載 & Import Tanks Assets
  - https://drive.google.com/file/d/1bim-9nM9YRor4UDiiB5YYE3TkuVarJ4q/view?usp=sharing

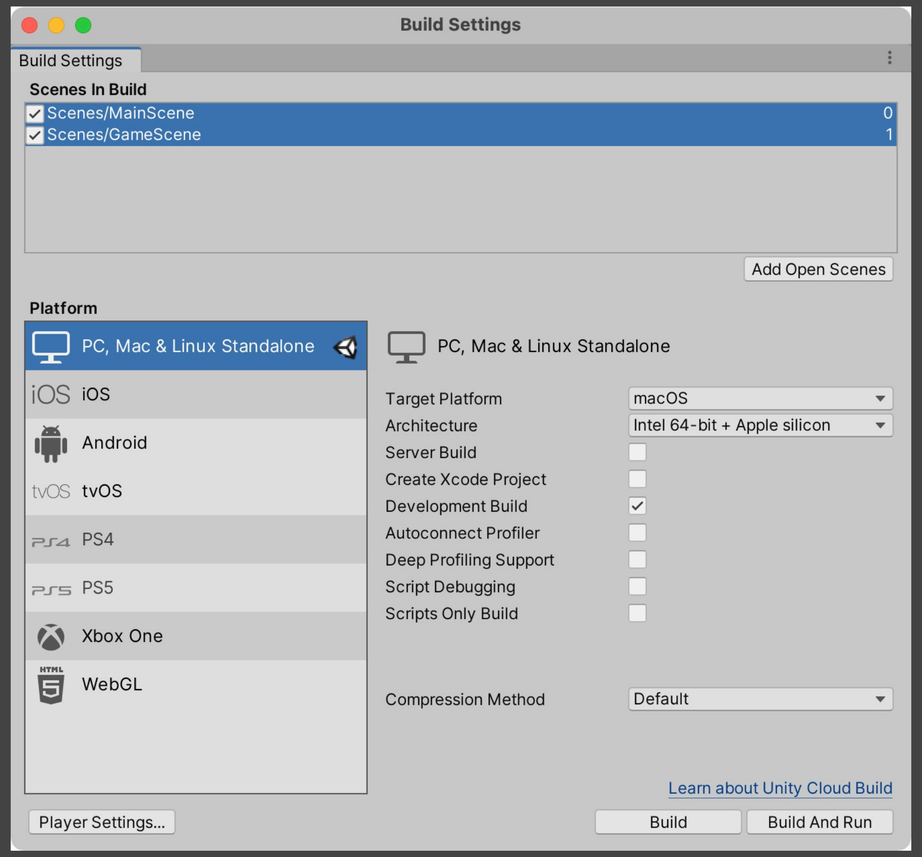
Create Game Scene

- 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)



Create Game Scene

- 開啟 File > Build Settings 視窗，將兩個Scene加到Build中



載入 Game Scene

- 自動同步場景機制

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

- 載入 Game Scene

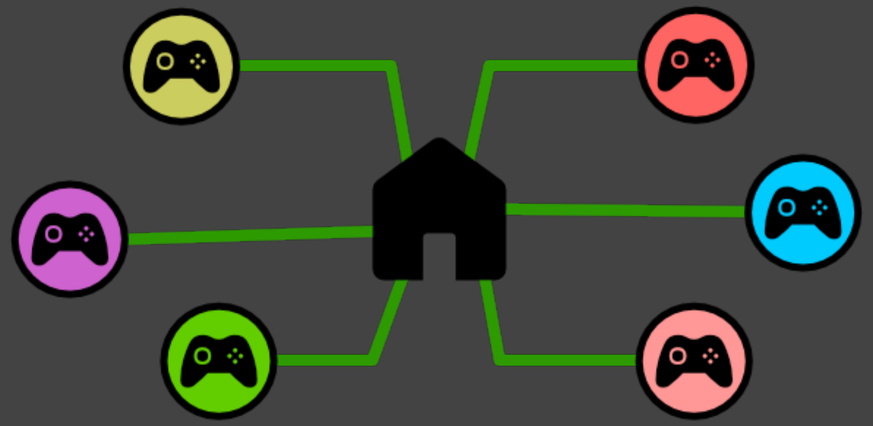
```
public override void OnJoinedRoom()
{
  if (PhotonNetwork.IsMasterClient)
  {
    Debug.Log("Created room!!");
    PhotonNetwork.LoadLevel("GameScene");
  }
  else
  {
    Debug.Log("Joined room!!");
  }
}
```

Master Client

- 在 P2P 遊戲中，通常其中一個客戶端(Client)會作為 Game Logic Server
- 通常是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

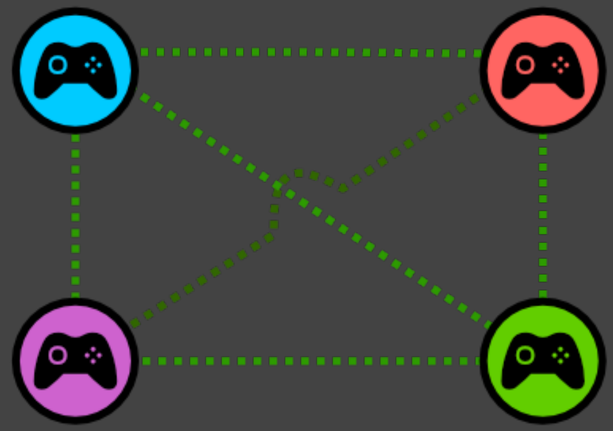
Client-Server 架構

- 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



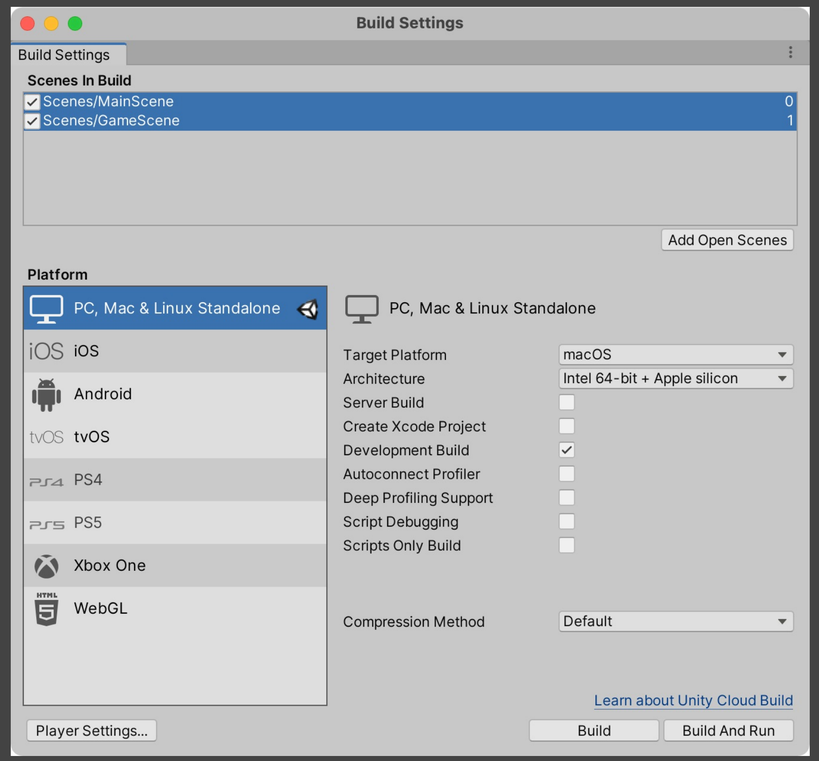
Peer-To-Peer (P2P)

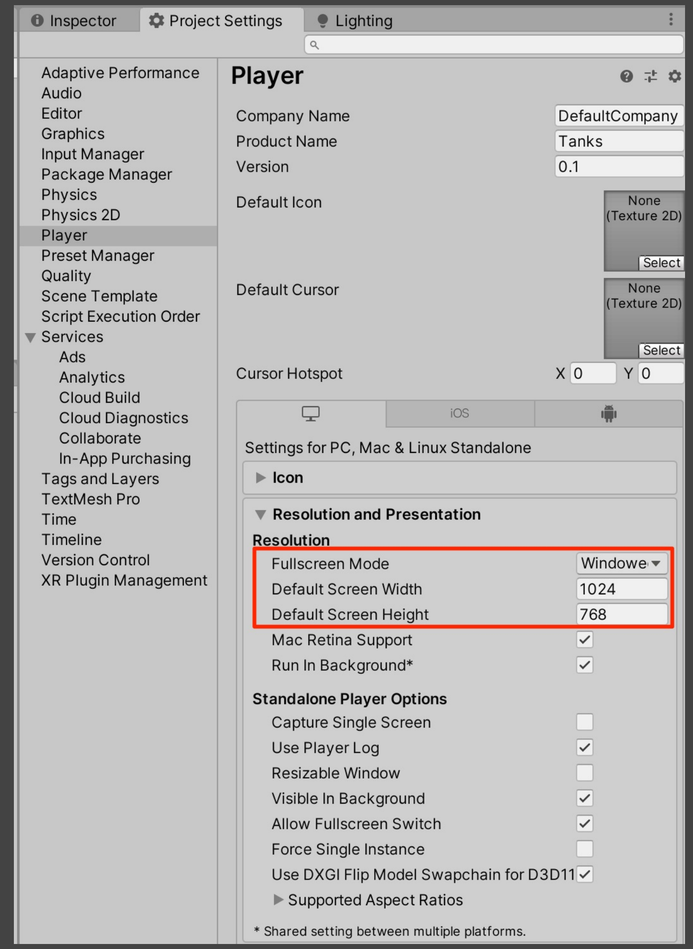
- 點對點模式
- 不需要獨立主機做訊息的派送
- 每個Client (Peer, Player) 會負責傳送自己資料給所有 Client
- 比較常見是區域網路遊戲(LAN Game, Local Network Game)，大概 4~8 人的遊戲



Play!

- 開啟 Edit > Project Settings，設定為視窗模式
- 執行2個遊戲，觀察 Created/Joined Room 訊息
  - Build And Run
  - Unity Play





練習

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;
  …
}
```

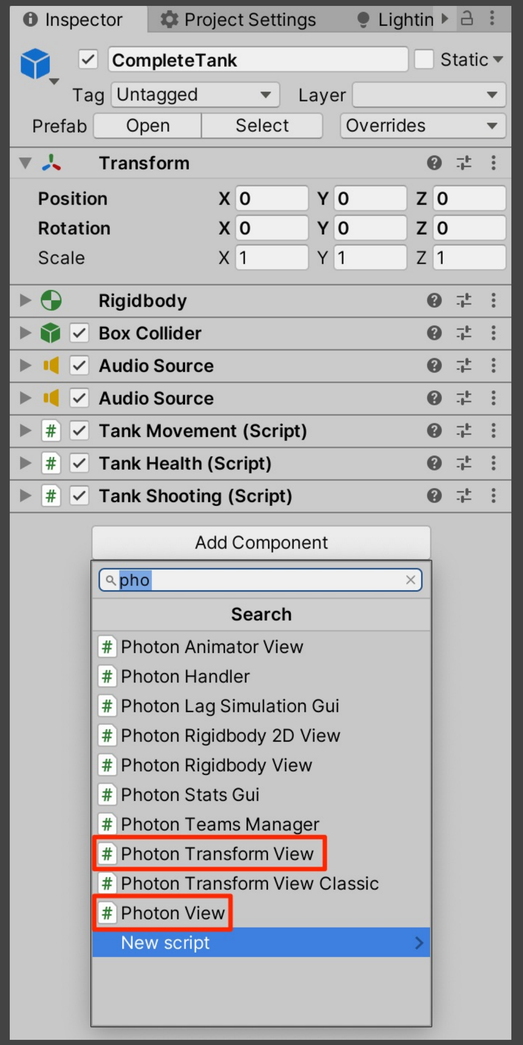
Create Player

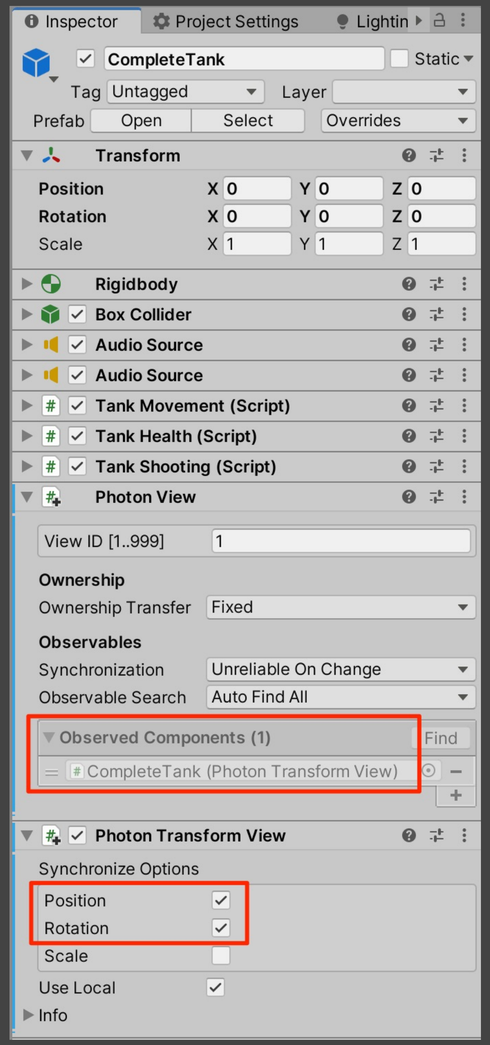
- 因為 sceneLoaded 在 MainScene 被載入時也會被呼叫，所以這邊使用 PhotonNetwork.InRoom 來檢查現在是否已進入遊戲，然後使用 TankPlayer prefab 來產生玩家的坦克

```
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviourPunCallbacks
{
  ...
  void Start()
  {
    SceneManager.sceneLoaded += OnSceneLoaded;
    ...
  }
  ...
  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());
  }
}
```

Create Player

- 設定 Player prefab
  - 拖移 _Completed-Assets\Prefabs\CompleteTank.prefab 到場景
  - 加入 PhotonView 來同步及自動建立其他玩家的坦克
  - 加入PhotonTransformView 行為來自動坦克的位置及方向



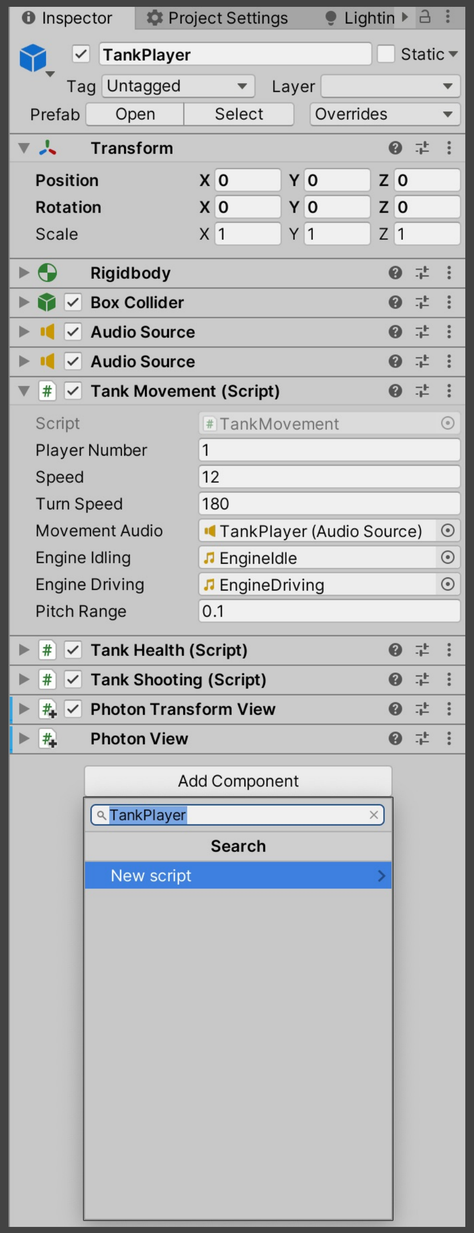


Create Player

- 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：不使用。
  - Unreliable：
    - 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：
    - 會檢查每一次更新的變化。如果數值與上一次一樣，就會以可靠的方式傳送這次更新，然後就會停止傳送後續的更新，直到數值再次發生變化。
    - 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
  - Reliable Delta Compressed：
    - 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。
    - 只有傳送與上次數值的差值。

Create Player

- 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



Create Player

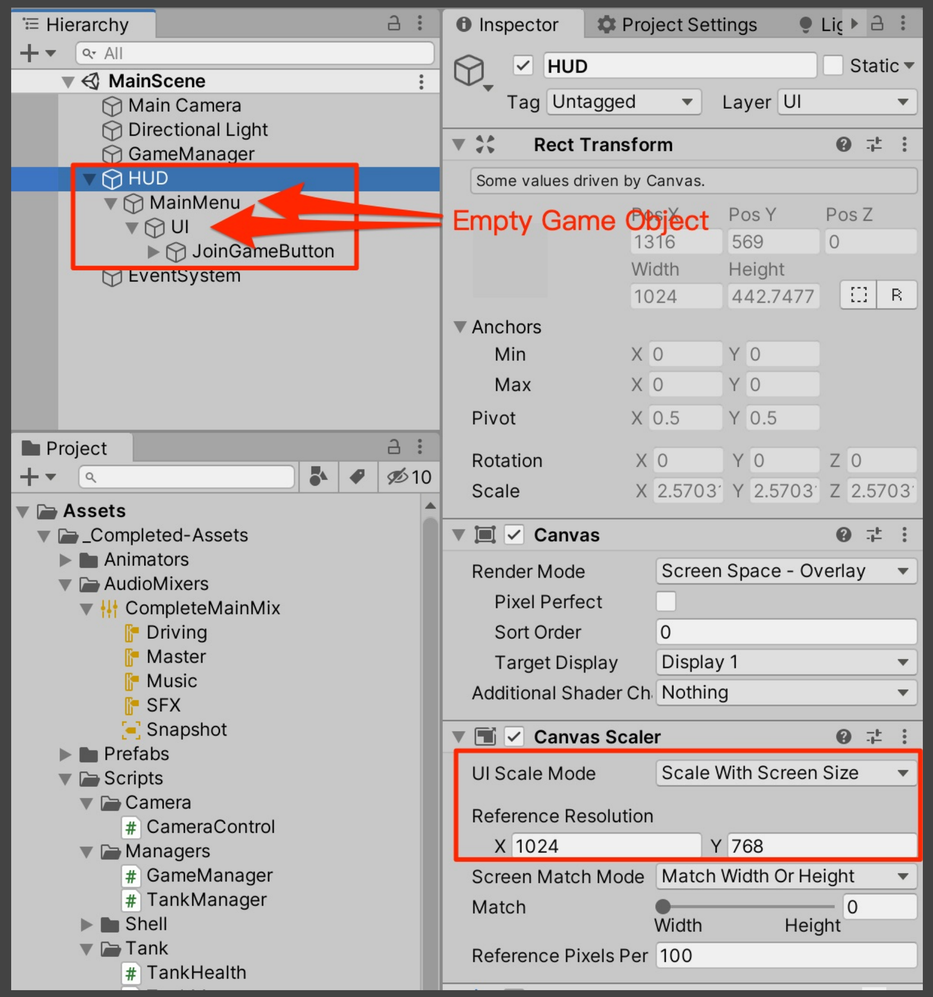
```
using Photon.Pun;

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

Main Menu


Hierarchy
All
MainScene
Main Camera
Directional Light
GameManager
HUD
MainMenu
UI
JoinGameButton
EventSystem
Empty Game Object
Project
Assets
_Completed-Assets
Animators
AudioMixers
CompleteMainMix
Driving
Master
Music
SFX
Snapshot
Prefabs
Scripts
Camera
CameraControl
Managers
GameManager
TankManager
Shell
Tank
TankHealth
Inspector
Project Settings
HUD
Static
Tag Untagged
Layer UI
Rect Transform
Some values driven by Canvas.
Pos X 1316
Pos Y 569
Pos Z 0
Width 1024
Height 442.7477
Anchors
Min X 0 Y 0
Max X 0 Y 0
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 2.5703 Y 2.5703 Z 2.5703
Canvas
Render Mode Screen Space - Overlay
Pixel Perfect
Sort Order 0
Target Display Display 1
Additional Shader Ch Nothing
Canvas Scaler
UI Scale Mode Scale With Screen Size
Reference Resolution
X 1024 Y 768
Screen Match Mode Match Width Or Height
Match 0
Width
Height
Reference Pixels Per 100

Main Menu

- 接下來在 Scripts 目錄下新增一個 ExtensionMethods Script，用來擴展 Transform 類別，讓它可以根據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

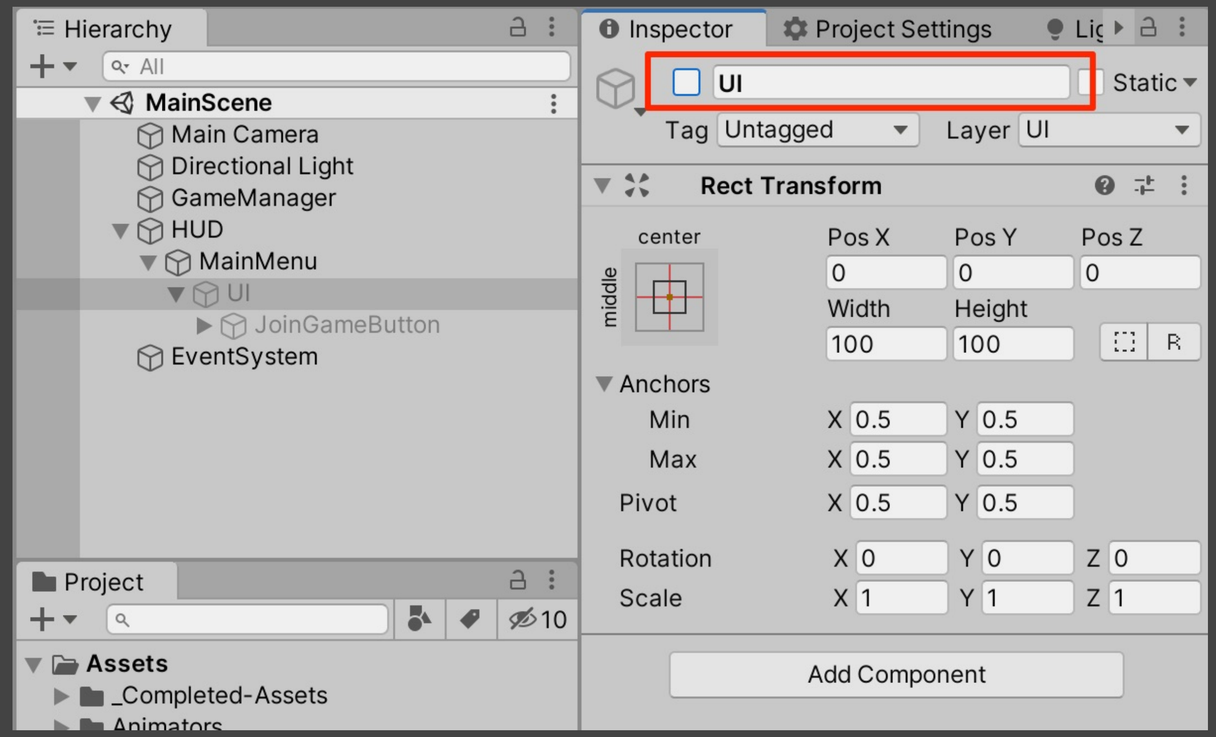
- 在 Scripts 目錄下新增 MainMenu Script，並且設定為 Singleton 型式，以及繼承 MonoBehaviourPunCallbacks，然後將此 Script 放到 MainMenu 物件上

```
using Photon.Pun;

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

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

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

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

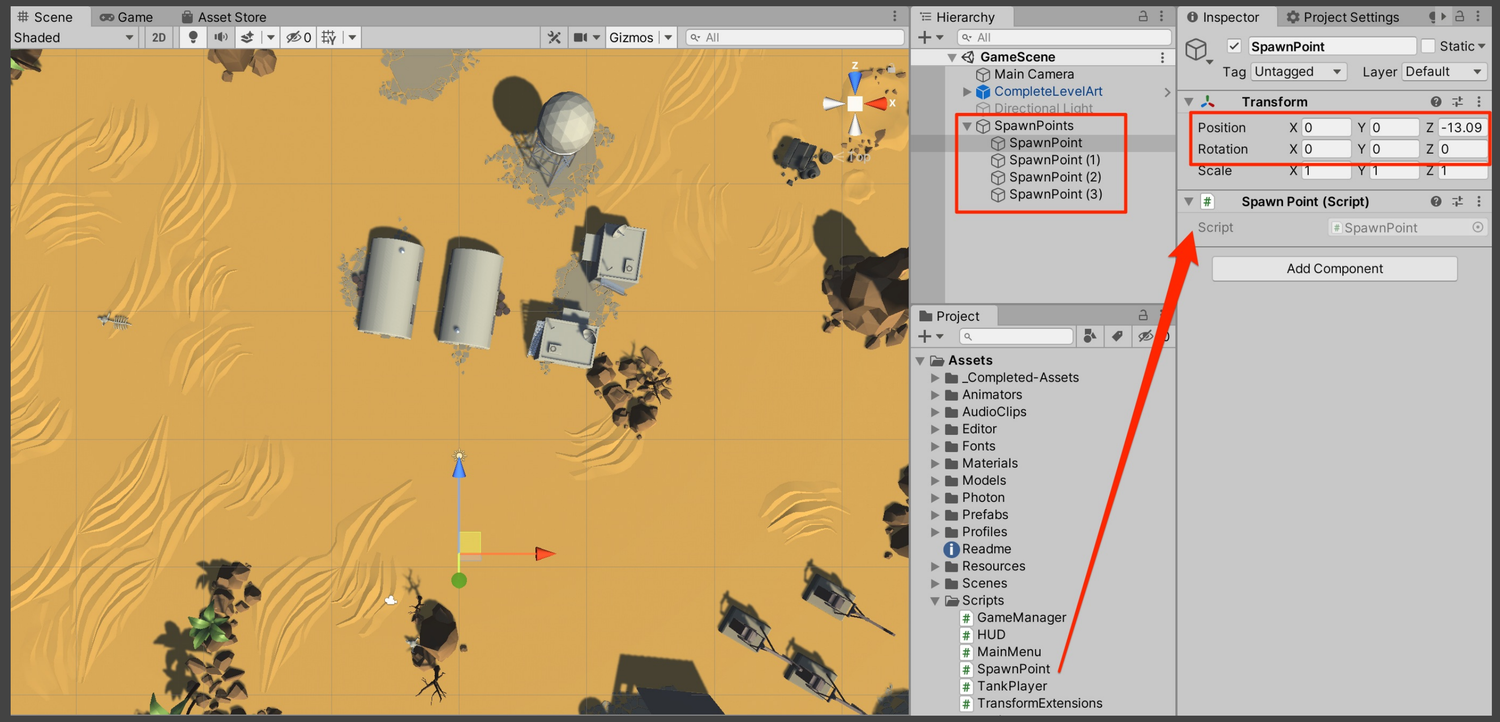
Spawn Points

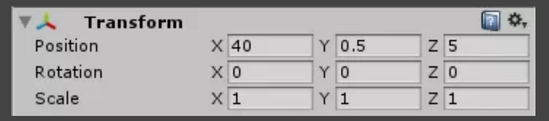
- 在 Scripts 目錄下新增 SpawnPoint Script

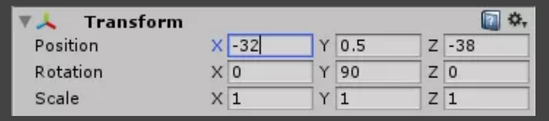
```
using UnityEngine;

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

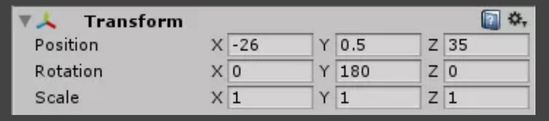
Spawn Points

- 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

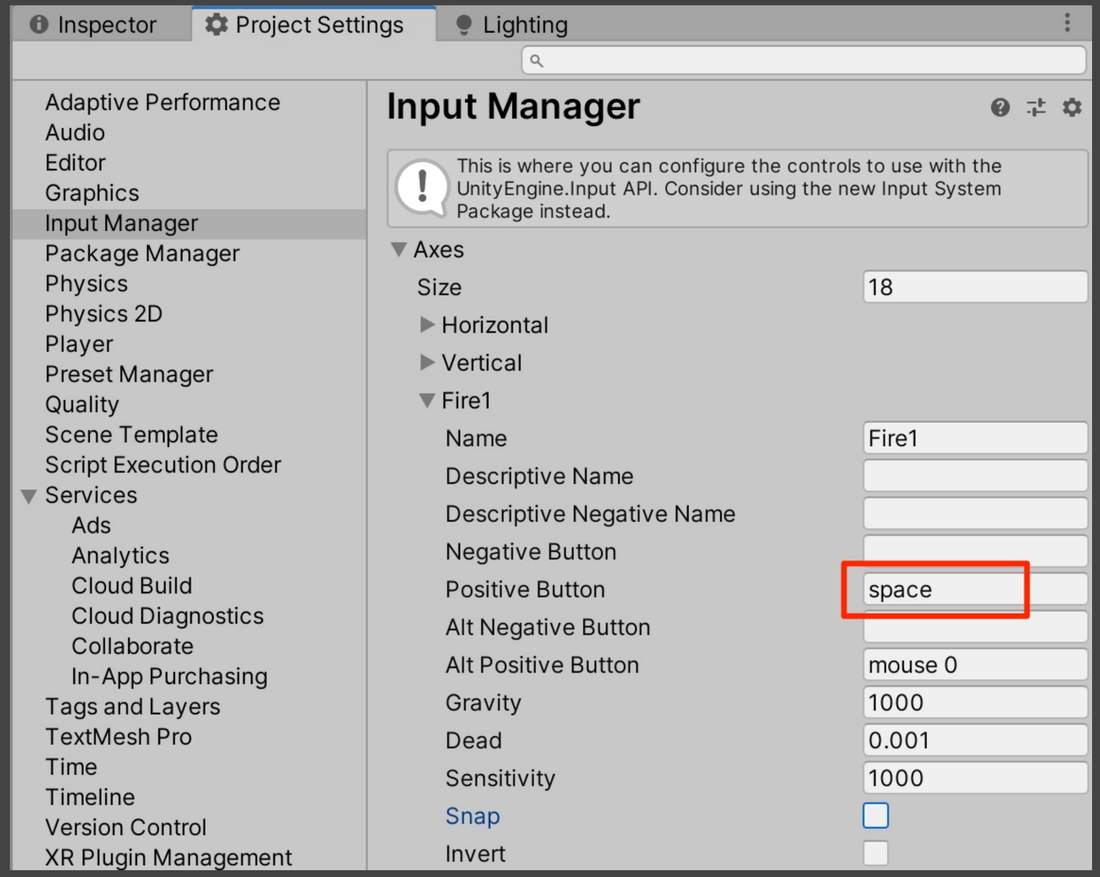
- 修改 OnSceneLoaded 函式，讓坦克可以在隨機位置出現

```
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

Shoot

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



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

Shoot Method 2
PUN Remote Procedure Call

- 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

```
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

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)