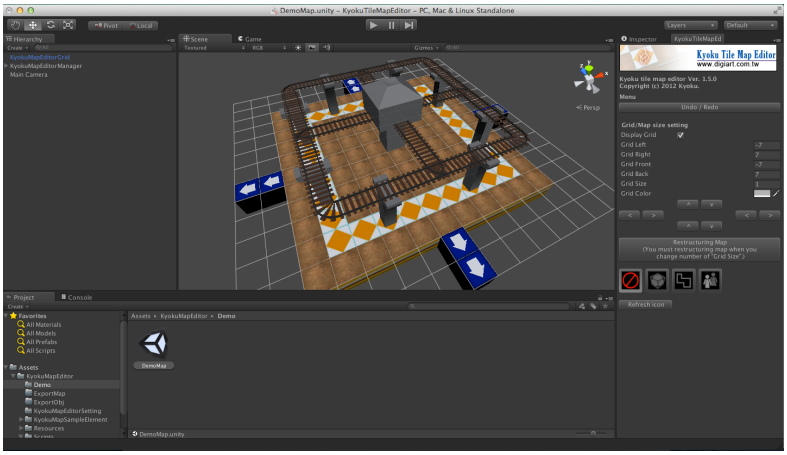


Unity 外掛開發

本課程將說明如何開發 Unity 的外掛工具，課程中會學到如何建立視窗選項，如何利利用工具產生物件，外掛是如何在場景內繪製物件的，並會以一個大型的 Kyoku 方塊地圖編輯器來說明如何架構一個大型工具，以及開發時會遇到的問題以及建議，課程的最後會介紹如何上傳自己的工具或模型到 Assets Store 上。

紀曲峰

目錄

概要

Unity 擴充功能入門與安裝

1. 什麼是 Unity 外掛，為何需要開發擴充程式

2. 透過 Kyoku 地圖編輯器了解 Unity 外掛的安裝與佈屬方式

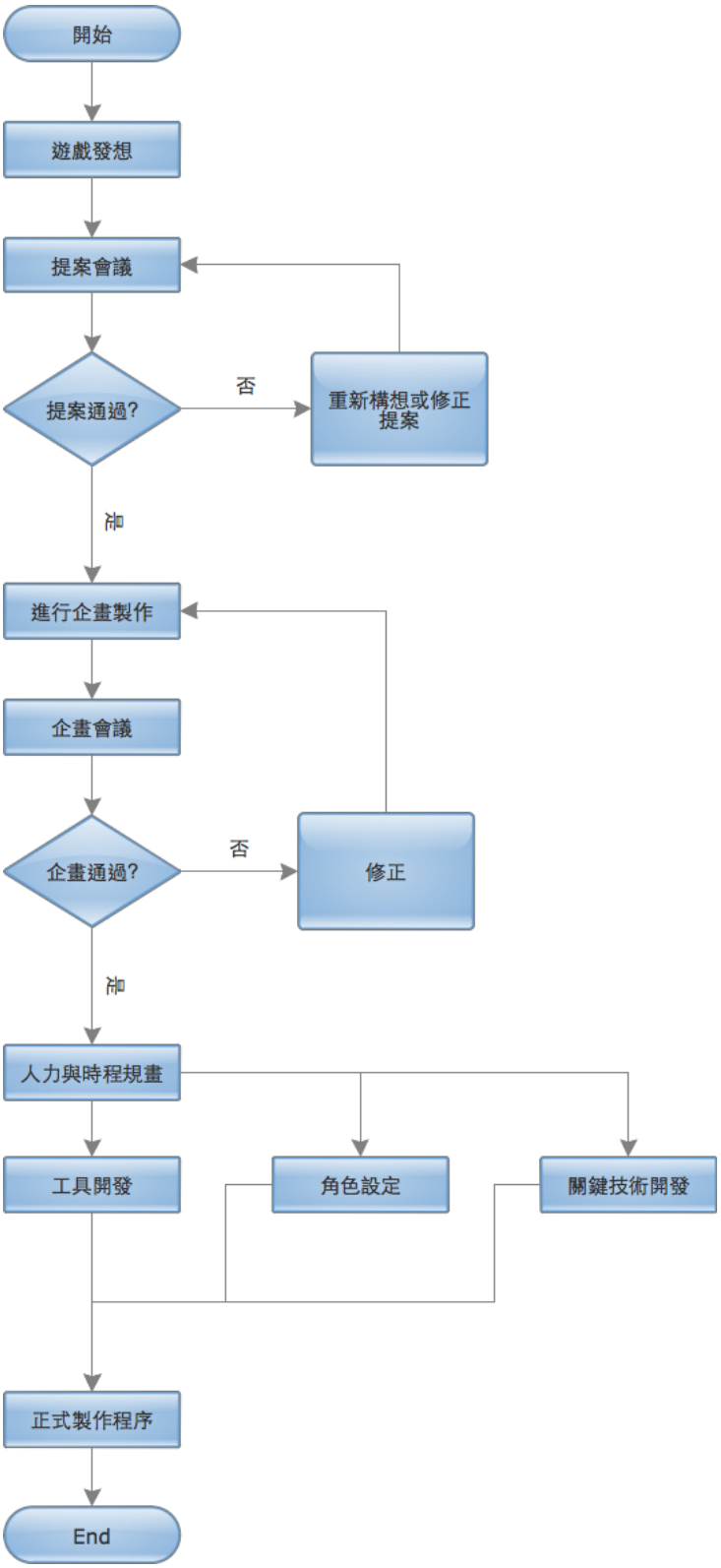
3. 透過 Kyoku 地圖編輯器了解 Unity 擴充程式的特性與可控制項有哪些

開發遊戲的流程與進度的掌控

一個遊戲的開發通常都是數人或數十人以一個複雜的 teamwork 方式通力合作才能完成的，一般來說要開發一個遊戲，必須先有基本的發想，要做出什麼樣的遊戲，遊戲的大綱和主要的訴求為何，完成提案工作後，接著必須有一個或一組的企畫人員對遊戲進行企畫的工作，大部份遊戲發想也是由企畫人員提出再由公司進行評估，當企畫每完成到一個階段就要進行企畫會議，會議中必須針對企畫所提出的人力及時間成本進行評估及估算。

企畫定案之後便要開始投入美術及程式的人力，這時開始要分配出美術、程式、企畫人員的一些細部工作，此時也會一併考慮到工具開發的需求，若是遊戲需要仰賴大量的關卡編輯，此時便需要配置遊戲關卡設計人員，為關卡設計人員開發出編輯工具則是程式設計師的任務。

當進入製作階段之後，進度的掌控變得非常重要，不但要借助一些專案軟體掌控所有的進度，如何分散作業與找出製作瓶頸便是一個重點，分擔程式設計師的作業通常的做法便是

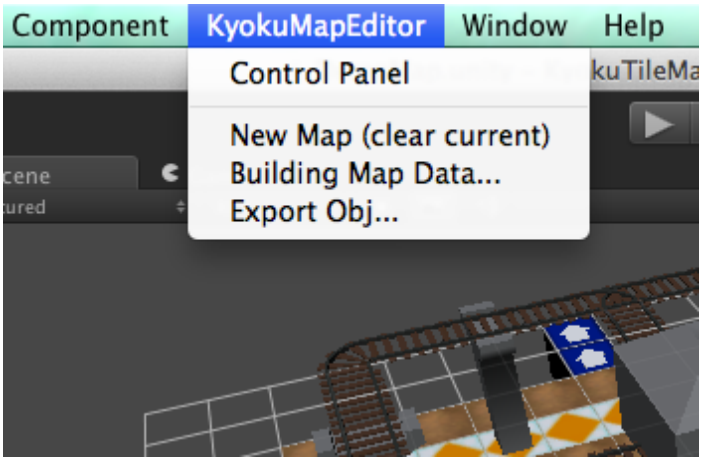


開發出好用的工具和編輯器，以免所有的美術元件和關卡內容都集中到程式設計師身上，到時便會發生製作上的瓶頸，而在專案的初期當美術和關卡和企畫正在進行之時，程式工程式等待資源時便是開發工具最佳的時機點，這時若能掌握時間開發好必要的工具便能讓後面的進度得以事半工倍。

當然，並不是所有的遊戲開發都需要另外開發編輯器或工具，後面還會對此做一些研究探討。

什麼是 Unity 的外掛

簡單說，就是安裝完 Unity 之後，額外安裝的 Unity 功能，如圖所示，KyokuMapEditor 不屬於 Unity 安裝完後內建的功能，是額外安裝的功能，這就是外掛。



是否需要開發工具

並非所有的遊戲製作都需要開發工具，在開發工具前必須探討幾件事：

1. 開發工具是否有助於降低成本
2. 工具是否能簡化製作程序
3. 工具是否限制了遊戲內容的創造力

有些簡單的遊戲，開發工具反而讓成本(所謂的成本不光指金錢，還包含了人力及時間都該算成成本)增加，或是當經驗不足，在研發工具花的時間遠大於製作所花費的時間，這

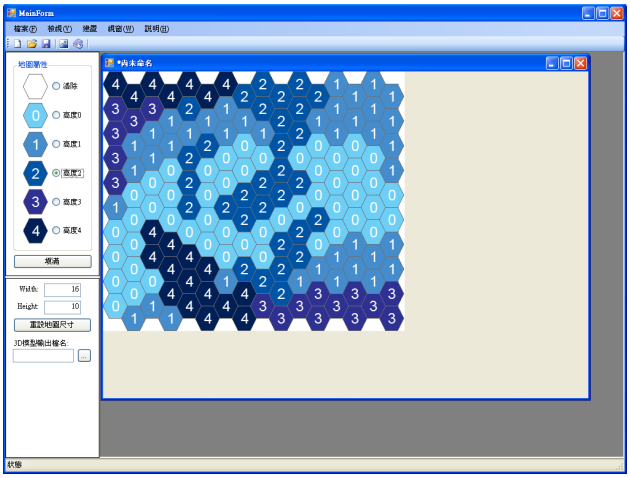
時還不如跳過工具開發階段直接製作，千萬不要為了滿足成就感為了開發而開發，必須是為了作品而開發。

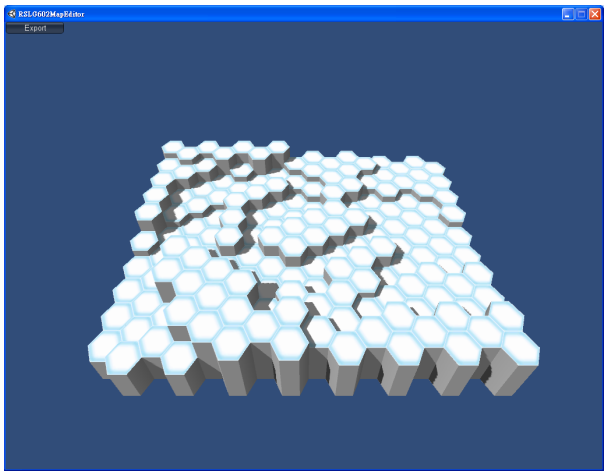
另外，不好用的工具反而增加遊戲製作上的困難，這也是要避免的。

獨立運行的工具與掛載於 Unity 底下的工具

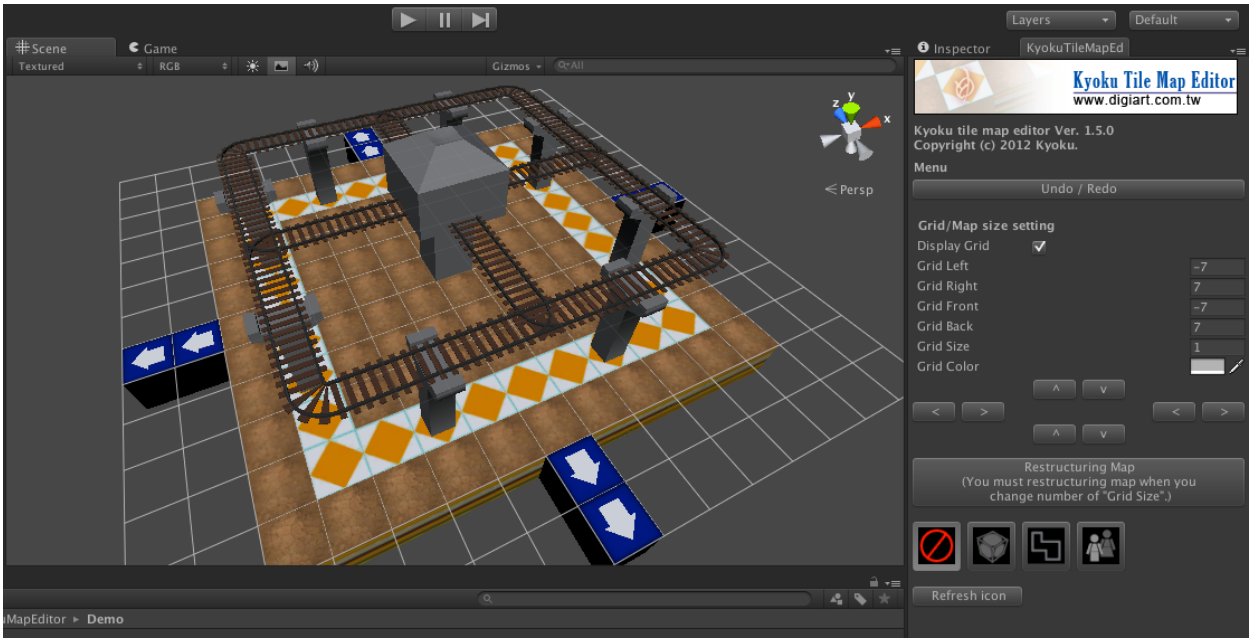
並不是所有的工具都需要掛載於 Unity 底下，有時開發的工具可以是獨立運行的，工具要掛載於 Unity 底下其實在開發上有一定的門檻，有時開發成獨立運行的工具其實也不失為一個好方法。

獨立運行的工具



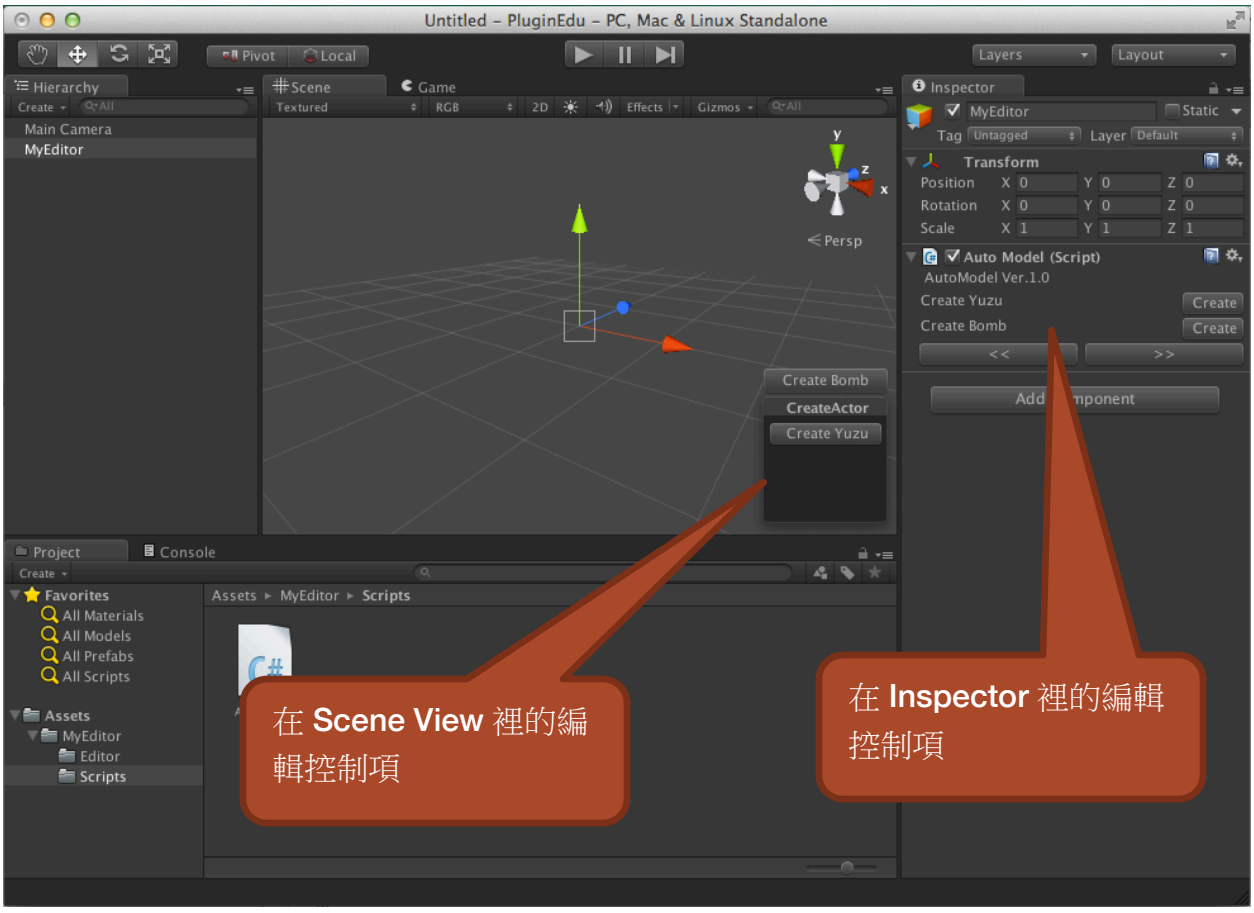


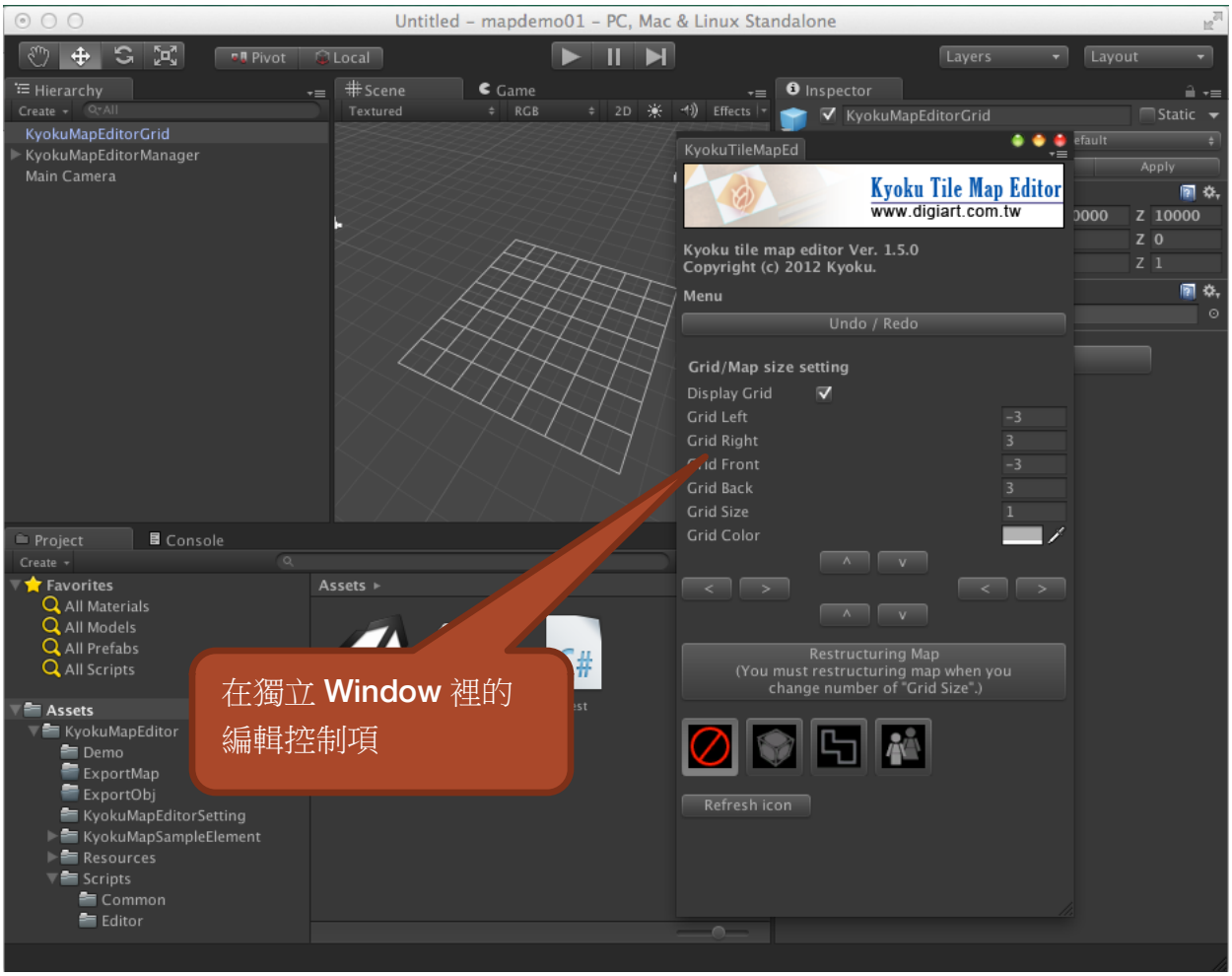
掛載於 Unity 底下運行的工具



外掛編輯器之控制項

一般來說使用外掛編輯器，可以進行編輯的地方會有 Inspector, Window, SceneView 三個區域，這三個區域 Unity 的 Script 都有提供可以控制的對應方法。





利用 Editor 類別客製 Inspector 內容

雖然直接利用 Editor 在 Inspector 建立控制項的做法比較少用，但偶爾會遇到某些元件希望製作控項但又不希望這些控項被帶到遊戲當中，這時便可以採用這種方法，我們可以順便了解一下編輯器的基本寫法。

要建立編輯工具的程式碼必須繼承自 Editor 或 EditorWindow，當類別繼承自這兩個類別時，程式碼一定要放在 Editor 這個資料夾內，否則會產生錯誤，放在此資料夾有兩個特性。

Editor 資料夾的特性

程式碼會在最後的順序才被執行

Unity 在執行程式碼是有順序的，會依資料夾的名稱而有執行的先後順序。

1. Unity 會優先執行「Standard Assets」和「Pro Standard Assets」和「Plugins」這三個資料夾的程式碼，含底下的子資料夾。
2. 接下來 Unity 會執行「Standard Assets/Editor」和「Pro Standard Assets/Editor」和「Plugins/Editor」這三個資料夾裡的程式碼。
3. 接下來會執行「Editor」資料夾以外的程式碼。
4. 最後才執行「Editor」資料夾底下的程式碼。

這些順序表面上看起來影響不大，但通常在執行編輯器時都希望遊戲裡的元件特性都已經被賦予完成了，也就是希望那些遊戲裡的 prefab 裡的 code 都已經被初始化完成，這時我們的編輯器工具才能取得正確的角色屬性，因此 Editor 通常都需要最後才被執行。

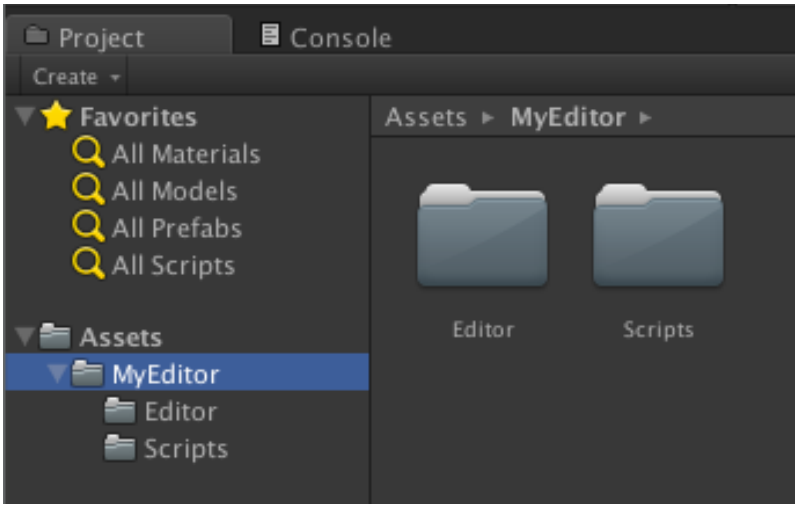
在編譯階段不會被編譯出去

我們編輯器裡的資訊和程式碼都是在編輯階段所需要的，遊戲最後要編譯出去時是不需要編譯這些編輯器的程式碼的，一旦程式碼放在 Editor 資料夾底下便能確保這些 code 不會在最後階段被編到遊戲當中，以免造成遊戲內容的效能低落。

建立專案

現在請先建立一個新專案來開發我們的第一個工具，專案名稱請自行取名，編輯器最後都會包成資源包出去讓多個專案使用，因此專案名稱並不重要。

專案完成後接下來建立資料夾，這裡建立一個名為「MyEditor」的資料夾。接下來開啟此資料夾後再建立兩個資料夾，分別為「Scripts」和「Editor」，如下圖。



Editor 的框架

繼承自 Editor 的類別無法單獨存在，他必須參考自「MonoBehaviour」的類別，因此我們必須先建立一個繼承自 MonoBehaviour 的類別物件，先到「Scripts」資料夾內建立一個新的 C#腳本檔，這裡取名為「AutoModel.cs」，這個類別內不做任何的更改。

再到「Editor 資料夾底下」建立一個 C#的腳本，取名為「AutoModelEditor.cs」，開啟這個腳本檔我們要編輯其內容，開啟後先刪除 class 底下原本的方法，接著加入以下的程式碼。

```
using UnityEngine;
using System.Collections;
using UnityEditor;


[CustomEditor (typeof(AutoModel))]
public class AutoModelEditor : Editor {

    public override void OnInspectorGUI()
    {

    }

}
```

要使用 Editor 類別必須先加入引用「using UnityEditor;」，接下來必須宣告這個 Editor 是為了編輯哪個類別，這裡是為了編輯 AutoModel 類別，因此在 class 的上方加入「[CustomEditor (typeof(AutoModel))]」，class 前方的中刮弧在 C#裡的意義是預設屬性的意思，也就是指定此類別的屬性是 CustomEditor()，必須代入要參照的類別，也就是 AutoModel。

接著移除原本 class 所繼函的「MonoBehaviour」，改成「Editor」，然後加入一個複寫的方法「public override void OnInspectorGUI()」，有了這個方法，我們便可以在裡面繪製 Inapector 的畫面了，繪製的方法和 GUI 繪製的方法雷同。

繪製 Inspector 的控制項

Inspector 的控制項繪製方式和 GUI 雷同，但又有點不一樣，主要在於 Inspector 裡面控件的排列是自動的，不是讓我們自由放置的。

先加入以下的程式碼

```
public override void OnInspectorGUI()
{
    GUILayout.BeginVertical();
    GUILayout.Label(" AutoModel Ver.1.0 ");
    GUILayout.Button("Hello");
    GUILayout.EndVertical();
}
```

程式碼

GUILayout.BeginVertical();

. . . . .

GUILayout.EndVertical();

中間的區段所有的元件都會自動的從上到下排列，所以只要利用「GUILayout.控件類別」自動繪製出由上到下的控制項。

相同的道理，若是。

```
public override void OnInspectorGUI()
{
    GUILayout.BeginHorizontal();
    GUILayout.Label(" AutoModel Ver.1.0 ");
    GUILayout.Button("Hello");
    GUILayout.EndHorizontal();
}
```

則是繪製出由左至右的控制項。

兩種可以混合製造出巢狀的控制項效果。

```
public override void OnInspectorGUI()
{
    GUILayout.BeginVertical();
    GUILayout.Label(" AutoModel Ver.1.0 ");

    GUILayout.BeginHorizontal();
    GUILayout.Label("Creat Yuzu");
    GUILayout.Button("Create", GUILayout.Width(50));
    GUILayout.EndHorizontal();

    GUILayout.EndVertical();
}
```

先在第一行繪製出一個 Label「AutoModel Ver.1.0」，接下來第二行則是左邊繪製 Label「Create Yuzu」，右邊繪製一個按鈕「Create」，並利用 GUILayout 設置其 style 為寬度 50 的按鈕。

接下來我們來讓這些控製項出現在 Inspector 當中。

在 Inspector 顯示編輯控制項

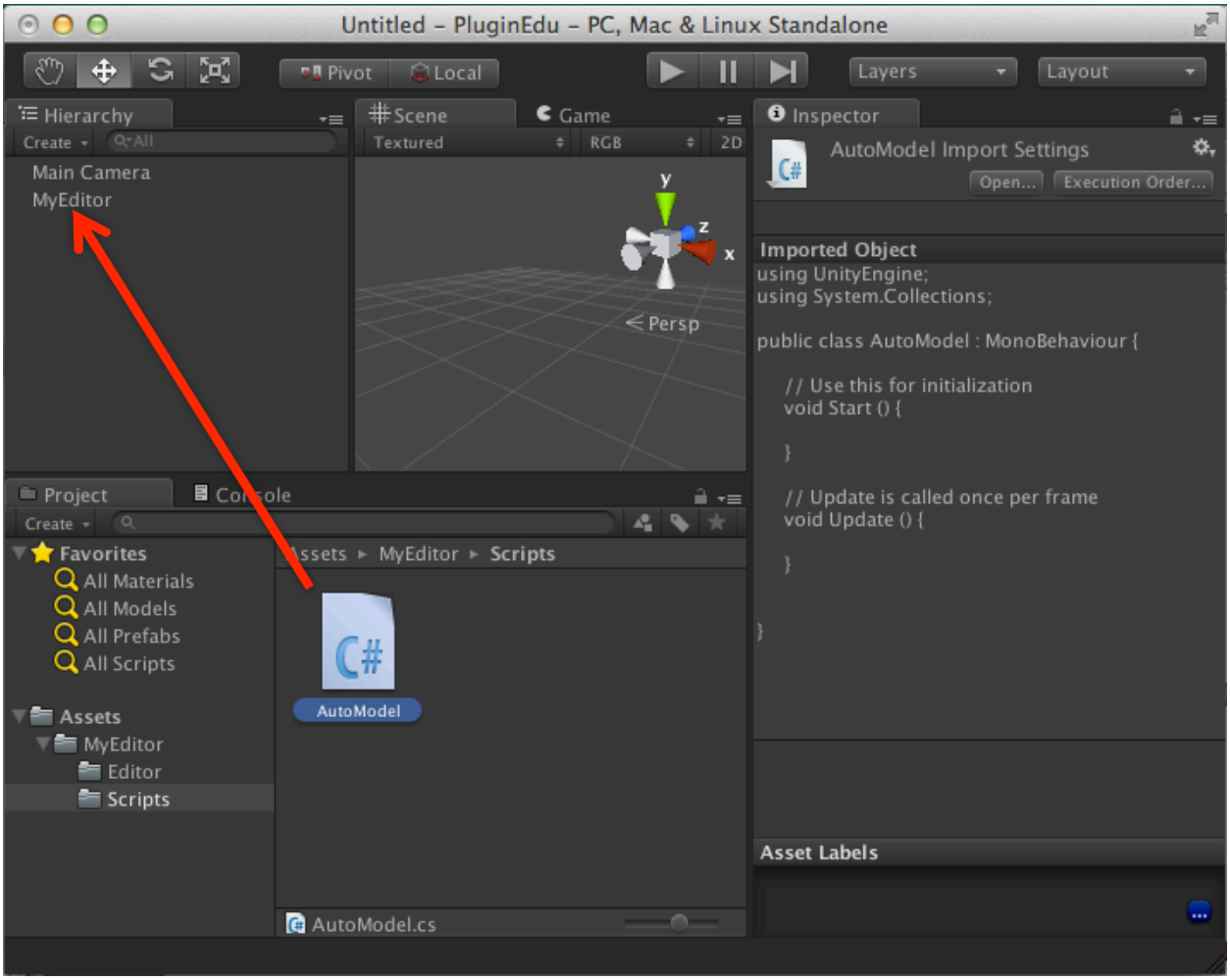
要將 Editor 的控制項顯示出來，必須在場景中建立一個物件然後將必要的 code 拉到物件上，一般來說我們不太希望是這種方式，因為一旦使用者不小心刪除掉這個物件，我們開發的編輯器便無法使用了，之後我們會改用 EditorWindow 解決此問題，但 Editor 的使用方式還是有其必要的，有時在某的遊戲角色當中我們希望加入編輯要素，但又不想要這些要素被帶到遊戲中，只希望他出現在編輯階段中，此時就可以使用 Editor 的類別。

現在先來看看如何顯示，讀者可能已經試著將「AutoModelEditor.cs」拉給場景物件了，實際上這個 code 是無法拉給場景物件的，要拉到場景物件中的只能是繼承自「MonoBehaviour」的類別，因此我們現在要拉進去的是「AutoModel.cs」這個腳本。

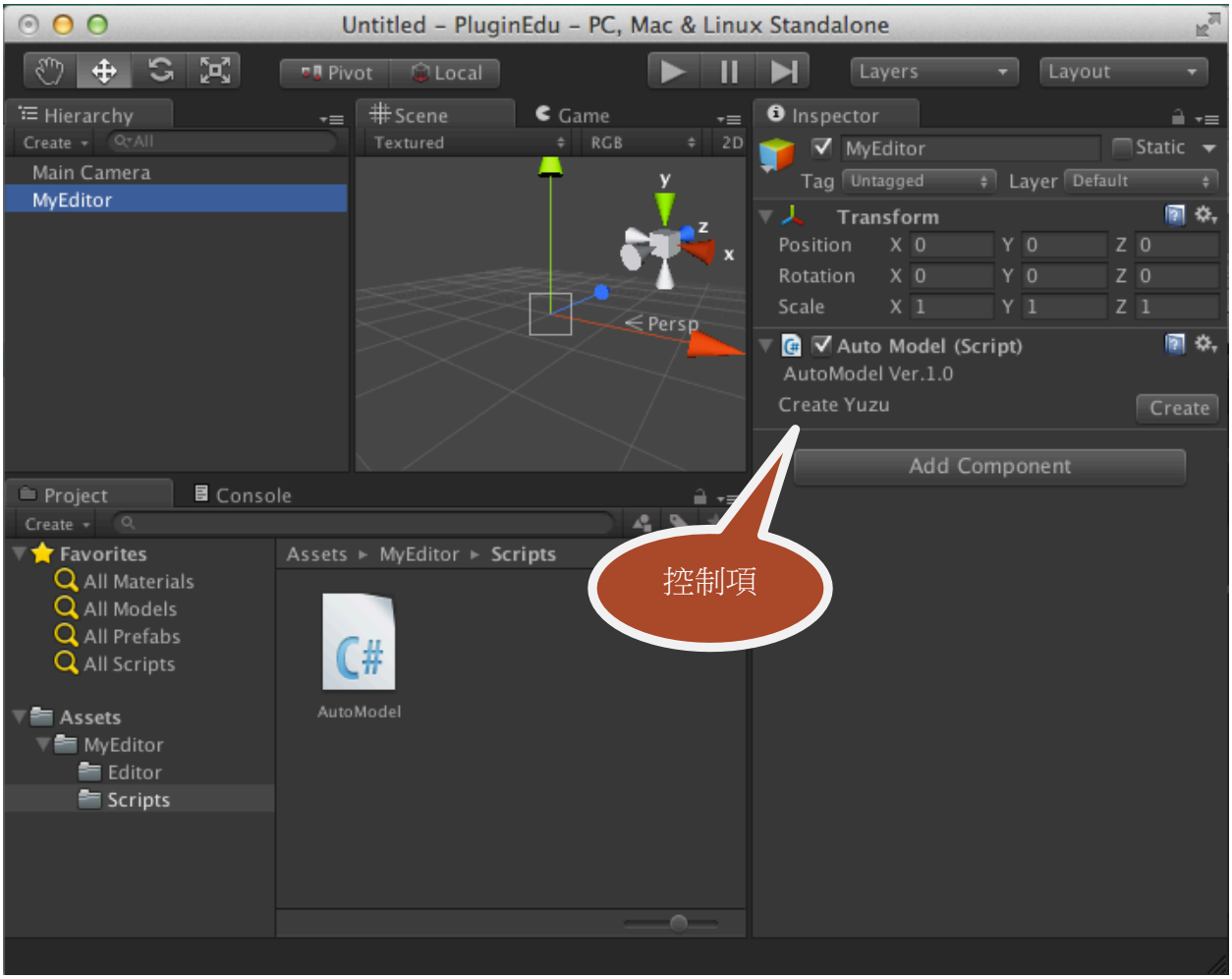
請在場景中建立一個空物件，執行以下指令。

GameObject > Create Empty

建立好之後將這個 GameObject 更名為「MyEditor」，或是自己的想要名稱也可以，接下來將「AutoModel.cs」拉到 MyEditor 上。



這時只要點選這個物件便會在 Inspector 上顯示出編輯用的控制項了。

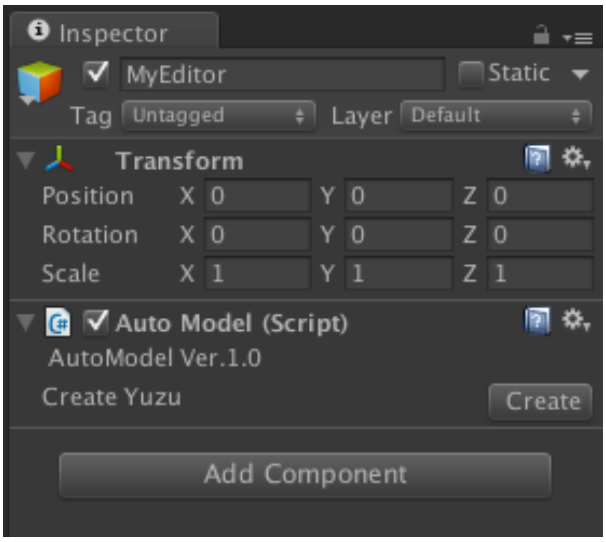


在編輯器中變更物件的參數內容

我們可以使用 Editor 去編輯物件「MyEditor」的參數，而且 Editor 主要的用途也是拿來編修場景中的物件，首先開啟「AutoModel.cs」腳本，增加一個「speed」的參數。

```
using UnityEngine;
using System.Collections;

public class AutoModel : MonoBehaviour {

    public float speed = 1.0f;

    // Use this for initialization
    void Start () {
        Debug.Log("speed = " + speed.ToString());
    }

    // Update is called once per frame
    void Update () {

    }


}
```

此時回去看 Inspector 會發現就算這個參數設為 public 但依然不會出現在 Inspector 裡面，一旦類別被 Editor 引用後就一定要透過 Editor 去控制。



回到「AutoModelEditor.cs」，修改程式碼。

```
using UnityEngine;
using System.Collections;
using UnityEditor;


[CustomEditor (typeof(AutoModel))]
public class AutoModelEditor : Editor {

    AutoModel autoModel;
    public void OnEnable()
    {
        autoModel = (AutoModel)target;
    }

    public override void OnInspectorGUI()
    {
        GUILayout.BeginVertical();
        GUILayout.Label(" AutoModel Ver.1.0 ");
```

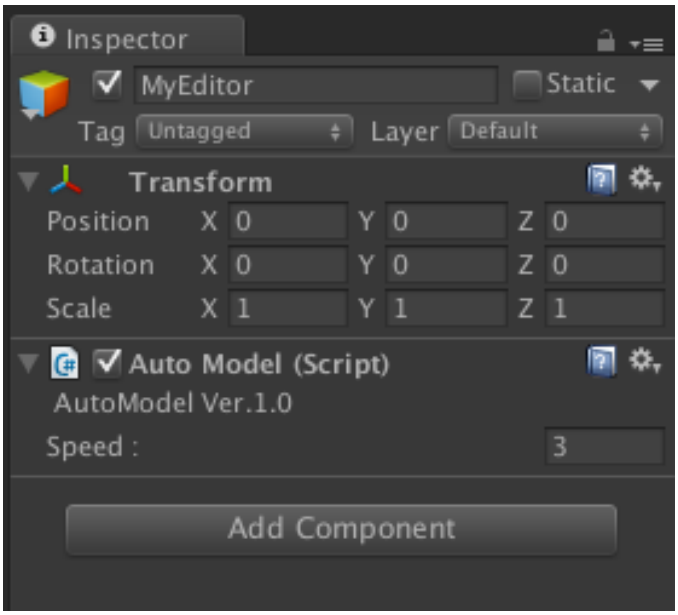
```
        GUILayout.BeginHorizontal();
        GUILayout.Label("Speed : ");
        autoModel.speed = EditorGUILayout.FloatField(autoModel.speed, GUILayout.Width(50));
        GUILayout.EndHorizontal();

        GUILayout.EndVertical();
    }


}
```

這裡增加了一個方法「OnEnable()」，這個就類似一般腳本的 Start()，但 Start 只會在執行時才會被觸發，所謂的編輯工具都是在編輯階段執行的，因此並不會被執行到 Start 動作，所以初始化必須放在 OnEnable() 裡面。

這裡建立了一個「AutoModel 」物件，取名為「autoModel 」，並在 OnEnable 執行時取得參考目標物，也就是 MyEditor 裡的 AutoModel 腳本，將他存在 autoModel 變數裡。

接下來修改 OnInspectorGUI，把原來的按鈕改成 FloatField 輸入項，並將修改值回傳給 AutoModel 的 speed 參數內。

回到 Inspector，這時可以看到多了 Speed 的控制內容了，我們可以填入別的數值並執行看看，Debug.Log 應該會印出修改後的數值。



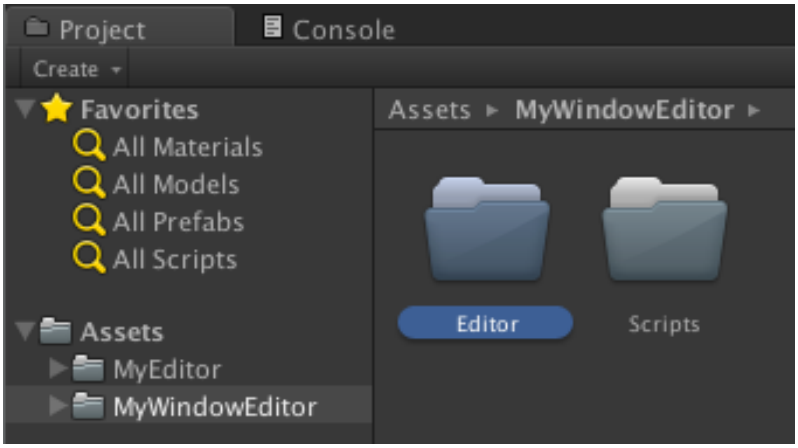
利用 EditorWindow 類別建立工具視窗

Editor 的類別提供了我們建立 Inspector 的基本方式，可是使用這種方法有幾個缺點，首先是必須手動將程式碼拉到場景的物件上才能使用，其次是要進行編輯時必須先選取該物件，這不太符合關卡編輯或場景編輯的習慣與需求，因此 Editor 比較適合拿來編輯物件本身，雖然他也可以建立或管理場景中所有物件，但使用上很不直覺。

通常我們製作關卡、事件、場景用的編輯器，會使用 EditorWindow 的方式建立。

建立主視窗程式

為了要開啟編輯器的編輯視窗，我們需要建立自己的主視窗，重新建立一個資料夾並取名為「MyWindowEditor」，並在裡面開兩個資料夾「Scripts」和「Editor」，和前面 Editor 類別的步驟都一樣。



進入 Editor 資料夾內，建立一個 C#的腳本取名為「ModelCreater.cs」，開啟這個腳本檔先刪除所有的內建方法，然後輸入以下的程式碼。

```
using UnityEngine;
using System.Collections;
using UnityEditor;


public class ModelCreater :  EditorWindow {

}
```

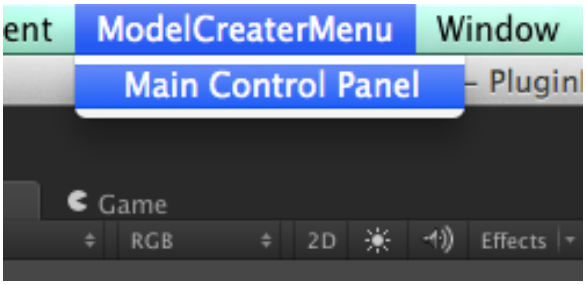
和之前的 Editor 類別相同，只是這次我們要繼承的是 EditorWindow 類別，這個類別可以為我們建立一個獨立的操作視窗，我們可以在裡面放入自己所需的所有功能，除了可以像 Editor 一樣以 Layout 方式繪製控制項以外，也可以自由的繪圖，功能非常強大。

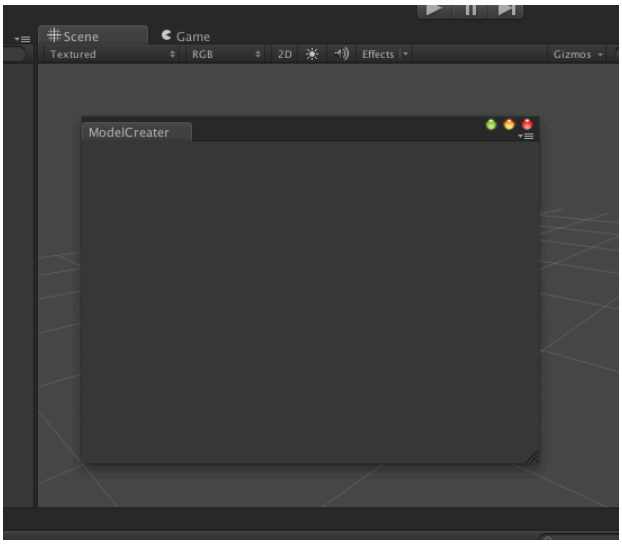
建立主選單

有了主視窗我們需要建立選單來開啟此視窗，再新增一個 C#腳本檔，取名為「ModelCreaterMenu.cs」開啟後輸入以下的程式碼。

```
using UnityEngine;
using System.Collections;
using UnityEditor;

public class ModelCreaterMenu : EditorWindow {

   [MenuItem ("ModelCreaterMenu/Main Control Panel", false, 0)]
   static void ModelCreater() {
      EditorWindow.GetWindow(typeof(ModelCreater));
   }

}
```

存檔後隨便按一下上方的 Menu，Unity 會自動新增一個新的選單「ModelCreaterMenu」，按下去就可以看到 Main Control Panel，點下去便可以開啟一個視窗了。



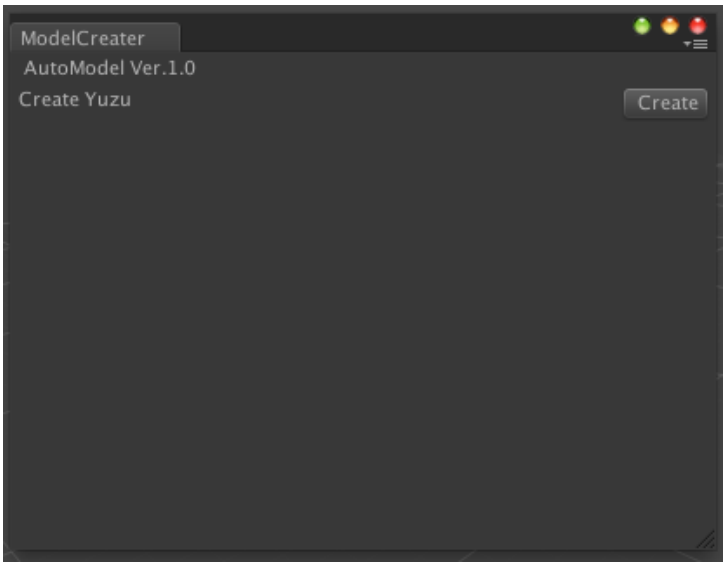


視窗內的控制項

開啟「ModelCreater.cs」腳本檔，我們可以像 Editor 類別一樣使用 Layout 繪製控制項，不同的是這次不是要繪在 Inspector 上，而是要繪製在我們自己的視窗中，因此直接使用 OnGUI 便可以在裡面繪製控制項。

```
public class ModelCreater :  EditorWindow {

    void OnGUI() {
        GUILayout.BeginVertical();
        GUILayout.Label(" AutoModel Ver.1.0 ");

        GUILayout.BeginHorizontal();
        GUILayout.Label("Create Yuzu ");
        GUILayout.Button("Create", GUILayout.Width(50));
        GUILayout.EndHorizontal();

        GUILayout.EndVertical();
    }
}
```

重新開啟視窗後便可以看到我們繪製好的控制項了。



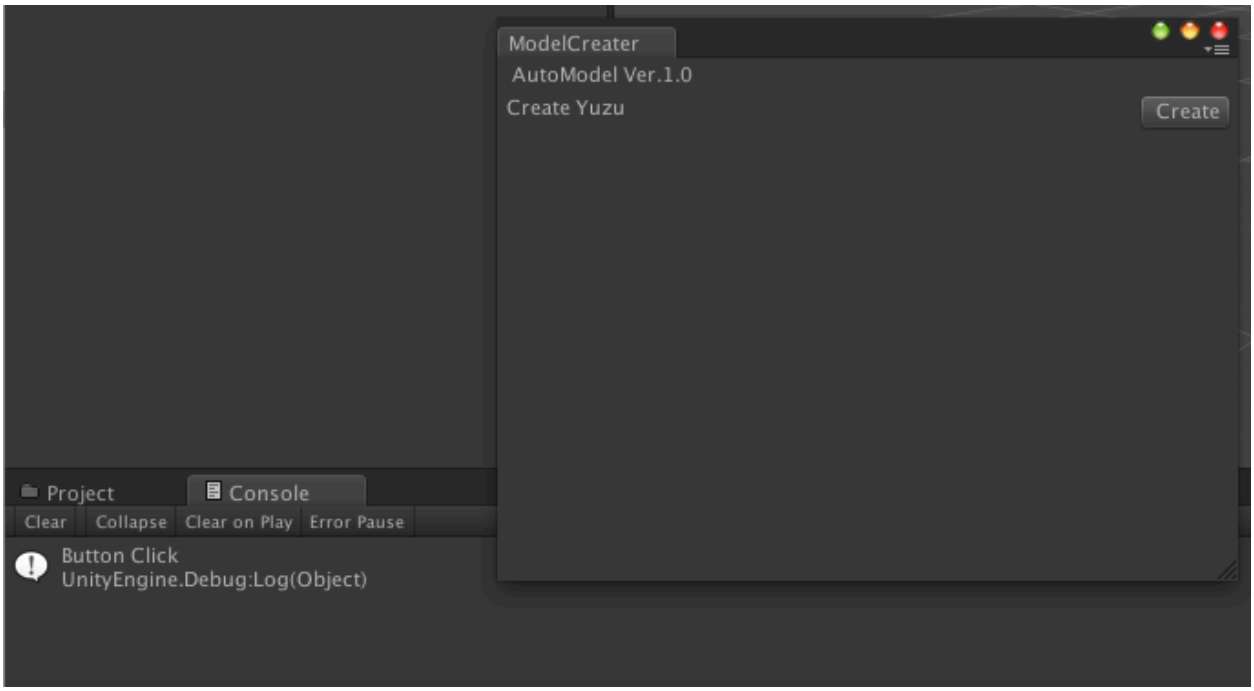
建立視窗內按鈕事件

單純繪製一個視窗和按鈕對編輯器來說沒什麼意義，現在我們要加入按鈕按下後的事件才能成為一個編輯器，加入事件的方法就和一般的 OnGUI 裡面按鈕事件觸發方式一樣，直接使用 if 判斷是否為 true 即可。

我們改寫建立 Button 的程式碼，加入判斷式。

```
GUILayout.BeginHorizontal();
      GUILayout.Label("Create Yuzu ");
      if(GUILayout.Button("Create", GUILayout.Width(50)))
      {
         Debug.Log("Button Click");
      }
      GUILayout.EndHorizontal();
```

當按下了按鈕後，便印出"Button Click"的 Log 出來。

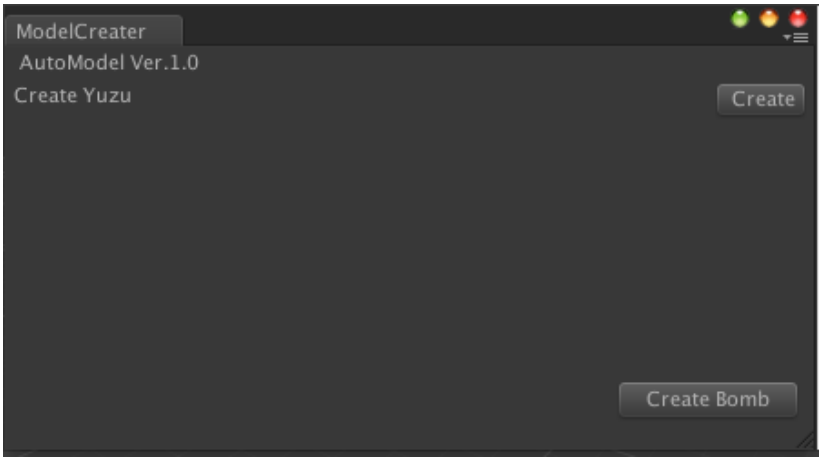


接下來我們便能像寫程戲一樣，透過編輯視窗為場景中的物件設置屬性或是動態建立物件到場景中了。

在視窗任意位置繪製元件

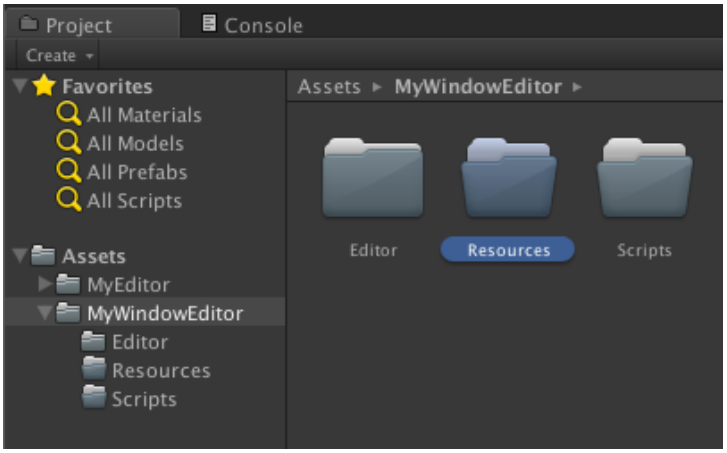
GUI 除了排列繪製以外，也可以利用 Rect 指定任意位置進行繪製，只要利用關鍵字「Handles.BeginGUI(); . . . . . Handles.EndGUI();」便可以在中間利用一般的 GUI 指令會製元件。

```
void OnGUI() {
    GUILayout.BeginVertical();
    GUILayout.Label(" AutoModel Ver.1.0 ");

    GUILayout.BeginHorizontal();
    GUILayout.Label("Create Yuzu ");
    if(GUILayout.Button("Create", GUILayout.Width(50)))
    {
        Debug.Log("Button Click");
    }
    GUILayout.EndHorizontal();

    GUILayout.EndVertical();


    Handles.BeginGUI();
    GUI.Button(new Rect(Screen.width-110,Screen.height-60,100,20), "Create Bomb");
    Handles.EndGUI();
}
```



建立物件到 SceneView

繪製物件到 SceneView 中是編輯器基本的功能，為了要能動態讀入資源，我們要建立一個放置資源的資料夾，Unity 要動態讀取資源必須將檔案放在 Resources 資料夾內，這個資料夾不需要放在根目錄，放置於任何的資料夾下面即可，請在「MyWindowEditor」底下建立一個資料夾並命名為「Resources」。

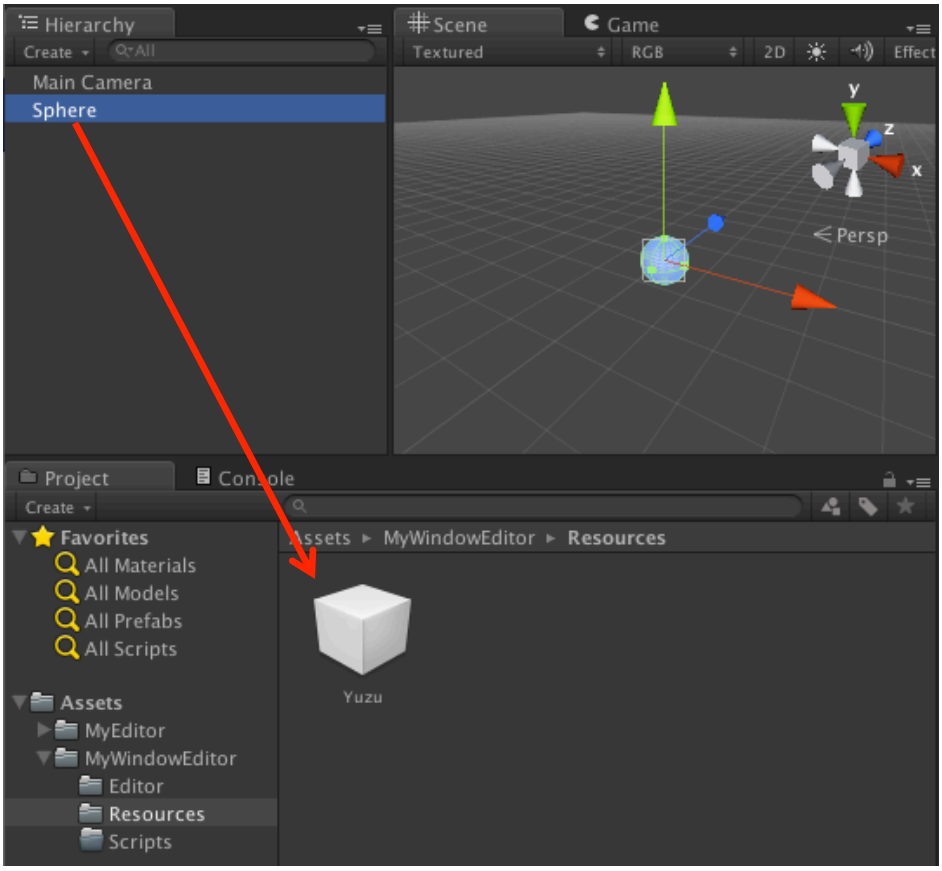


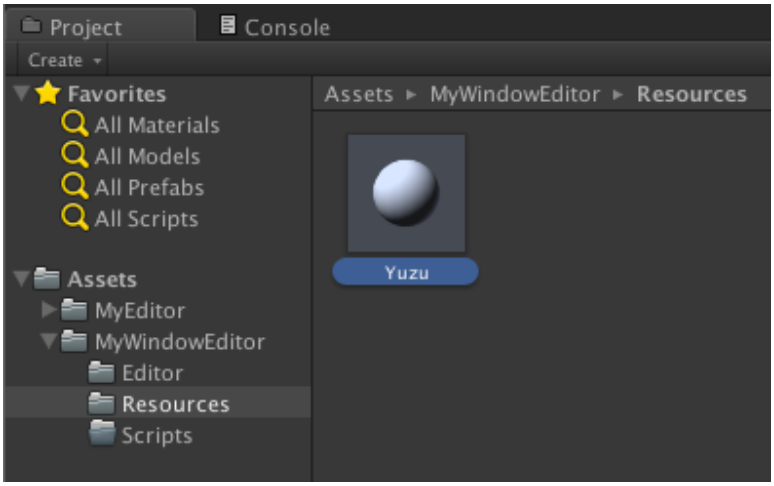
讀取資源的方式和一般開發遊戲讀取資源的方法一樣，直接使用「Resources.Load()」指令即可讀取資源，在這之前，我們先把模型給放進來，這裡我直接使用內建的 Cube 製作成 Prefab 來當作遊戲資源。

先在場景中建立一個球體。

GameObject > Create Other > Sphere

完成之後到 Resources 資料夾底下建立一個 Prefab，取名為 Yuzu，並將這個 Sphere 拉到 Prefab 裡面。



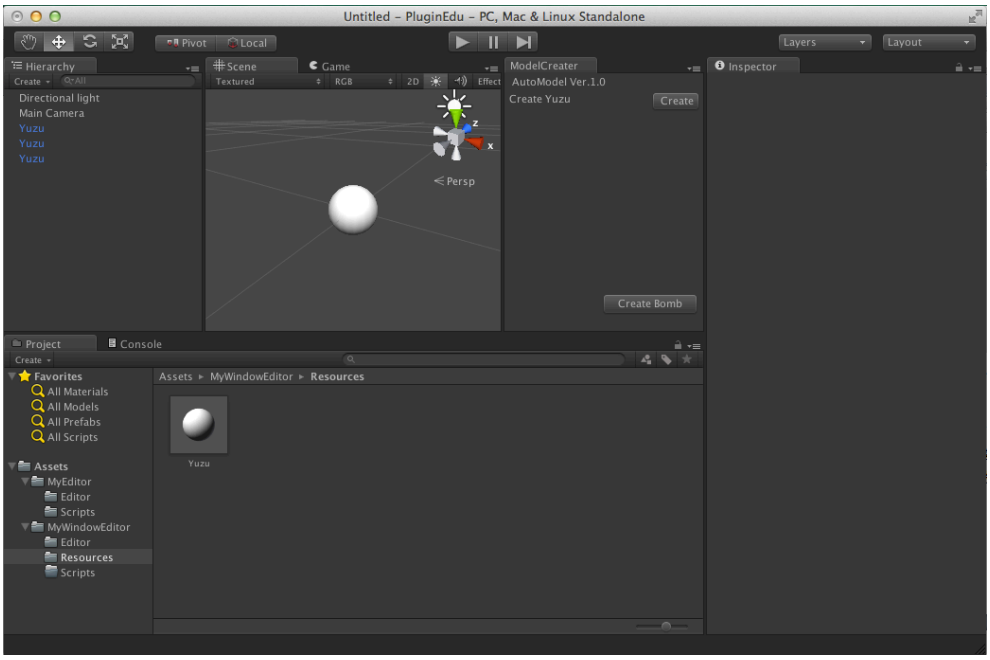


建立好 Prefab 後就可以將場景中的 Sphere 刪除，接下來改寫 Button 的事件，之前我們按下按鈕只印出一個「Button Click」的 Log，現在我們來改成建立一個模型到場景中。

以「Resources.Load()」讀取資源後用「PrefabUtility.InstantiatePrefab()」可以將此資源複製一份到場景中，複製好之後可以修改 transform 的屬性讓這個 Prefab 放到正確的位置上，這裡先直接放在 (0, 0, 0) 的位置上。

```
if(GUILayout.Button("Create", GUILayout.Width(50)))
{
    GameObject YuzuObj = (GameObject)Resources.Load("Yuzu", typeof(GameObject));
    GameObject obj = PrefabUtility.InstantiatePrefab(YuzuObj);
    obj.transform.position = new Vector3(0f,0f,0f);
}
```

現在我們可以測試一下程式碼，開啟 EditorWindow 後，為了方便使用可以將此 Window 固定在 Unity 的編輯視窗任意的地方，例如 Inspector 旁邊，然後按下 Button 便會在畫面的 (0, 0, 0) 的地方建立一顆球體了，雖然可以一直建立，不過因為全部疊在一起看不出來建立多顆球，之後我們會讀取滑鼠資訊讓球建立在滑鼠所在位置上，為了測試我打了一盞平行燈上去。



到這裡還有一個小缺點，當我們每次按下按鈕時都會重新從 Resources 裡讀取一次 Prefab 再進行繪製，若編輯器大一點；資源多一些時，這種做法無疑會降低不少效能，因此我們應該在一開始時便將用到的資源全部讀進來，然後按下按鈕直接進行複製，在之前已經有介紹過 Editor 底下的「OnEnable() { }」，在 EditorWindow 一樣也是在一開始時會先呼叫這個函式，我們可以將 Resources.Load 移到這裡來，如此就不會一直重新讀取了。

```
public class ModelCreater :  EditorWindow {

   GameObject YuzuObj;

   public void OnEnable()
   {
      YuzuObj = (GameObject)Resources.Load("Yuzu", typeof(GameObject));
   }


   void OnGUI() {
```

```
        GUILayout.BeginVertical();
        GUILayout.Label(" AutoModel Ver.1.0 ");

        GUILayout.BeginHorizontal();
        GUILayout.Label("Create Yuzu ");
        if(GUILayout.Button("Create", GUILayout.Width(50)))
        {
            GameObject obj = PrefabUtility.InstantiatePrefab(YuzuObj);
            obj.transform.position = new Vector3(0f,0f,0f);
        }
        GUILayout.EndHorizontal();

        GUILayout.EndVertical();


        Handles.BeginGUI();
        GUI.Button(new Rect(Screen.width-110,Screen.height-60,100,20), "Create Bomb");
        Handles.EndGUI();
    }
}
```

編輯器的主迴圈

如同開發遊戲的「Update() { } 」，在編輯器中也可以設置一個主迴圈，來讓編輯器可以隨時判斷鍵盤或滑鼠等等的輸入裝置觸發的訊號，主圈迴的設置可在「OnEnable()」裡利用運算元「 += 」建立委派事件「 SceneView.onSceneGUIDelegate 」，如此一來只要 SceneView 有進行重繪時便會呼叫我們的事件函式，通常建立委派都會順便建立移除的事件，在 EditorWindow 要被關閉時會呼叫「OnDisable()」，我們可以在這裡利用運算元「-=」移除委派事件。

```
public void OnEnable()
{
    YuzuObj = (GameObject)Resources.Load("Yuzu", typeof(GameObject));

    SceneView.onSceneGUIDelegate += OnSceneGUI;
}

public void OnDisable()
{
    SceneView.onSceneGUIDelegate -= OnSceneGUI;
}

public void OnSceneGUI( SceneView sceneView )
{
}
```

讀取滑鼠事件

利用 Event.current 可以讀到所有的輸入事件，包括鍵盤和滑鼠的事件，一般來說要使用滑鼠的行為不能使用滑鼠左鍵，因為在編輯狀態下滑鼠左鍵已被 Unity 使用掉，Unity 利用左鍵來選取場景中的物件，因此左鍵的指令會被 Unity 吃掉，我們必須使用滑鼠右鍵來輸入滑鼠指令。

因為 Unity 底下「Alt+滑鼠」是有作用的，所以我們要先判斷沒有按住「Alt」時才執行我們的指令。

```
public void OnSceneGUI( SceneView sceneView )
{
    Event e = Event.current;
    if( ! e.alt )
    {
        if( e.type == EventType.MouseDown && e.button == 1 )
        {
            e.Use();

            // Painting something

        }
    }

    if( e.type == EventType.MouseUp )
    {
        // release painting
    }
}
```

若有必要在滑鼠放開時釋放資源，可以讀取事件「EventType.MouseUp」來進行釋放動作。

讀取滑鼠座標

在 3D 環境底下讀取滑鼠座標不能直接取得滑鼠的 x,y 參數，原因是滑鼠的參數是螢幕上的 2D 座標，我們必須將這個座標轉成 3D 座標才行，具體的方法是從滑鼠點擊處發射一個射線出去打到地面(xz 平面)或是牆面(xy 平面)，籍此取得 3D 中的座標。

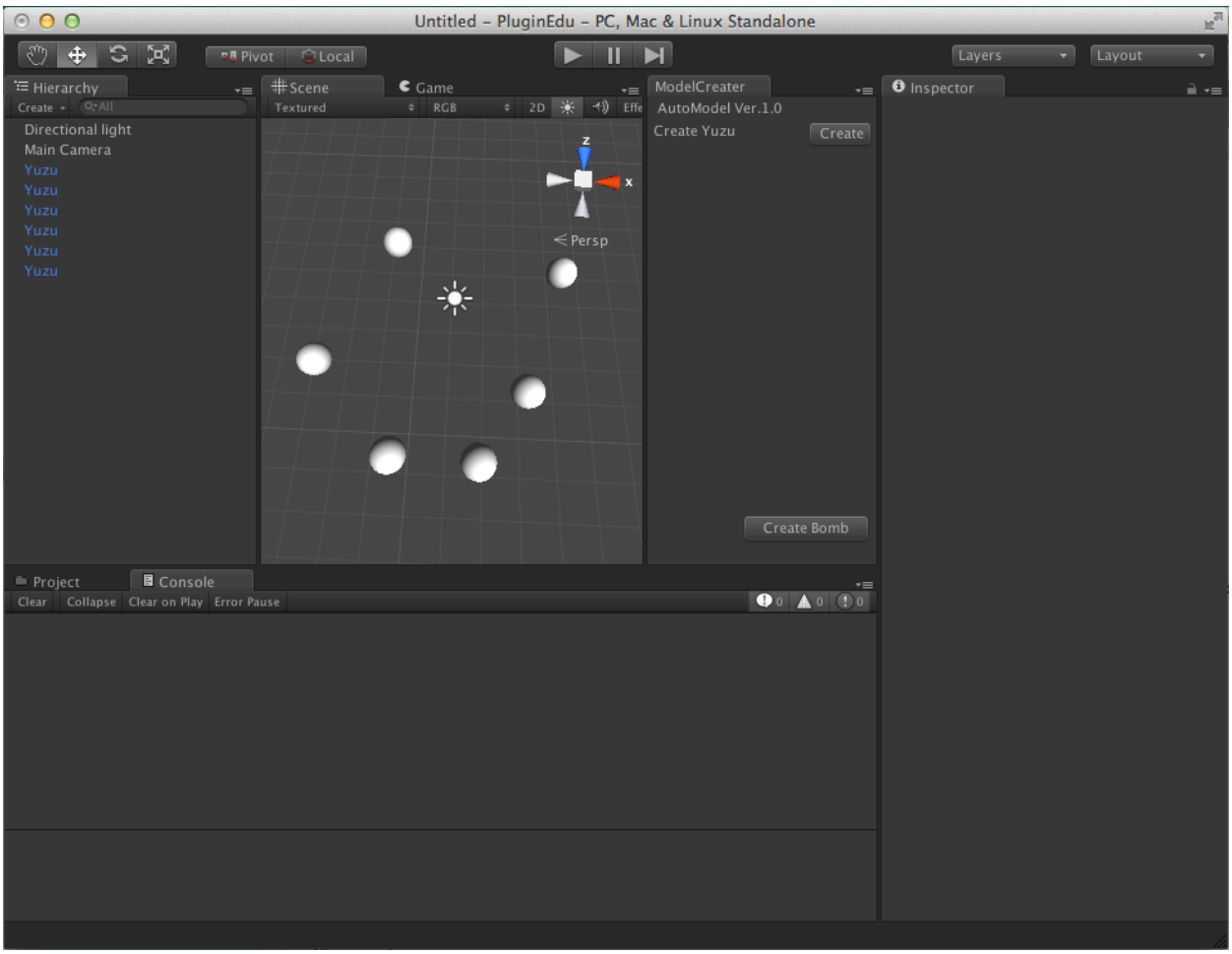
這段程式碼先利用 Ray 從螢幕的滑鼠點擊處打出一個射線，接下來判斷射線的 y 值必需小於 0，若大於 0 表示沒有打到平面上，接下來用 origin 換算成 xz 平面的座標。最後在此座標處用「PrefabUtility.InstantiatePrefab()」繪製物件。

```
        if( e.type == EventType.MouseDown && e.button == 1 )
        {
            e.Use();

            Ray mouseRay = Camera.current.ScreenPointToRay(new Vector3(e.mousePosition.x, Camera
.current.pixelHeight - e.mousePosition.y, 0.0f));

            if (mouseRay.direction.y <= 0.0f)
            {
                float t = -mouseRay.origin.y / mouseRay.direction.y;
                Vector3 mouseWorldPos = mouseRay.origin + t * mouseRay.direction;
                mouseWorldPos.y = 0.0f;


                GameObject obj = (GameObject)PrefabUtility.InstantiatePrefab(YuzuObj);
                obj.transform.position = mouseWorldPos;
            }
        }
```

若工具要在 x,y 平面上編輯請改用這段程式碼。

```
        if( e.type == EventType.MouseDown && e.button == 1 )
        {
            e.Use();

            Ray mouseRay = Camera.current.ScreenPointToRay(new Vector3(e.mousePosition.x,  Camer
a.current.pixelHeight - e.mousePosition.y, 0.0f));

            if (mouseRay.direction.z >= 0.0f)
            {
                float t = -mouseRay.origin.z / mouseRay.direction.z;
                Vector3 mouseWorldPos = mouseRay.origin + t * mouseRay.direction;
```

```
            mouseWorldPos.z = 0.0f;


            GameObject obj = (GameObject)PrefabUtility.InstantiatePrefab(YuzuObj);
            obj.transform.position = mouseWorldPos;
        }
    }
```



利用按鈕選單選擇要繪製的物件

剛才的篇例直接繪製單一物件到 SceneView 裡面，實際上大部份會從選單中選取要繪製的物件，所以我們要修改一下選單的程式碼，將原本的按鈕改成是 SelectionGrid，使用此元件可以繪製出一排可保留狀態的按鈕，按鈕的內容可直接傳入文字陣列，Unity 會自動繪製好。

```
public class ModelCreater :  EditorWindow {

   GameObject YuzuObj;
   private static int selItem = 0;

    . . . . . .

   void OnGUI() {
      GUILayout.BeginVertical();
      GUILayout.Label(" AutoModel Ver.1.0 ");


      string[] ObjNameArray = new string[3] {"none", "prefab1", "prefab2"};
      selItem = GUILayout.SelectionGrid(selItem, ObjNameArray, 3, GUILayout.Width(290));


      GUILayout.EndVertical();


      Handles.BeginGUI();
      GUI.Button(new Rect(Screen.width-110,Screen.height-60,100,20), "Create Bomb");
      Handles.EndGUI();
   }
}
```

接下來重新修改繪製物件的程式碼，判斷 selItem 為 1 時才進行繪製，讀者可以不斷增加繪製的內容物，只要修改陣列的長度即可。

```
        if (mouseRay.direction.y <= 0.0f)
        {
            float t = -mouseRay.origin.y / mouseRay.direction.y;
            Vector3 mouseWorldPos = mouseRay.origin + t * mouseRay.direction;
            mouseWorldPos.y = 0.0f;


            if( selItem == 1 )
            {
                GameObject obj = (GameObject)PrefabUtility.InstantiatePrefab(YuzuObj);
                obj.transform.position = mouseWorldPos;
            }
        }
```

解析場景中的資源

在我們的編輯器完成關卡編輯之後，我們必須將這些關卡資訊讀到遊戲當中，因此我們還需要有隻程式專門解析場景中的物件，以判斷哪些是我們所編輯的物件及所代表的意義。

通常來說我們有幾種方法進行解析：

1. 利用物件名稱解析編輯資源
   直接利用名稱要小心，有可能發生非編輯物件名稱重複的現像，這種情形就會造成判斷上的錯誤。

2. 利用 Tag 解析編輯資源

這種方法大概是最多人使用的方式，事先將遊戲物件編成不同的 Tag 進行編輯，之後利用「GameObject.FindGameObjectsWithTag」取得該 Tag 的所有物件列表，好處是非常容易理解，程式碼也相對簡單，缺點是所有的 Prefab 都要事先加工過，設置正確的 Tag 才能夠使用。

3. 利用父物件加上物件名稱進行編輯資源管理

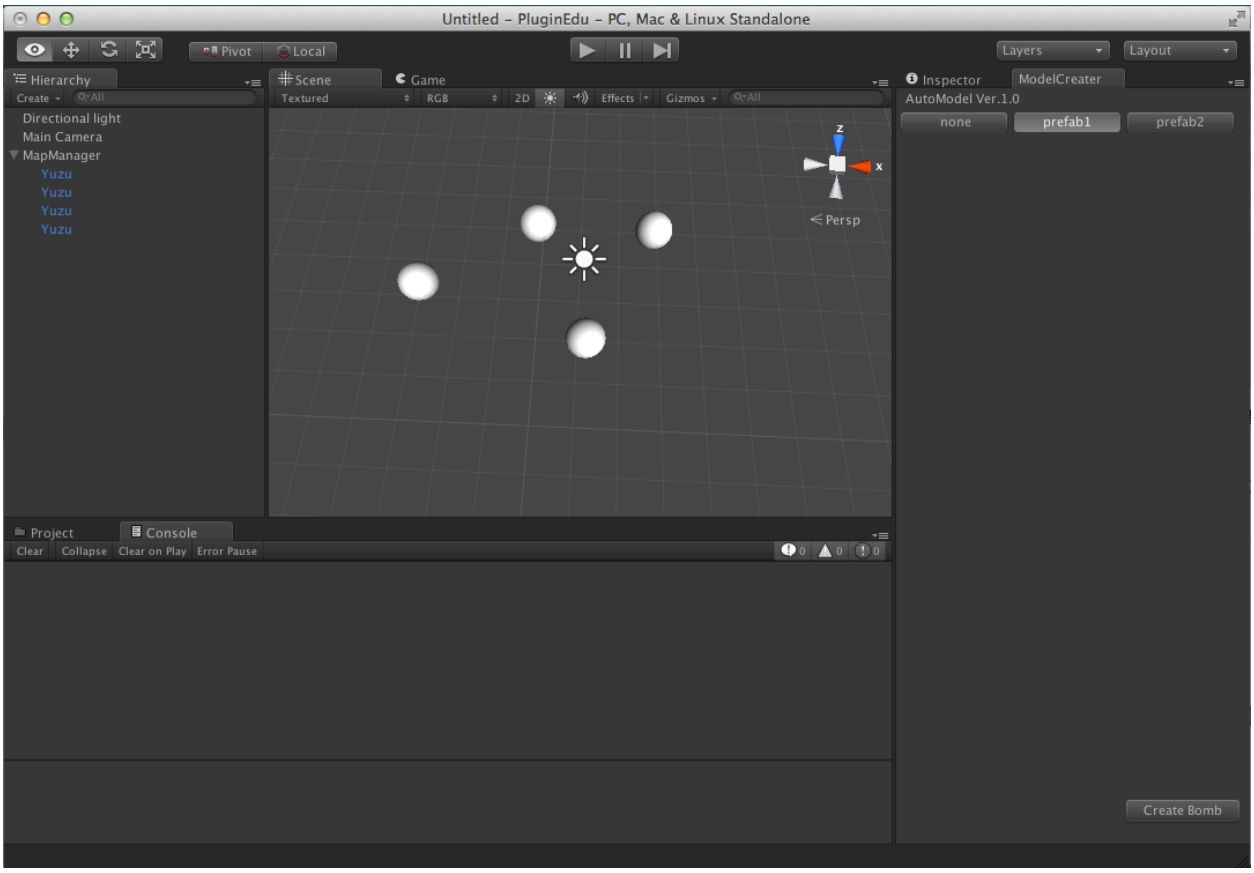
這是我推薦的方法，不用管 tag，在我們建立物件時直接將此物件歸到我們的一個管理器物件底下，然後利用「GetComponentsInChildren<Transform>()」取得所有編輯物件。

我們來看第三種的具體做法，首先在開啟視窗時先建立管理物件「MapManager」，若場景中已存在則不建立。

```
public class ModelCreater :  EditorWindow {

   GameObject YuzuObj;
   private static int selItem = 0;
   private static GameObject mapManager = null;

   public void OnEnable()
   {
      mapManager = GameObject.Find("MapManager");
      if( !mapManager )
      {
         mapManager = new GameObject();
         mapManager.name = "MapManager";
      }

      YuzuObj = (GameObject)Resources.Load("Yuzu", typeof(GameObject));

      SceneView.onSceneGUIDelegate += OnSceneGUI;
   }
```

接下來每建立的模型都歸到 MapManager 底下。

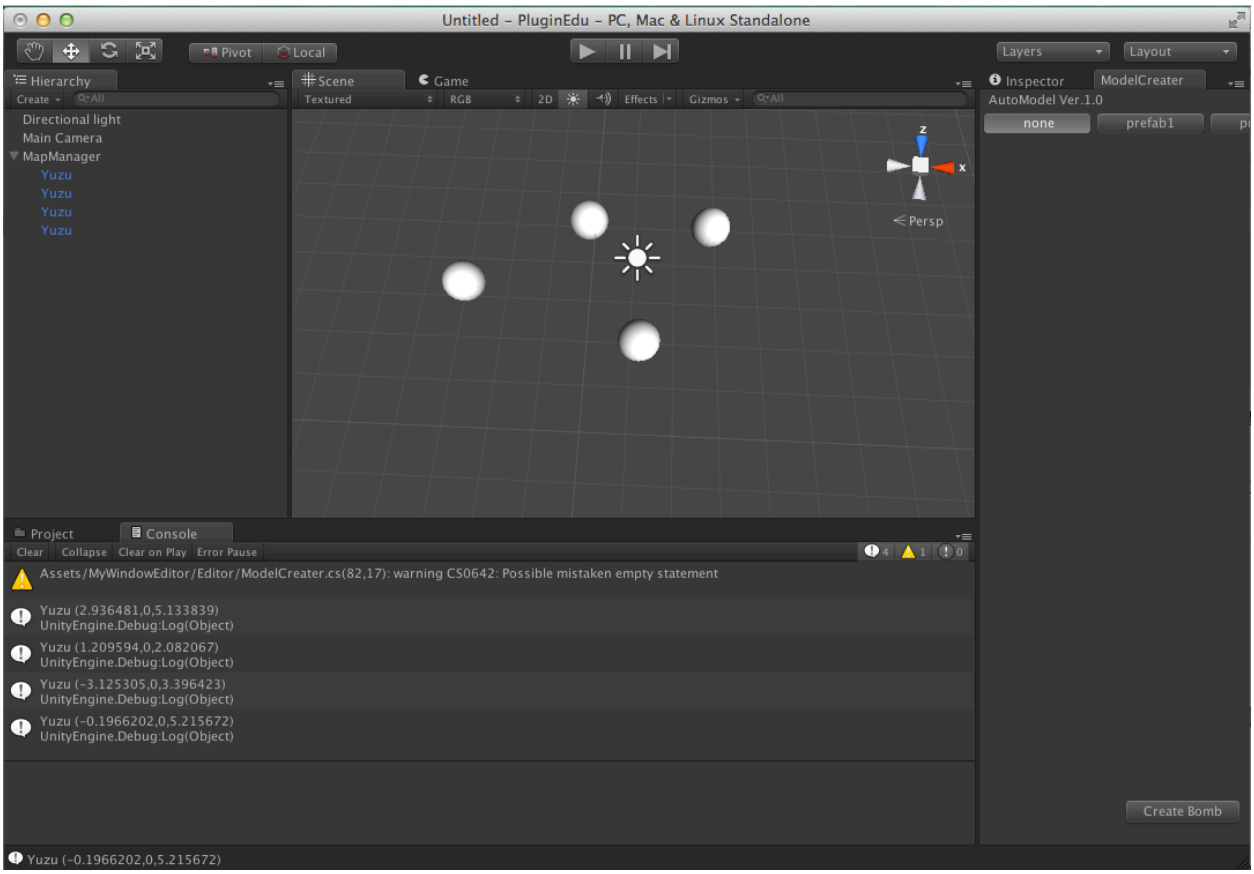
```
        if (mouseRay.direction.y <= 0.0f)
        {
            float t = -mouseRay.origin.y / mouseRay.direction.y;
            Vector3 mouseWorldPos = mouseRay.origin + t * mouseRay.direction;
            mouseWorldPos.y = 0.0f;


            if( selItem == 1 )
            {
                GameObject obj = (GameObject)PrefabUtility.InstantiatePrefab(YuzuObj);
                obj.transform.position = mouseWorldPos;
                obj.transform.parent = mapManager.transform;
            }
        }
```



當遊戲中要取得這些資料時可使用「GetComponentsInChildren<Transform>()」取得，為了示範我建立了一個新的 script 取名為「GetEditorComponent 」，加入以下的程式碼。完成後拉給攝影機，執行遊戲後按下按鈕便可以讀到所有的物件座標了

```
using UnityEngine;
using System.Collections;

public class GetEditorComponent : MonoBehaviour {

    private GameObject mapManager = null;
    // Use this for initialization
```

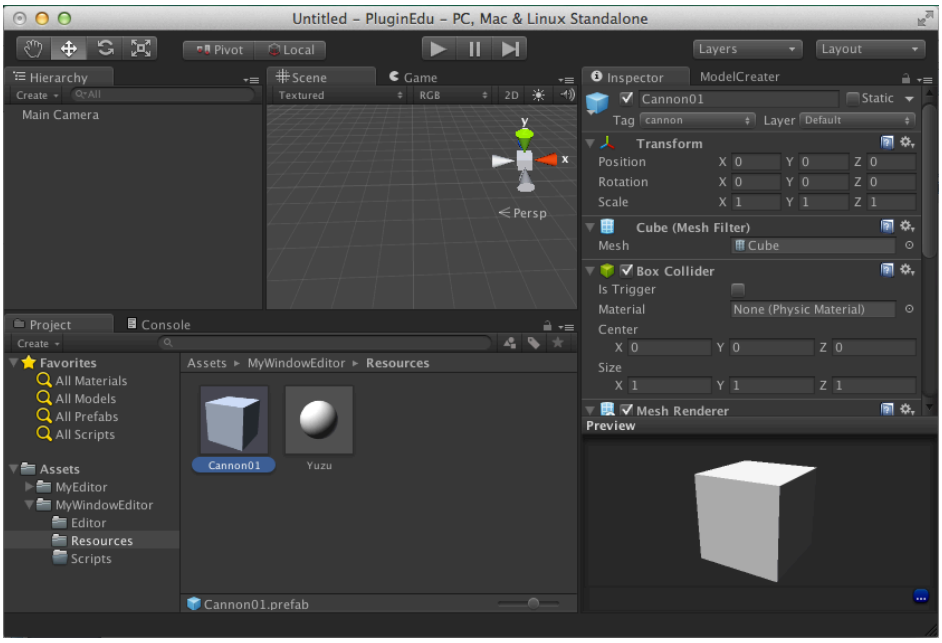
```
    void Start () {
        mapManager = GameObject.Find("MapManager");
    }

    // Update is called once per frame
    void Update () {

    }

    void OnGUI () {
        if( GUI.Button(new Rect(10,10,100,20), "GetComponent"))
        {
            if( mapManager != null )
            {
                Component[] editorObj = mapManager.GetComponentsInChildren<Transform>();
                foreach( Component eo in editorObj)
                {
                    if( eo.name == "Yuzu" )
                    {
                        Debug.Log(string.Format("Yuzu ({0},{1},{2})",eo.transform.position.x, eo.transform.position
.y, eo.transform.position.z));
                    }
                }
            }
        }
    }
}
```



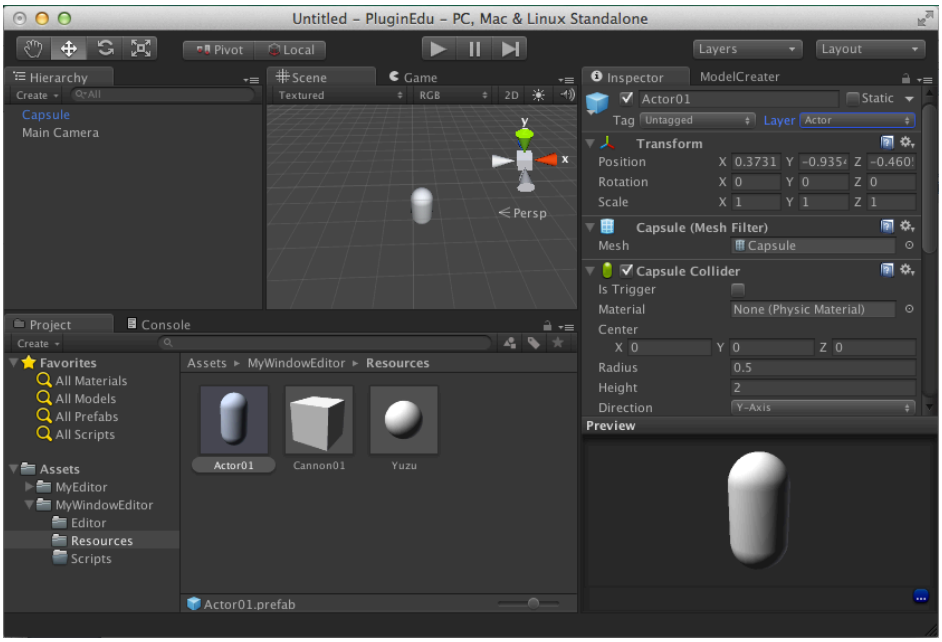
請勿以 Layer 規畫編輯資源

有一個比較特殊的狀況要另外說明，當規畫編輯器的資源時，可以利用 Tag，但最好不要使用 Layer 做資源的管理，這邊做一個實驗馬上就可以了解為什麼了。

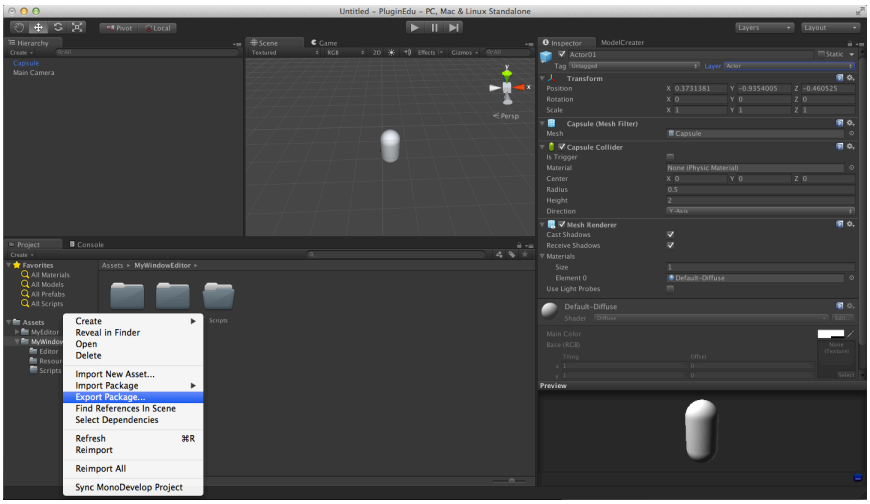
先在 Resources 裡建立一個 Prefab 取名為「Cannon01」，並新增一個 Tag 叫做「cannon」表示這種資源代表砲塔，然後將 Cannon01 的 Tag 設為 cannon。



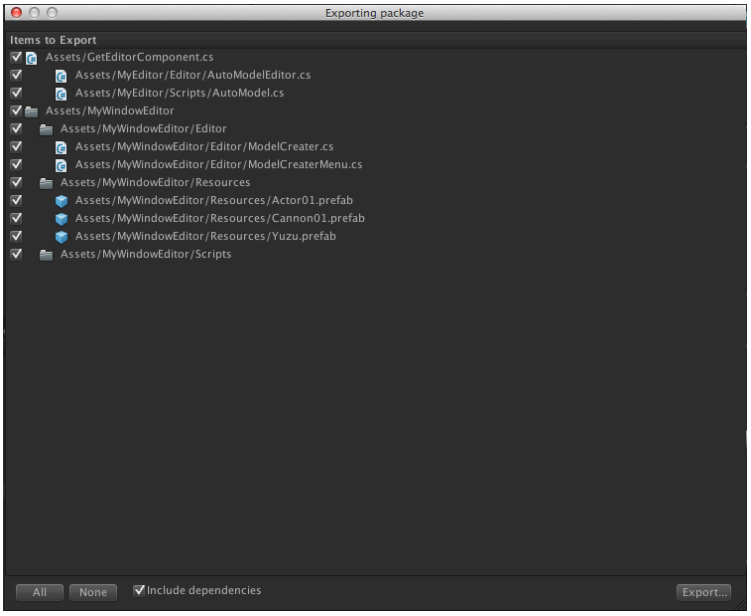
接著建立另一個 Prefab 取名為「Actor01」，這次不用 Tag 而改用 Layer，建立一個 Layer 叫「Actor」代表演員，並將 Actor01 這個 Prefab 設為此 Layer。



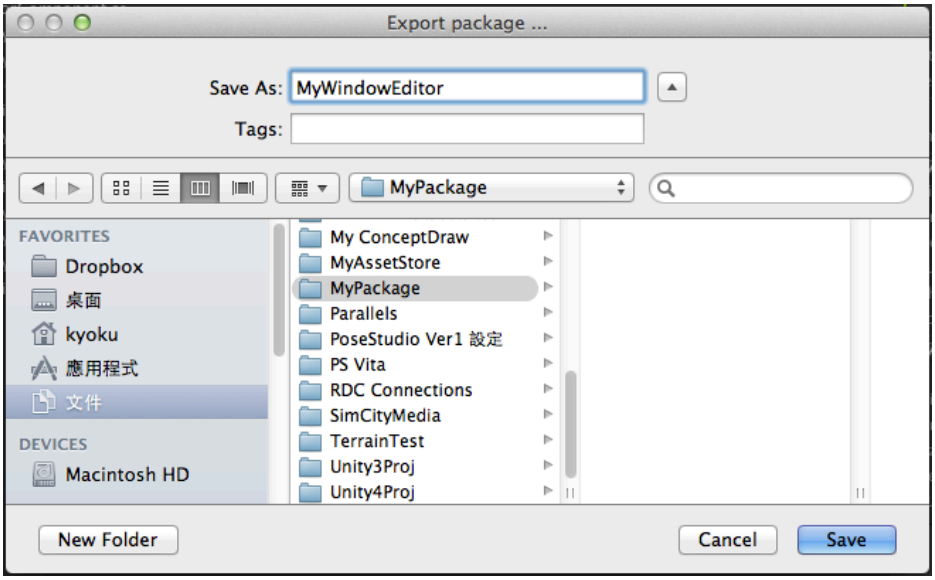
我們把「MyWindowEditor」資源匯出，在 MyWindowEditor 按滑鼠右鍵，選擇「Export Package ...」



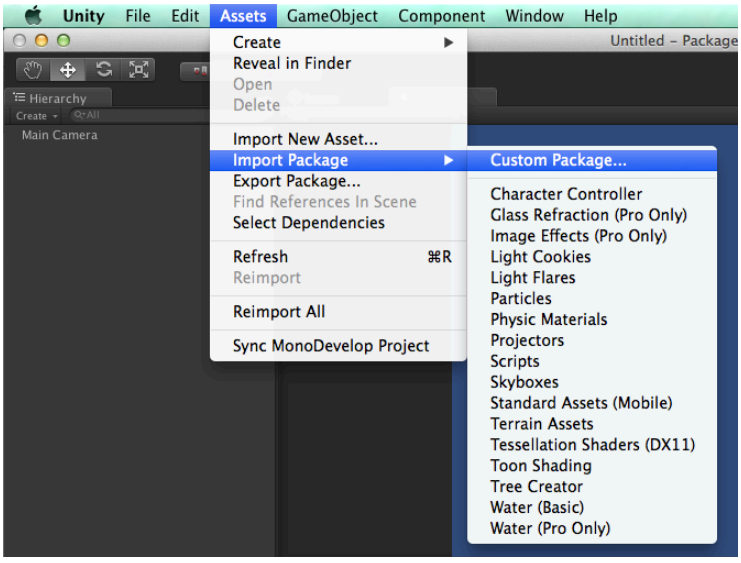
接下來人彈出的匯出視窗預設是選全部，不要變動，按下右下方的[Export]按鈕。



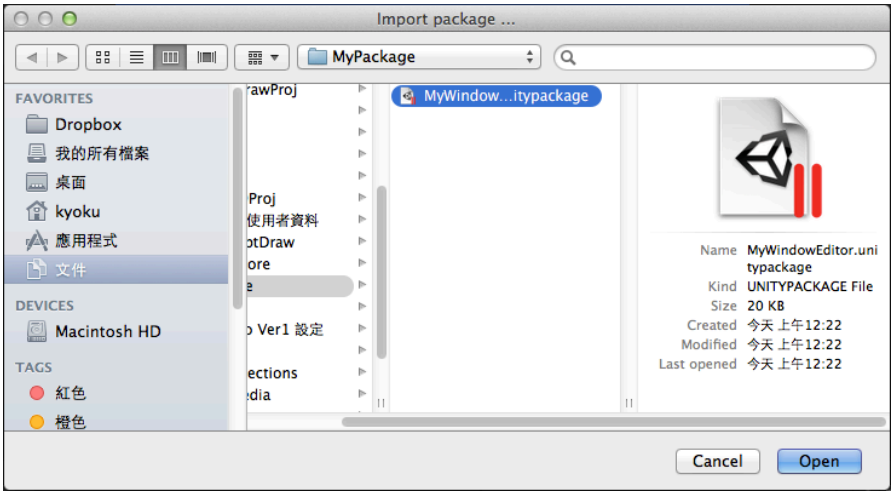
這時請選取要存放的位置並輸入名稱「MyWindowEditor」，亦可自行取名。



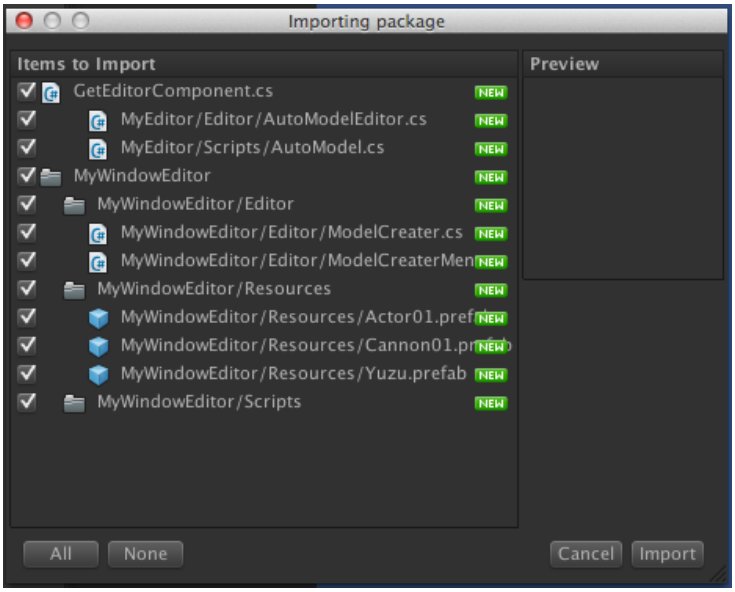
接下來另外建立一個新專案，建立好之後執行「Assets > Import Package > Custom Package...」



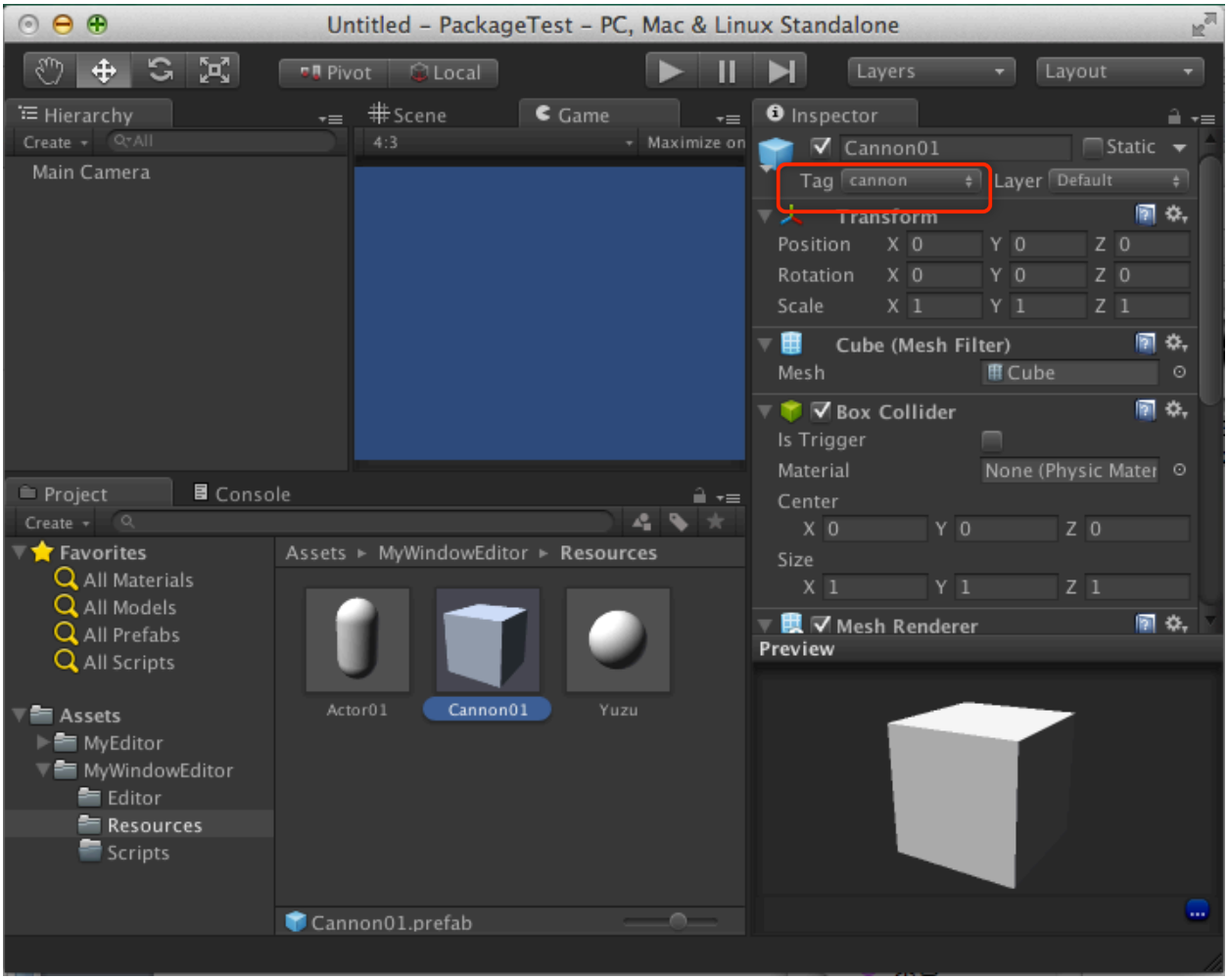
選取剛才存好的資源包。



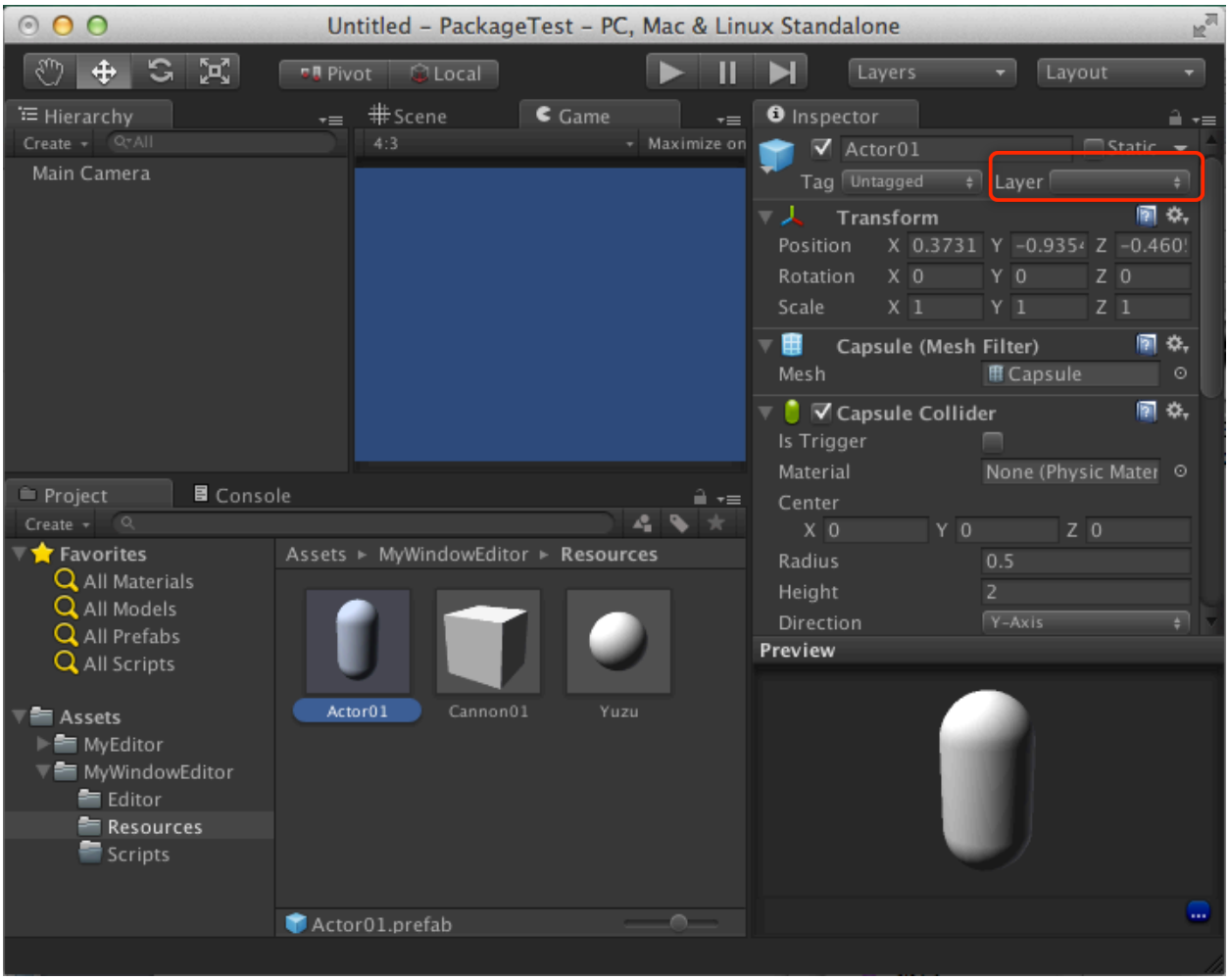
請選擇全部匯入，直接按右下的 Import 即可。



匯入成功後可以看到設置自訂 Tag 的 Cannon01 物件成功的保留了 Tag 過來。



但是以 Layer 分類的 Actor01 物件，Layer 卻遺失了。



因此最好不要使用 Layer 來進行資源的規畫。

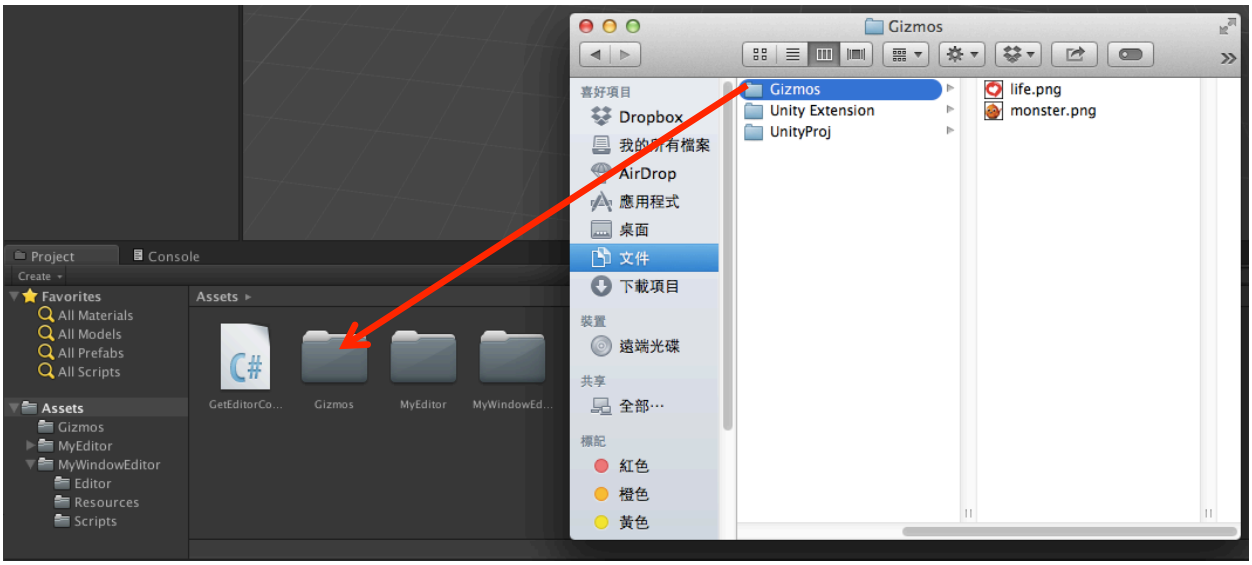
利用 Gizmos 在 SceneView 中繪製

除了編輯器外，就算一般做遊戲我們也時常需要在畫面中標示一些符號，例如生怪磚或是重生點之類的，希望在遊戲中看不到但在編輯中看得到，這種情形很適合利用 Gizmos 對這類物品進行繪製。

使用 Gizmos.DrawIcon 繪製圖示

匯入 Icon

為了示範，我準備了兩個 PNG 檔案格式的圖示，分別代表生怪點的 monter.png 和重生點的 life.png，放在 Gizmos 的資料夾，直接將此資料夾拉到 Unity 專案內的「根目錄」內，要使用 Gizmos 讀取的圖示一定要放在名為「Gizmos」的資料夾內並且此資料夾一定要放在根目錄。



建立生怪點和重生點的 Prefab

接下來回到「Scripts」資料夾，建立一個新的 C# Script 取名為「MonsterPoint.cs」，打開這個腳本檔，加入以下的程式碼。

```
using UnityEngine;
using System.Collections;

public class MonsterPoint : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
```

```
    }

    void OnDrawGizmos () {
        Gizmos.DrawIcon (this.transform.position, "monster.png");
    }
}
```

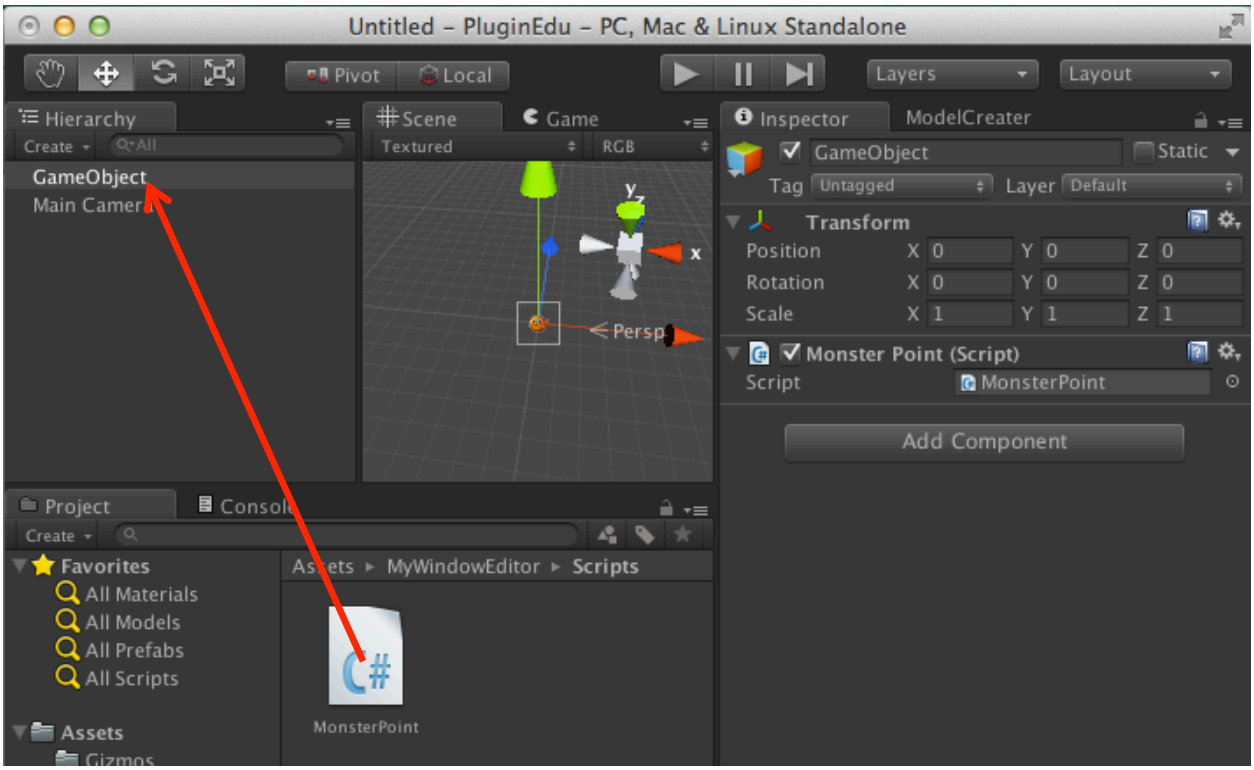
仔細看腳本的內容，有一個 OnDrawGizmos()的方法，這個也是 Unity 內建的方法之一，其作用便是可以將要繪製在 SceneView 的內容物寫在這裡，並且這裡的物件不會編譯到遊戲當中，所以遊戲裡不會看到這些圖示。

在 OnDrawGizmos()裡我們只簡單加了一個繪製 icon 的指令。

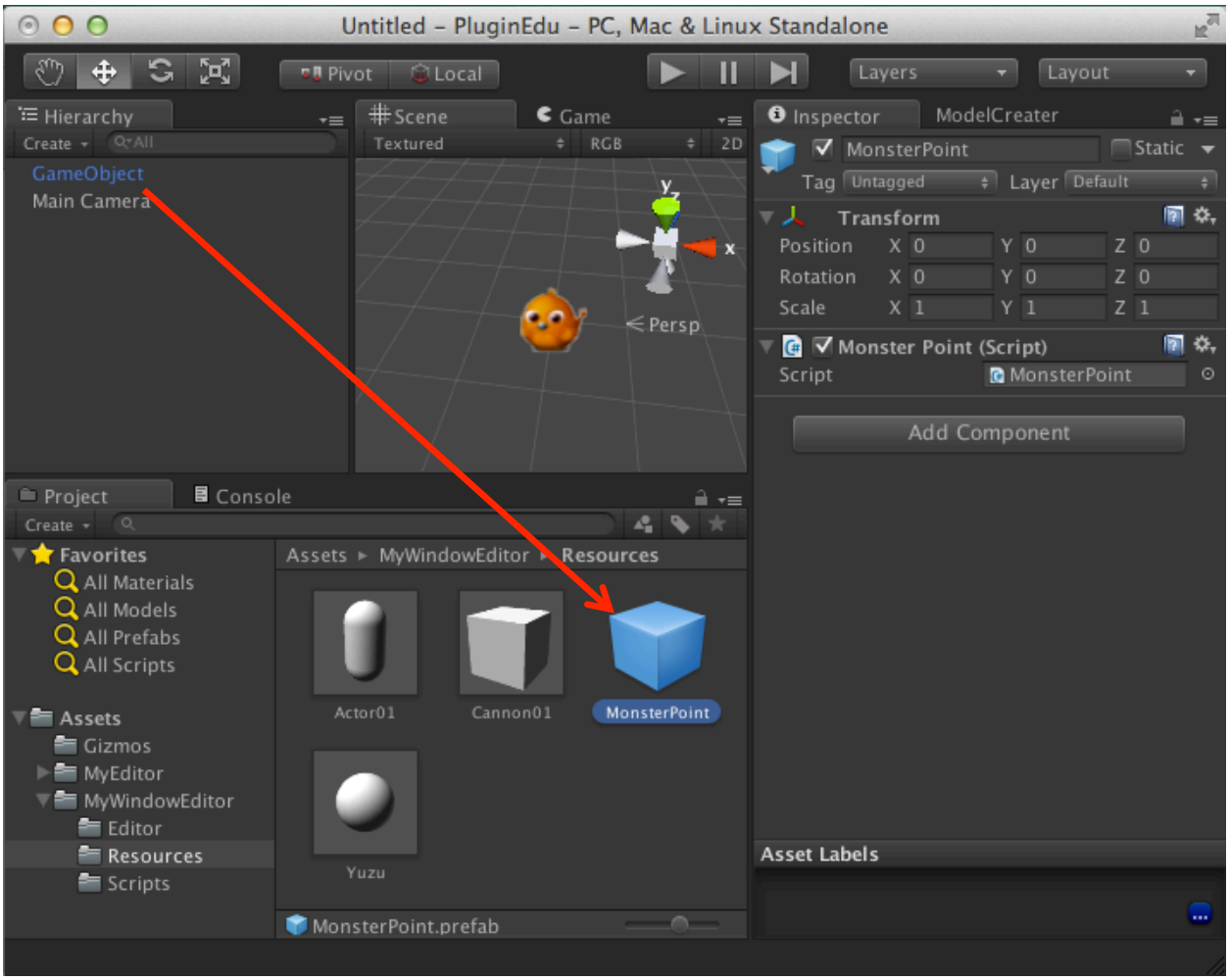
Gizmos.DrawIcon (this.transform.position, "monster.png");

在 Gizmos 的指令裡比較特殊的一點是，檔名要連同副檔名一起，鍵入完整的圖檔名稱。通常在此腳本內還會加上生怪點所需要的程式碼和參數等等，但此時我們的重點在編輯器，因此這些額外設置在此省略。

接下來建立一個生怪點物件，用一般的 Empty 即可，執行「GameObject > Create Empty」，記得檢查 GameObject 的參數，若不在原點就將其歸 0，然後將 MonsterPoint.cs 腳本拉到 GameObject 上，拉上去後 GameObject 會產生一個圖示，如下圖。

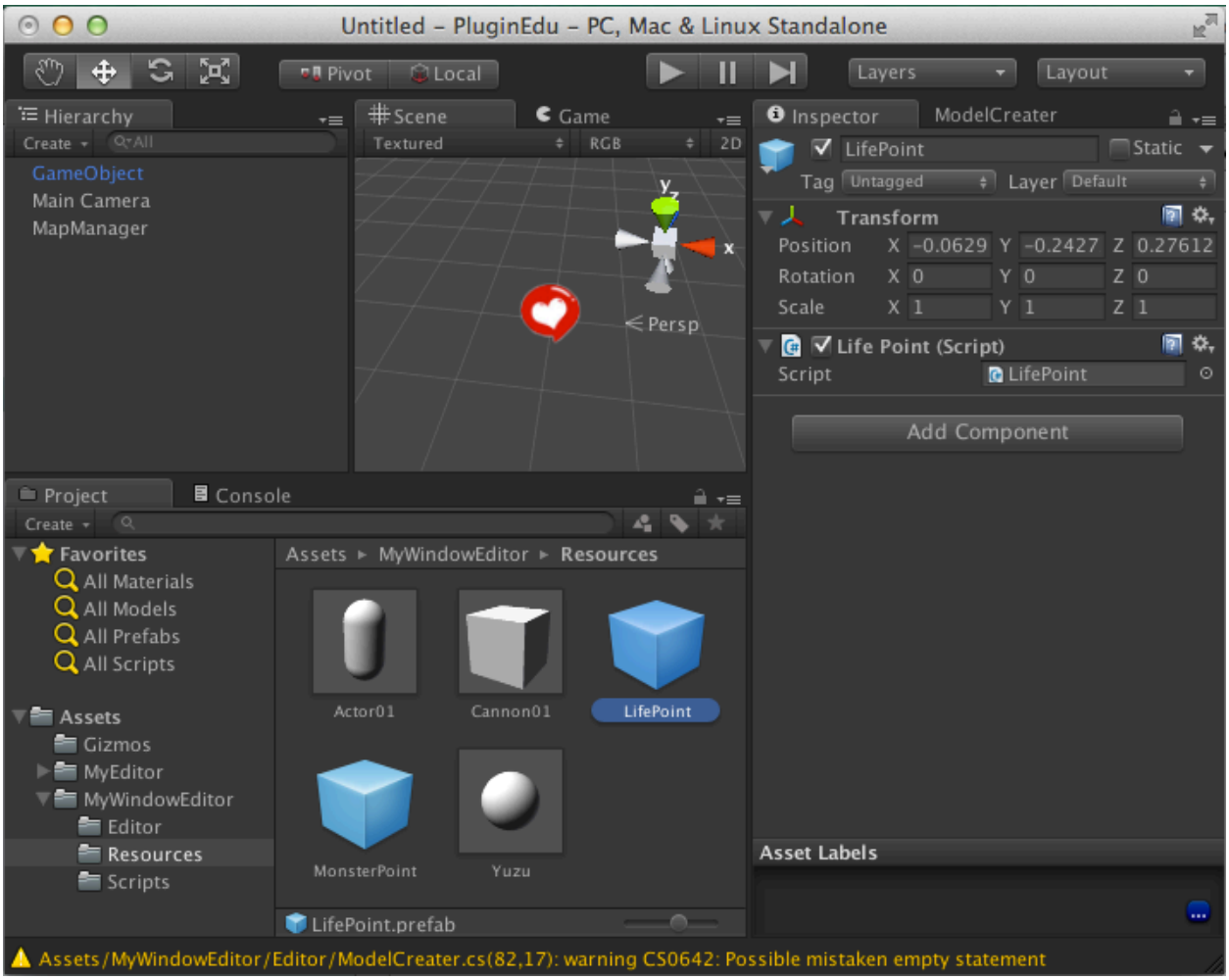


接下來建立生怪點用的 Prefab，到「MyWindowEditor/Resources」資料夾下，建立一個 Prefab，取名為「MonsterPoint」，然後將剛才的 GameObject 拉到 MonsterPoint 的 Prefab 上。



完成後就可以刪除 GameObject 了。

將另一個 LifePoint 也如法泡製建立好 LifePoint 的 Prefab，過程不再贅述。



修改 GUI 增加生怪點和重生點的功能

回到「ModelCreater.cs」，找到 OnGUI，修改 ObjNameArray 的內容，修改陣列元素數量為 4，並將第 3 個元素取名為「MonsterPoint」，第 4 個元素取名為「LifePoint」。

```
void OnGUI() {
      GUILayout.BeginVertical();
      GUILayout.Label(" AutoModel Ver.1.0 ");


      string[] ObjNameArray = new string[4] {"none", "prefab1", "MonsterPoint", "LifePoint"};
      selItem = GUILayout.SelectionGrid(selItem, ObjNameArray, 3, GUILayout.Width(290));


      GUILayout.EndVertical();


      Handles.BeginGUI();
      if( GUI.Button(new Rect(Screen.width-110,Screen.height-60,100,20), "Create Bomb"));
      Handles.EndGUI();
  }
```

接下來增加兩個型別為 GameObject 的成員變數「MonsterPointObj」和「LifePointObj」，並在 OnEnable()將這兩個成員用 Resources.Load 載入相對應的 Prefab。

```
  public class ModelCreater :  EditorWindow {

  GameObject YuzuObj;
  GameObject MonsterPointObj;
  GameObject LifePointObj;

  private static int selItem = 0;
  private static GameObject mapManager = null;

  public void OnEnable()
  {
     mapManager = GameObject.Find("MapManager");
     if( !mapManager )
     {
```

```
        mapManager = new GameObject();
        mapManager.name = "MapManager";
    }

    YuzuObj = (GameObject)Resources.Load("Yuzu", typeof(GameObject));
    MonsterPointObj = (GameObject)Resources.Load("MonsterPoint", typeof(GameObject));
    LifePointObj = (GameObject)Resources.Load("LifePoint", typeof(GameObject));

    SceneView.onSceneGUIDelegate += OnSceneGUI;
}
```

修改 OnSceneGUI，將原本的「if(selItem == 1)」改為使用「switch () . . . casse」的寫法，看起來比較整潔，程式碼和之前大同小異，只是改為複製物為 MonsterPointObj 和 LifePointObj 這兩個成員。

```
public void OnSceneGUI( SceneView sceneView )
{
    Event e = Event.current;
    if( ! e.alt )
    {
        if( e.type == EventType.MouseDown && e.button == 1 )
        {
            e.Use();

            Ray mouseRay = Camera.current.ScreenPointToRay(new Vector3(e.mousePosition.x, Camera
.current.pixelHeight - e.mousePosition.y, 0.0f));

            if (mouseRay.direction.y <= 0.0f)
            {
                float t = -mouseRay.origin.y / mouseRay.direction.y;
                Vector3 mouseWorldPos = mouseRay.origin + t * mouseRay.direction;
                mouseWorldPos.y = 0.0f;


                switch( selItem  )
```

```
            {
                case 1:
                {
                    GameObject obj = (GameObject)PrefabUtility.InstantiatePrefab(YuzuObj);
                    obj.transform.position = mouseWorldPos;
                    obj.transform.parent = mapManager.transform;
                    break;
                }
                case 2:
                {
                    GameObject obj = (GameObject)PrefabUtility.InstantiatePrefab(MonsterPointObj);
                    obj.transform.position = mouseWorldPos;
                    obj.transform.parent = mapManager.transform;
                    break;
                }
                case 3:
                {
                    GameObject obj = (GameObject)PrefabUtility.InstantiatePrefab(LifePointObj);
                    obj.transform.position = mouseWorldPos;
                    obj.transform.parent = mapManager.transform;
                    break;
                }
            }

        }
    }

    }

    if( e.type == EventType.MouseUp )
    {
        // release painting
    }


}
```

現在我們可以利用選單加上滑鼠右鍵繪製生怪點和重生點了，繪製出來的 icon 將永遠面對鏡頭，不用擔心轉向而看不清楚。

使用 Gizmos.DrawLine 繪製線條

Gizmos.DrawLine 也是常用的指令，一般常拿來繪製格線或走位編輯器裡角色的行進路線，這裡我們利用此功能來繪製 xz 平面的 Grid 格線作為示範。

到 Scripts 資料夾內新增一個腳本，取名為「Grid.cs」，開啟這個腳本填入下面的程式碼。

```
using UnityEngine;
using System.Collections;

public class Grid : MonoBehaviour {

    public bool displayGrid = true;
    public int gridLeft = -3;
    public int gridRight = 3;
    public int gridFront = -3;
    public int gridBack = 3;
    public Color gridLineColor = new Color(0.7f,0.7f,0.7f);

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }

    void OnDrawGizmos()
    {
        if( displayGrid )
        {

            if( gridBack < gridFront || gridLeft > gridRight )
            {
                Debug.Log("Grid Limit Error!!");
                return;
            }

            float coordinateBack = (float)gridBack  + 0.5f;
            float coordinateFront = (float)gridFront  - 0.5f;

            float coordinateLeft = (float)gridLeft  - 0.5f;
            float coordinateRight = (float)gridRight  + 0.5f;

            Gizmos.color = gridLineColor;

            // DrawLine from back to front
```

```
        for (float z = coordinateBack; z >= coordinateFront; z-= 1)
        {
            Gizmos.DrawLine(new Vector3(coordinateLeft, 0.0f, z),
                        new Vector3(coordinateRight, 0.0f, z));
        }

        // DrawLine from left to right
        for (float x = coordinateLeft; x <= coordinateRight; x+= 1)
        {
            Gizmos.DrawLine(new Vector3(x, 0.0f, coordinateBack),
                        new Vector3(x, 0.0f, coordinateFront));
        }
    }
  }
}
```

要用 Gizmos 繪製必須是繼承自 MonoBehaviour 的類別才行，所以繪製格線的程式碼我們不直接寫在「ModelCreater」裡，而是再建立這個新的腳本，因此我們簡單的利用 public 讓這些格線的參數可以在 Inspector 裡編輯(若你的變數想為 public 但又不想在 Inspector 裡編輯，可以在變數上方加上 [HideInInspector] 關鍵字)。為了可以隨時開關格線，我們設了一個變數「displayGrid 」，只有這個變數為 true 時才進行 Grid 繪製。

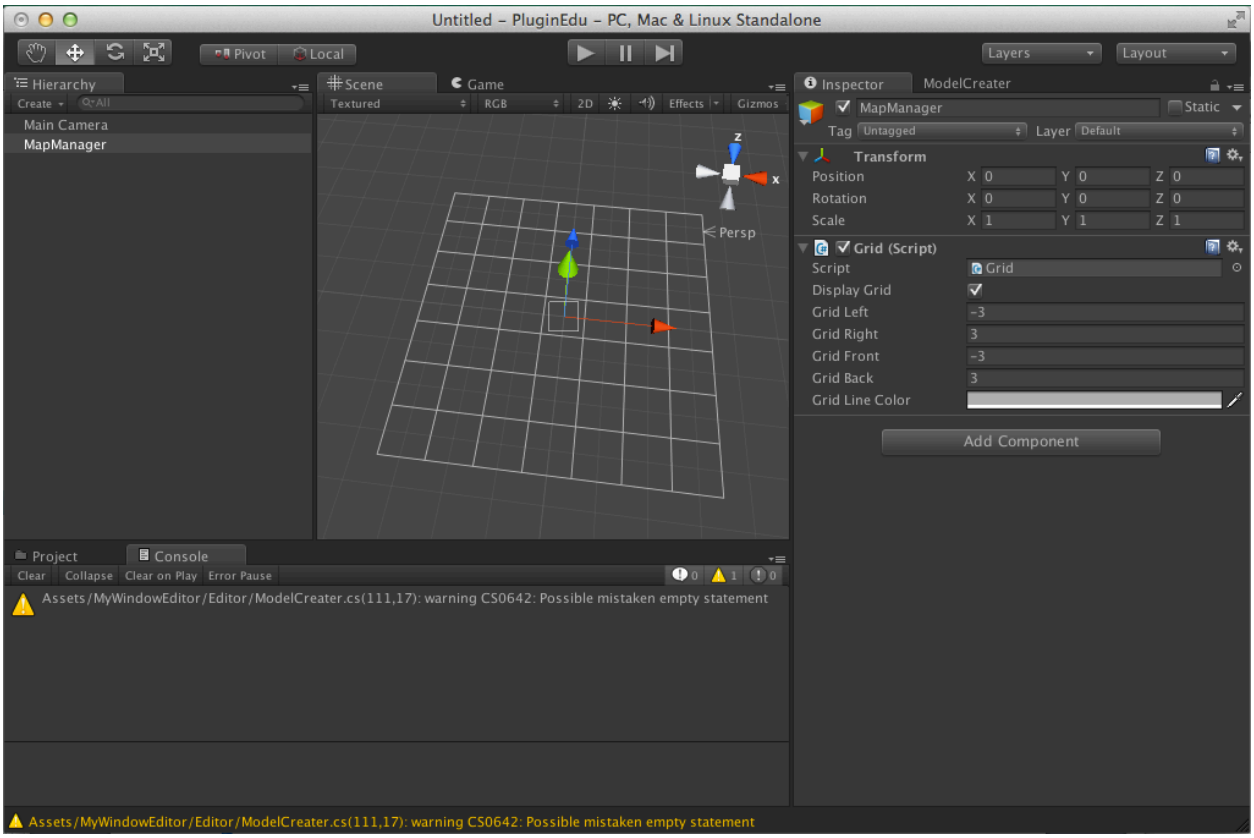
繪製前先用「if(gridBack < gridFront || gridLeft > gridRight) { }」判斷輸入的參數是不合理，若不合理就直接 return 不再執行後面的 code，接下來就利用兩個 for 迴圈每隔 1 的單位就繪製一條直線將線格繪製完成。

因為是繼承自 MonoBehaviour 的類別，因此要將這個腳本加到場景中的物件才能正確顯示，最簡單的做法便是加到我們的「MapManager」物件上，這個物件只要開啟視窗就會建立出來，最適合我們來放置格線的腳本了。但我們不能直接「拉」到 MapManager 上，因為這個物件是我們用程式自動建立出來的，所以我們要修改建立時的程式碼，當建立 MapManager 後，若此物件上沒有 Grid 的腳本，我們就自動將 Grid 腳本加上去。

回到「ModelCreater.cs」，在 OnEnable()建立 mapManager 的下方加入一段程式碼，先用「GetComponent<Grid>()」判斷 mapManager 裡是否已有 Grid 腳本，若找不到會傳回 null，此時我們就用「AddComponent<Grid>()」將腳本加上去，如下面程式碼。

```
public void OnEnable()
{
    mapManager = GameObject.Find("MapManager");
    if( !mapManager )
    {
        mapManager = new GameObject();
        mapManager.name = "MapManager";
    }

    if( mapManager.GetComponent<Grid>() == null )
    {
        mapManager.AddComponent<Grid>();
    }

    . . . . . .
```

現在，只要建立好 mapManager 就會自動顯示格線了，要調整時只要選取 mapManager 物件就可以調整參數，若要關閉格線就把「Display Grid」打勾拿掉便會隱藏格線。



其他的 Gizmos 指令

其他要有一些 Gizmos 的指令可以用，使用 Gizmos 繪製出來的物件不會出現在遊戲當中，只會顯示在編輯視窗內，可以依需求自行加入。

DrawBox : 傳入中心點 Center 與尺寸 Size 繪製出一個實心立方塊。

DrawFrustum : 繪製一個楔型物。

DrawGUITexture : 繪製一個貼圖。

DrawIcon : 繪製一個圖示。

DrawLine : 繪製線條。

DrawRay : 繪製射線。

DrawSphere : 繪製實心球體。

DrawWireBox : 繪製線條立方體。

DrawWireSphere : 繪製線條球體。

將資訊直接顯示在 SceneView 上

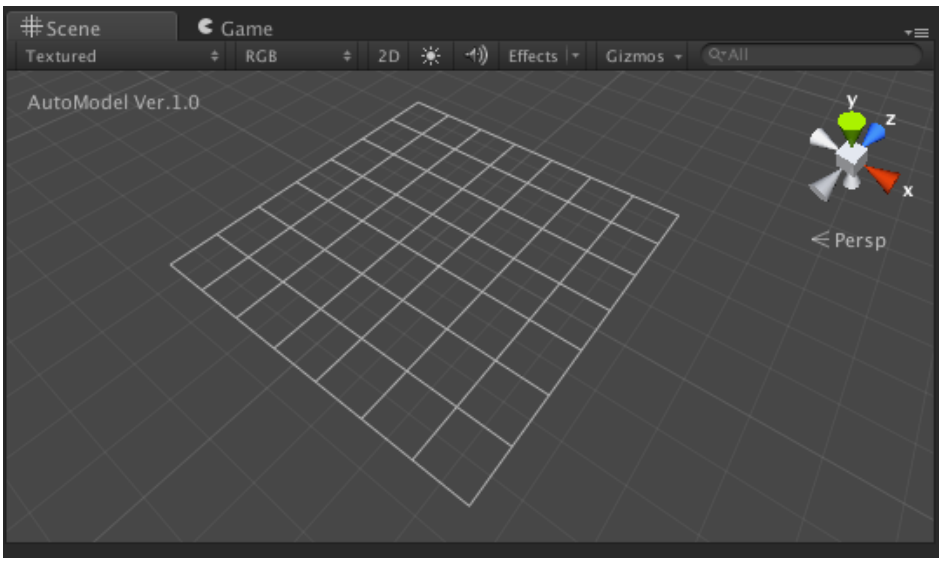
在 SceneView 顯示 GUI

我們除了將編輯器使用的 UI 繪製在 Inspector 或獨立的視窗外，也可以直接將 UI 繪製在 SceneView 上，但一般來說不建議這樣做，因為在編輯時會不小心按到畫面上的按鈕，但

若只是顯示一些資訊在 SceneView 上則非常的方便，我們來看一下如何顯示 UI 在 SceneView 上。

回到「ModelCreater.cs」腳本，找到「OnSceneGUI(SceneView sceneView)」方法，這個是我們自己建立的 SceneView 繪製的方法，要在 SceneView 繪製 UI 必須使用「Handles.BeginGUI(); . . . . . Handles.EndGUI();」，將 GUI 的指令包在兩個關鍵字裡面即可。

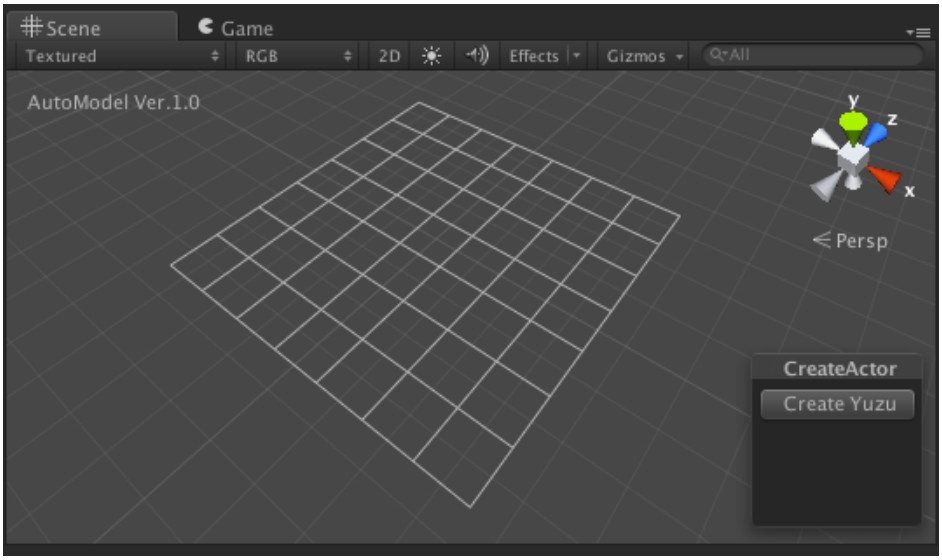
```
public void OnSceneGUI( SceneView sceneView )
{
    . . . . . .

    Handles.BeginGUI();

    GUI.Label( new Rect(10,10,400,20), "AutoModel Ver.1.0 ");

    Handles.EndGUI();

}
```



在 SceneView 裡顯示小視窗

也可以在 SceneView 裡面顯示小視窗，再將 GUI 放在視窗內，下面的範例是加入一個視窗並在裡面建立一個按鈕。

```
Handles.BeginGUI();

GUI.Label( new Rect(10,10,400,20), "AutoModel Ver.1.0 ");

GUILayout.Window(2, new Rect(Screen.width-110, Screen.height-130, 100, 100), (id)=> {
   // 視窗內部
   GUILayout.Button("Create Yuzu");
} , "CreateActor");

Handles.EndGUI();
```



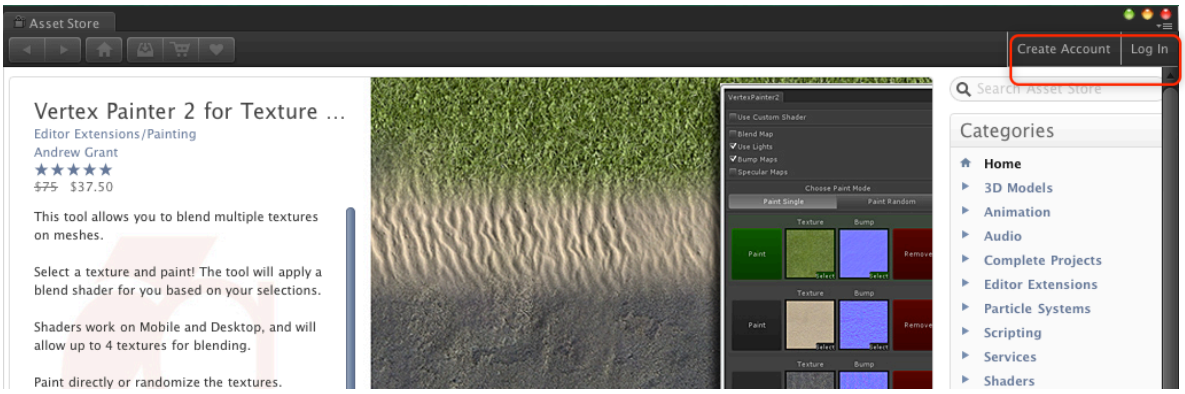
將製作好的模型或外掛上傳到 Asset Store

當我們開發好一個外掛或是建立一些 3D 模型或其他資源，包括外掛、腳本、模型、音效檔、貼圖等等，任何與 Unity 有關的資源，都可以放到 Asset Store 上販賣或是免費分享，現在來教如何將製作好的資源放到 Asset Store 上。

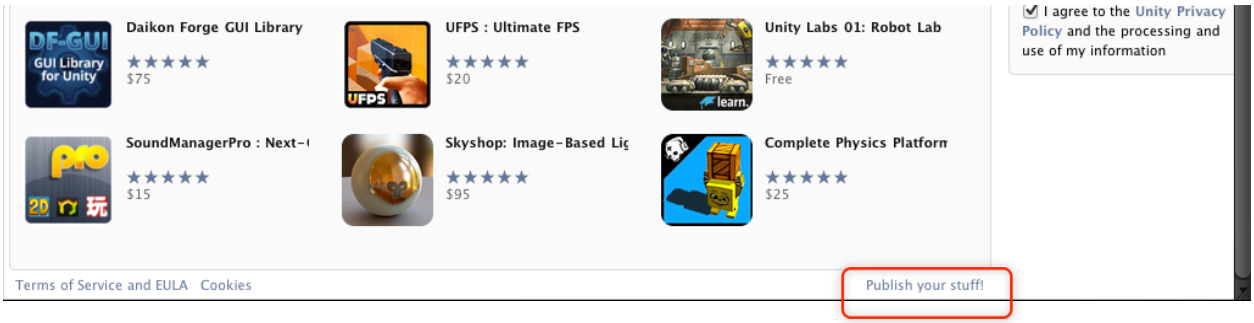
先開啟 Asset Store 視窗。

Window > Asset Store

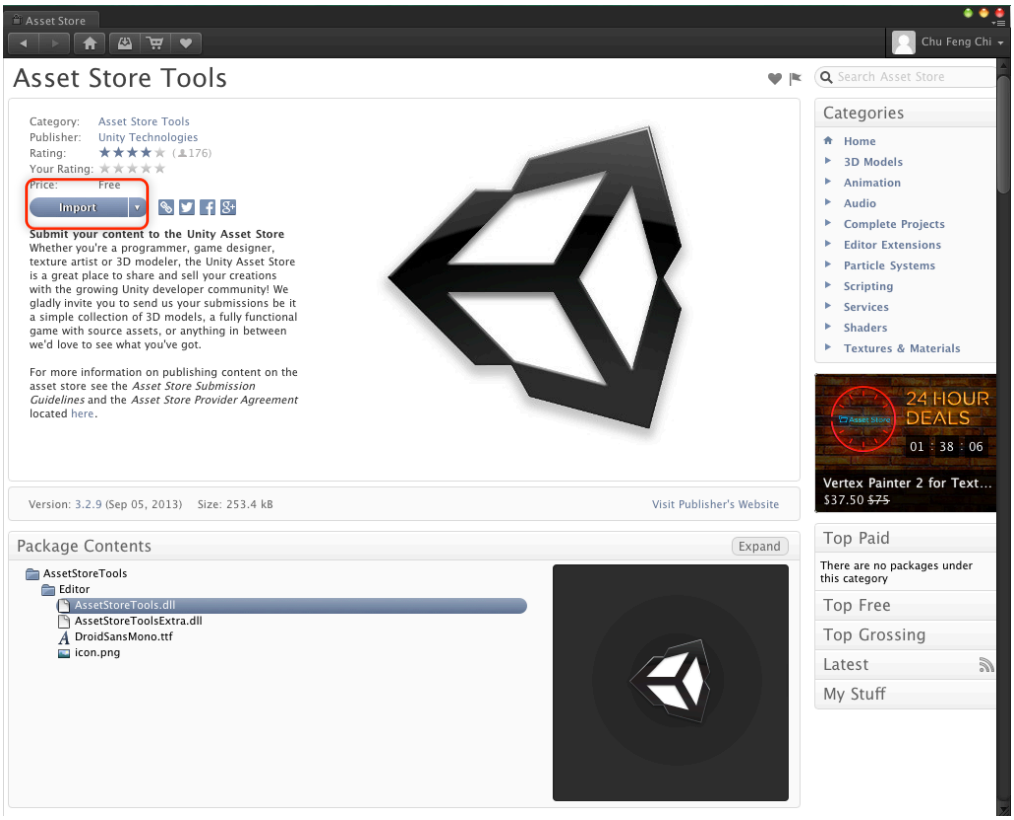
要將產品放到 Asset Store 必須先有會員，若已有會員請按下右上角的「Log In」並登入，若還沒有會員請選擇「Create Account」建立一個新會員，裡面的資料請正確填寫，以免將來產品販賣之後無法收到支票，加入會員後一樣要從 Log In 進行登入。

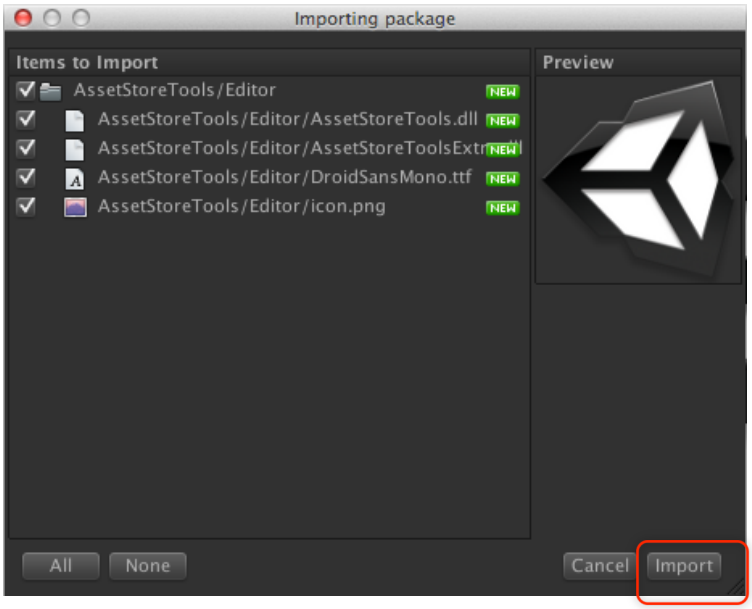


登入完成後，Asset Store 視窗右上會變成會員名字，接下來將 Asset Store 視窗捲到最下面，有一個「Public your stuff !」的字，請點選下去。

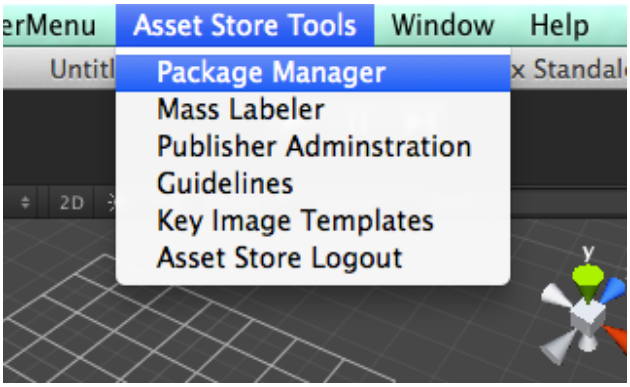


點選下去後畫面會變成一個可以下載的「Asset Store Tools」的外掛，請按下「Import」將這個外掛匯入進來。

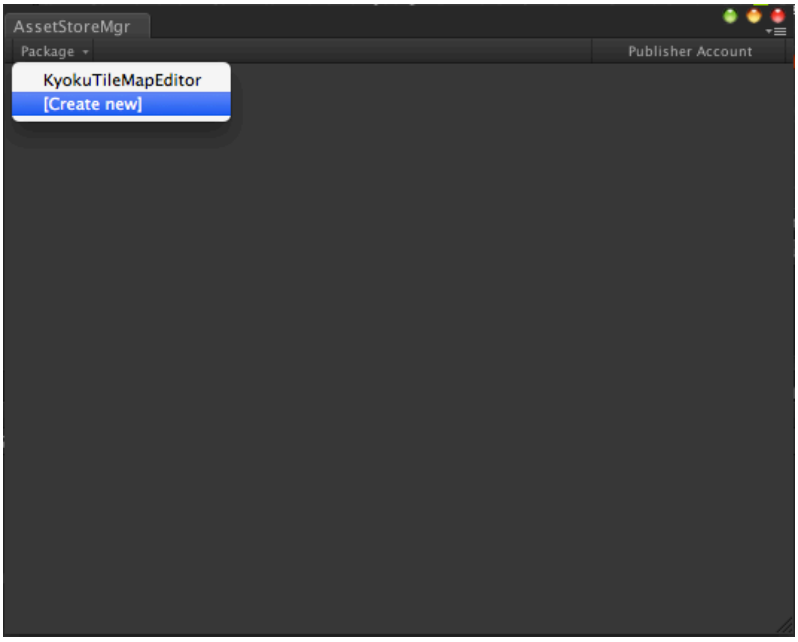

Asset Store
Chu Feng Chi
Asset Store Tools
Search Asset Store
Category: Asset Store Tools
Publisher: Unity Technologies
Rating: (176)
Your Rating:
Price: Free
Import
Submit your content to the Unity Asset Store
Whether you're a programmer, game designer, texture artist or 3D modeler, the Unity Asset Store is a great place to share and sell your creations with the growing Unity developer community! We gladly invite you to send us your submissions be it a simple collection of 3D models, a fully functional game with source assets, or anything in between we'd love to see what you've got.
For more information on publishing content on the asset store see the Asset Store Submission Guidelines and the Asset Store Provider Agreement located here.
Version: 3.2.9 (Sep 05, 2013) Size: 253.4 kB
Visit Publisher's Website
Package Contents
Expand
AssetStoreTools
Editor
AssetStoreTools.dll
AssetStoreToolsExtra.dll
DroidSansMono.ttf
icon.png
Categories
Home
3D Models
Animation
Audio
Complete Projects
Editor Extensions
Particle Systems
Scripting
Services
Shaders
Textures & Materials
24 HOUR DEALS
01 : 38 : 06
Vertex Painter 2 for Text...
$37.50 $75
Top Paid
There are no packages under this category
Top Free
Top Grossing
Latest
My Stuff


Importing package
Items to Import
Preview
AssetStoreTools/Editor
AssetStoreTools/Editor/AssetStoreTools.dll
AssetStoreTools/Editor/AssetStoreToolsExtra.dll
AssetStoreTools/Editor/DroidSansMono.ttf
AssetStoreTools/Editor/icon.png
NEW
All
None
Cancel
Import

匯入完成後，上方的選單會多一個「Asset Store Tools」，請執行「Package Manager」。



彈出 AssetStoreMgr 視窗後，從左上角選單按下「Package」，接下來選取「[Create new]」，若編輯到一半下次要繼續編輯，拉下來會顯示自己建立的所有上傳專案，無論是已上傳或編輯中，非常方便。

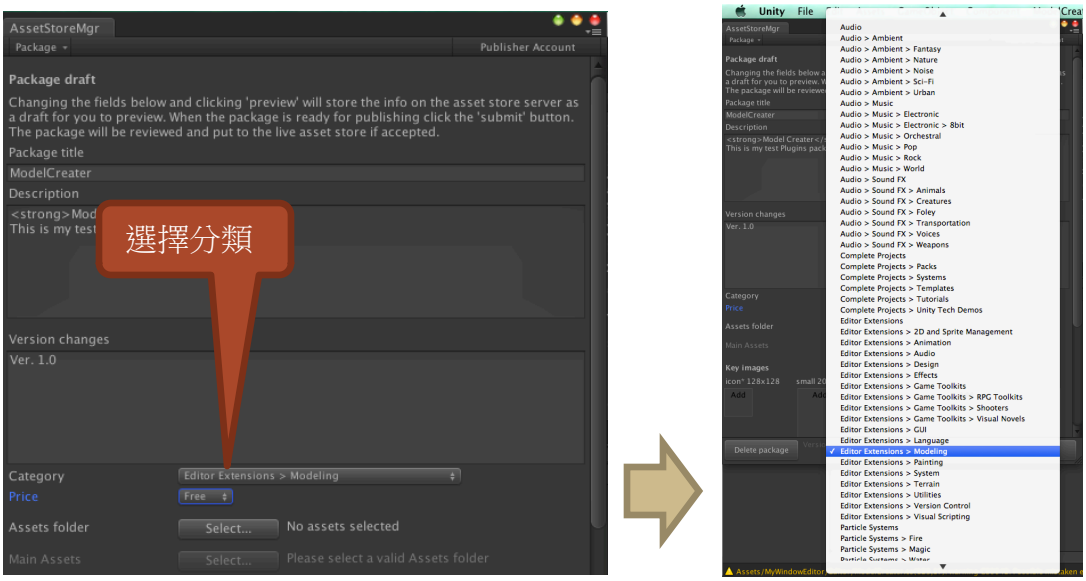


由於要填寫的內容頗多，這裡仔細的對所有欄位一一說明，首先是最上面的「Package title」，請填入你想要在 Asset Store 顯示的名稱，可以是你工具的名稱或是模型角色的名字等等。

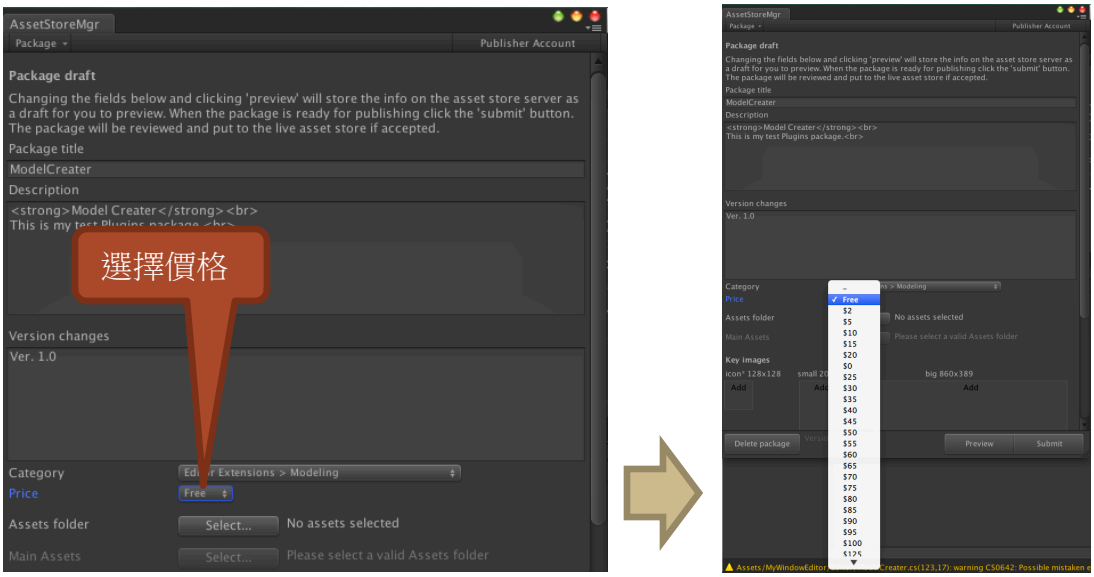
下面一點的 Description 請用 html 格式填入簡要說明文，但不是所有的 html 標簽都能用，只可以用「<a>
」，請注意，要換行請用
不要用</br>。

再下面的 Version changes 請填入版本更新內容資訊。

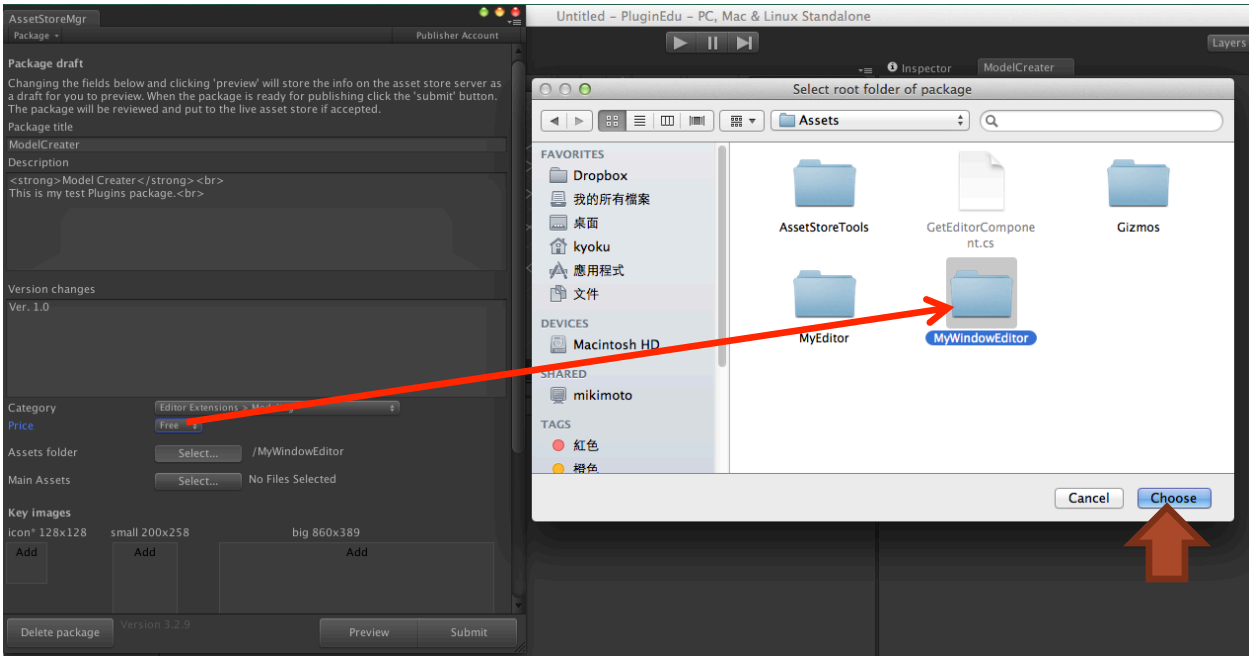
接下來下面的 Category 要選取的是顯示在 Asset Store 裡的分類，請依內容選擇適當的分類。



再下面的 Price 選擇售價。



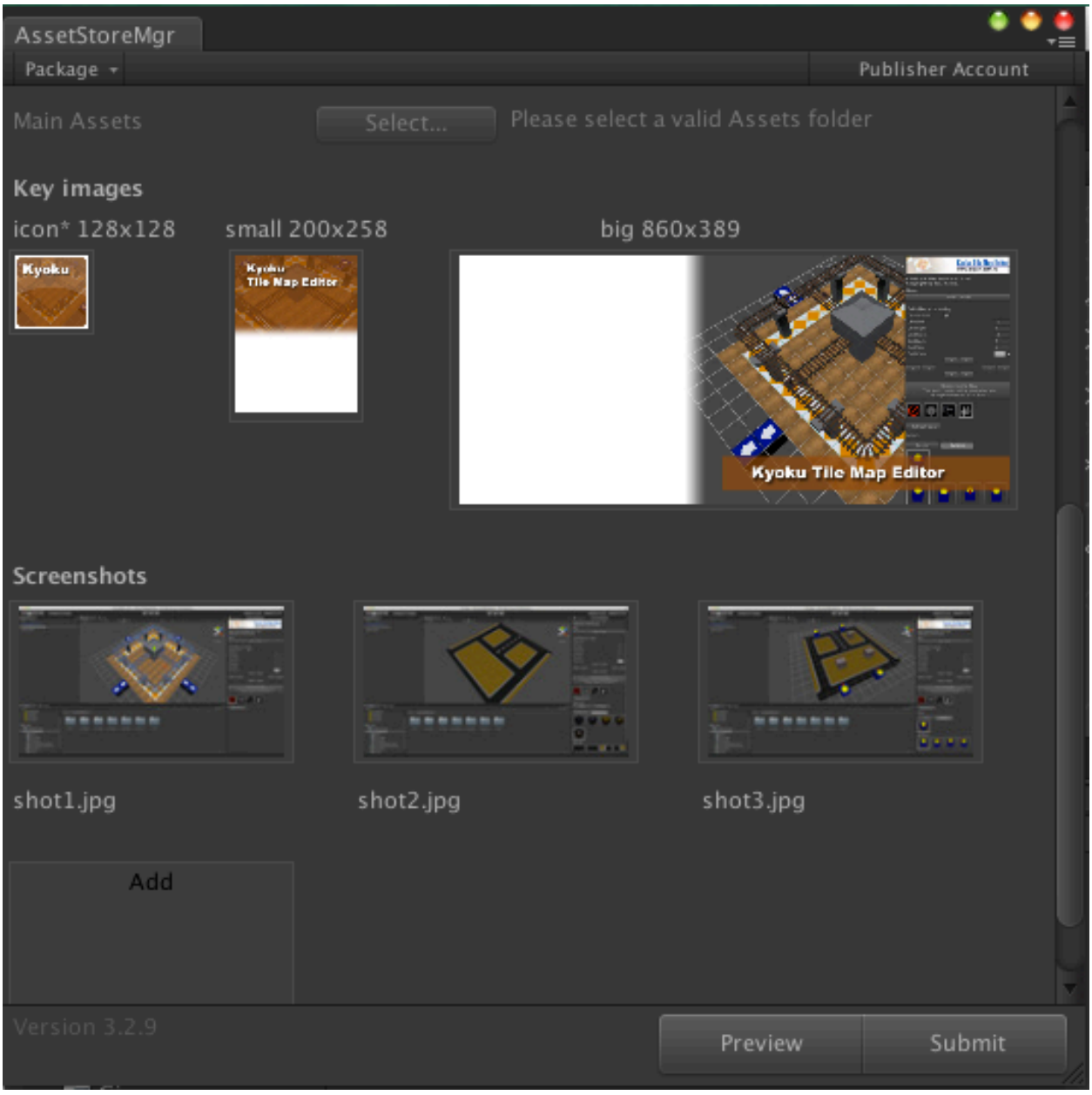
接下來按下「Assets Folder」的按鈕[　Select ...　]，在彈出的檔案對話盒選擇想要上傳的資料夾，然後按下[Choose]。



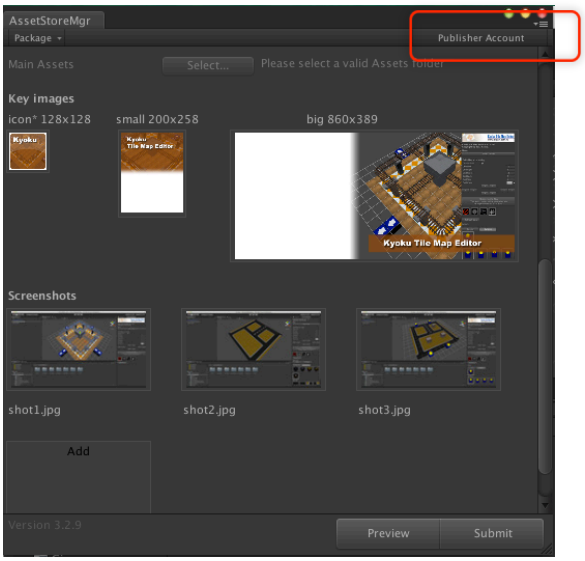
Main Assets 是當你有多個檔案上傳時，要以哪個做為主要的檔案，這裡的檔案沒有分主要次要所以不用選擇。

再下面的 Key images 請依指示的大小準備好專案的圖片放進去，在「small 200x258」的圖請在下方留 50%的白，而「big 860x389」請在左邊留 50%的白，這些留白處是到時會顯示文字的部份，留以以避免文字看不清楚。

再下方的 Screenshots 可以放入幾張使用抓圖，大小沒有限制，但高度最好不要大於 800 以免一般的筆電看不到全圖。

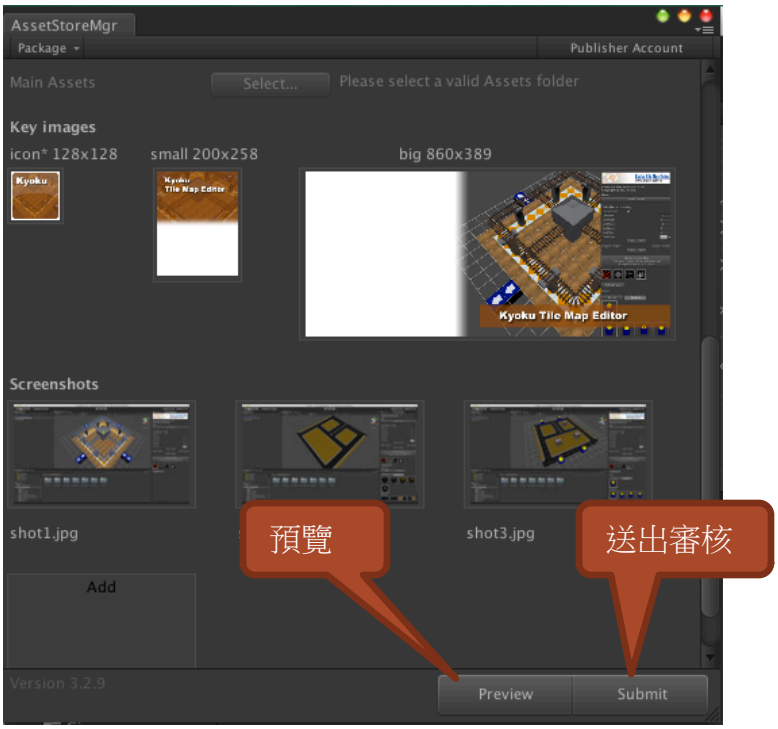


接下來按下視窗右上方的「Publisher Account」。



在彈出的視窗中填入開發者的介紹和官網網址等資訊，填寫完成後按下 [Save]。

待全部填寫完成後可以按下「Preview」進行預覽，若沒問題就按下「Submit」送出，接下來 Unity 公司會進行審核，審核若有問題會以郵件通知修改，待全部通過就可以上架了。



到這裡其實會有個問題，就是我們有用到「Gizmos」資料夾，但上傳 Asset Store 時不允許選取多個資料夾，因此我們無法上傳這個資料夾，因此到時會發生讀不到 icon 的窘境，這個問題要如何克服呢!? 依一般的解決方法是直接上傳 Asset 這個根目錄，如此便可以含我們開發的外掛以及 Gizmos 資料夾，而「Asset Store Tools」的內容 Unity 公司審核時會自行幫我們去掉其內容。

參考書目

Unity 3D + Photon 線上遊戲開發入門

版權聲名

作者：紀曲峰(M.K.)

Email：mkkii@ms49.hinet.net

HomePage：http://www.digiart.com.tw

※ 若有任何問題需要 Email 聯絡我，請在主旨註明 Photon 問題、Unity3D 問題、Maya 問題等，若只填入打招呼字句會被我直接移到垃圾郵件箱。