SpaceShooter2D 单机版实作范例

大风起兮云飞扬
威加海内兮归故乡
安得猛士兮守四方

《大风歌》

本文将以 Unit3D 的 3D 场景制作一个 2D 太空射击游戏，透过实作让读者更能了解如何以 Unity3D 开发游戏。除了与美术有关的模型与素材外，这个游戏项目将会从无到有，一步一步带领读者实际操作，完成太空射击游戏。

接着我们就开始进行，让大家体会开发游戏的乐趣吧！

1-1 建立游戏项目基础架构

1-2 玩家角色

1-3 敌人系统

1-4 碰撞处理

完整教程请至 http://developer.arcalet.com 进行下载。

1-5 游戏机制与 GUI 设计

完整的游戏必须有完整的游戏机制，本文将以 Unity3D 的 GUI 接口设计玩家与游戏对话的画面，另外也将为游戏加上玩家角色的生命值与得分机制，让整个游戏变得更好玩。

显示游戏状态信息

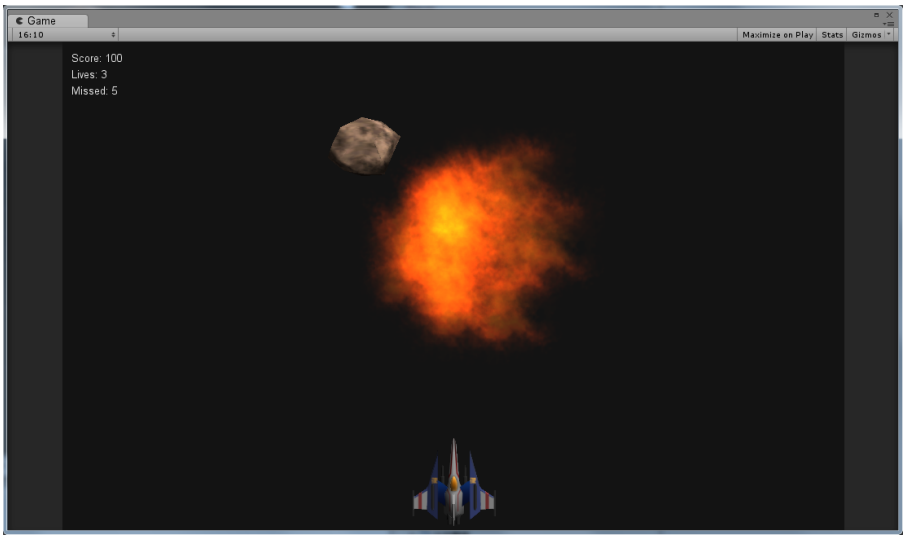
游戏进行中常会看到画面的某个角落都会显示游戏状态信息，玩家可以藉由这些信息了解自己所处的环境状态及拥有的资源等讯息。现在我们使用 GUI 功能来显示「分数」、「生命值」及「未击中的殒石数」等信息，这三项信息记录则分别存放在整数变量 `Score`、`Lives`、`Missed` 中。请注意，为了让其它脚本程序可以存取这三个变量，请务必将它们定义为公开的静态变量，稍候在其它 GameObject 脚本程序中随着游戏进行实时更新变量内容值，游戏画面就会看到最实时的状态信息。

以下是加上显示游戏状态信息的程序代码后的 Player 脚本：

```
// Player.cs
using UnityEngine;
using System.Collections;

public class Player : MonoBehaviour {

        ... (略) ...

    public static int Score = 0;
    public static int Lives = 3;
    public static int Missed = 0;

        ... (略) ...

    void OnGUI() {
        GUI.Label(new Rect(10, 10, 150, 20),"Score: " + Player.Score.ToString());
        GUI.Label(new Rect(10, 30, 60, 20),"Lives: " + Player.Lives.ToString());
        GUI.Label(new Rect(10, 50, 120, 20),"Missed: " +Player.Missed.ToString());
    }
```

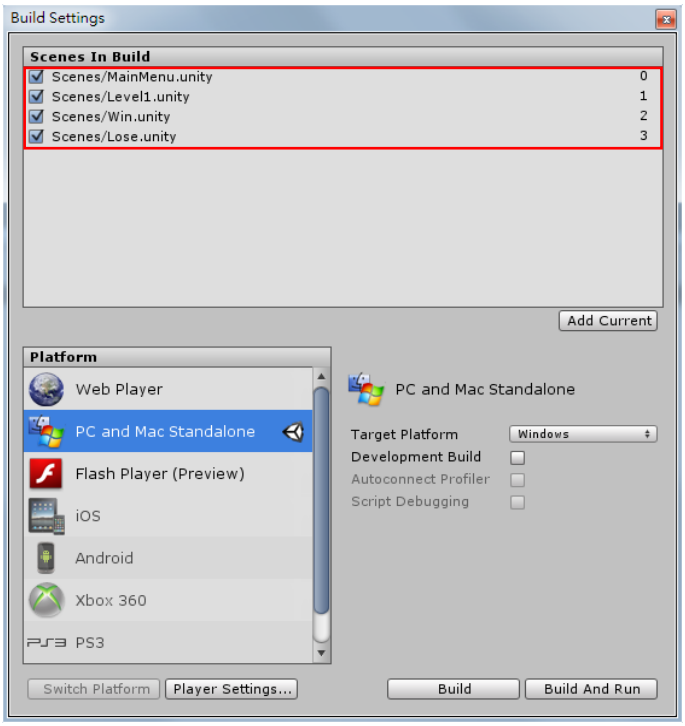
```
        ... (略) ...
}
```

完成后执行游戏，画面的左上角出现游戏状态信息了，不过因为游戏机制尚未建立，所以目前状态信息都不会有任何的变化，稍后我们会开始建立游戏规则，让相关的状态变量能够随着游戏的进度而改变。



串连场景

到目前为止，我们所制作的游戏都是在 Level1 这个场景里，事实上早在项目建立之初，我们就计划要以 MainMenu 为游戏的第一个场景，现在再确认一下，点选主选单→「File」→「Build Settings」，看到「Scene In Build」内的场景顺序应该是这样子：



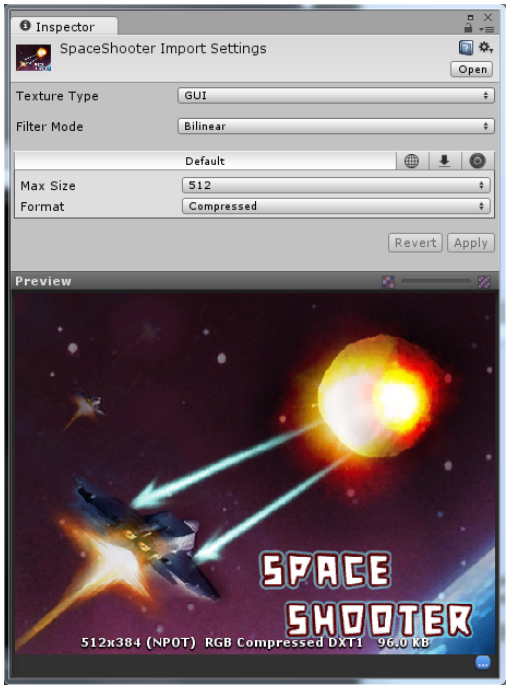
现在除了 Level1 以外，其它三个场景都是空的，由于这三个场景都只是简单的显示画面，所以我们以 Unity3D 的 GUI 接口来设计场景。

一、MainMenu 场景

MainMenu 有点像是一般游戏的大厅系统，玩家可以在此了解游戏的玩法，并选择开始游戏。由于 MainMenu 场景除了 Main Camera 以外没有其它的 GameObject，所以我们将 GUI 的脚本程序附加到 Main Camera 中。

场景画面我们可以使用一整张背景图呈现，这张图请以绘图软件事先制作，或是使用已存放在 spacebag 里的图档(名称为 SpaceShooter)。为了让 GUI 程序代码可以存取这个图档，我们在脚本定义一个形态为 Texture 的公共变量，然后从 Project 窗口中的 spacebag 数据匣里把「SpaceShooter」图档拖曳到这个公共变量，好让脚本程序可以呼叫 GUI.DrawTexture()来显示这张背景图。

为了优化「SpaceShooter」图档，我们必须把它的 Texture Type 属性设定为「GUI」，同时设定 Max Size 为「512」：

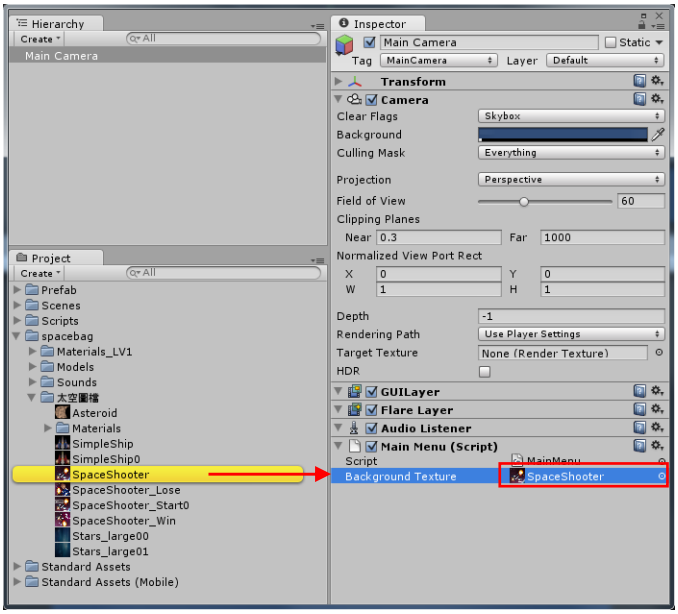


接下来新增一个名为 MainMenu 的脚本，然后把它拖曳到 MainCamera (请确定目前场景是 MainMenu)，脚本内容如下：

```
// MainMenu.cs
using UnityEngine;
using System.Collections;

public class MainMenu: MonoBehaviour {
    public Texture backgroundTexture;
    private string instructionText="Instructions:\nPress Left and Right Arrows to move.\nPress Spacebar to
fire.";

    void OnGUI() {
        GUI.DrawTexture(new Rect(0,0,Screen.width,Screen.height),backgroundTexture);
        GUI.Label(new Rect(10,10,250,200),instructionText);

        if (Input.anyKeyDown)
            Application.LoadLevel(1);
    }
}
```

按键后进入 Level1 场景，游戏开始！

先前提到的背景图档公共变量为 backgroundTexture，现在指定它的初值：

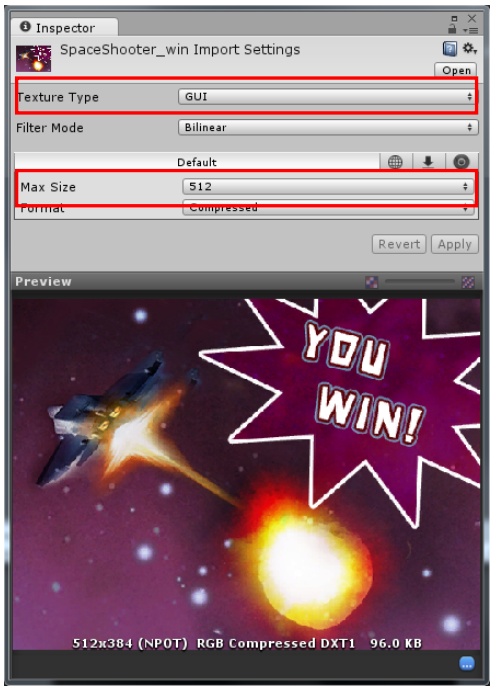


完成后执行游戏，以下是第一个出现的画面，也就是 MainMenu 场景的画面：

二、Win 场景

Win 在项目中的场景顺序是 2，稍后我们会在游戏中加入一些游戏机制，当玩家击中陨石获得足够分数后游戏便会结束，然后将场景转换到 Win，现在我们要制作的 Win 场景也是以 GUI 在屏幕上呈现「You Win!!」讯息，画面则是事先做好的图档，制做方法和前面的 MainMenu 一样，同样的，我们已经为各位事先准备好图档，就存放在 spacebag 数据匣中，名称为「SpaceShooter_Win」。

和前面的 MainMenu 场景一样，在使用图档之前，必须设定 Texture Type 属性为「GUI」，同时设定 Max Size 为「512」：



现在我们把编辑中的场景切换到 Win 场景，然后新增一个名为 Win 的脚本，再把脚本拖曳到 Main Camera 里。脚本内有个公共变量 `backgroundTexture`，也要记得把图档「SpaceShooter_Win」指定为它的初值。以下是完整的 Win.cs 脚本内容：

```
// Win.cs
using UnityEngine;
using System.Collections;

public class Win : MonoBehaviour
{
    public Texture backgroundTexture;
    private string instructionText = "Instructions : Any Key Put Down to Start";

    void OnGUI() {

        GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height),backgroundTexture);
        GUI.Label(new Rect(230, 270, 250, 200), instructionText);

        if (Input.anyKeyDown) {

            Player.Score = 0;
            Player.Lives = 3;
            Player.Missed = 0;

            Application.LoadLevel(1);
        }

    }
}
```

进入 Level1 之前先重设游戏状态变量

进入 Level1 场景，游戏重新开始！

脚本中以 `GUI.Label()`显示一段文字，告诉玩家按下任意键可以再玩一次，任意键则是以

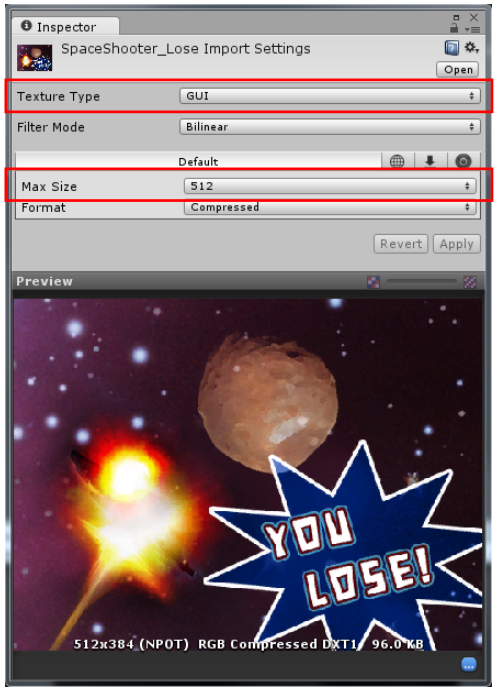
```
if (Input.anyKeyDown)
```

判断是否按下。

当玩家按下任意键后，接着呼叫 `Application.LoadLevel(1)`进入 Level1 场景，游戏重新开始，不过在进入 Level1 之前，别忘了重设 Player 的三个游戏状态变量 `Score`、`Lives`、`Missed`。

三、Lose 场景

Lose 场景的制作和 Win 场景的做法几乎一样，背景图为「SpaceShooter_Lose」，脚本内容也和 Win 脚本很像，别忘了指定公共变量 `backgroundTexture` 的初值为「SpaceShooter_Lose」，使用图档之前，也请设定 Texture Type 属性为「GUI」，Max Size 属性为「512」：



以下是 Lose 脚本的内容：

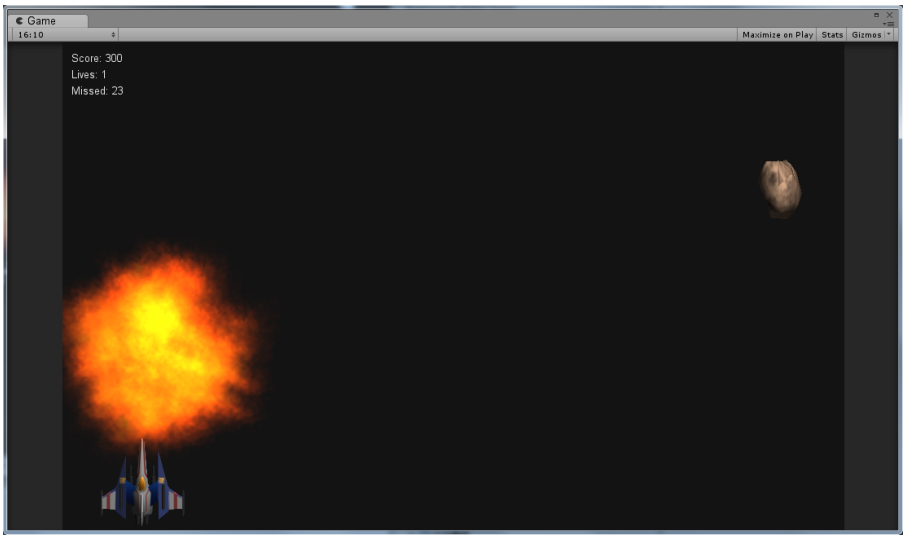
```
// Lose.cs
using UnityEngine;
using System.Collections;

public class Lose : MonoBehaviour
{
    public Texture backgroundTexture;
    private string instructionText = "Instructions : Any Key Put Down to Start";

    void OnGUI() {

        GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height),backgroundTexture);
        GUI.Label(new Rect(230, 270, 250, 200), instructionText);

        if (Input.anyKeyDown) {

            Player.Score = 0;
            Player.Lives = 3;
            Player.Missed = 0;                  } 进入 Level1 之前先重设游戏状态变量

            Application.LoadLevel(1);           } 进入 Level1 场景，游戏重新开始！
        }

    }
}
```

玩家得分

最简单的得分机制是炮弹击中陨石时就得到固定的分数，当然我们也可以设计复杂一些的得分规则，例如得分的高低与击中陨石时两者之间的距离成正比，距离越远被击落，所得到的分数就越高，反之则越低；或是得分和陨石的速度有关，击中速度越快的陨石也可以获得较高的分数，总之，得分机制并非一成不变，开发者可以任意发挥创意建立自己的得分机制。

此处我们采用简单的得分机制，每击中一次就得 100 分，累加分数的程序代码可以放在 Projectile 脚本里的碰撞处理函式 `OnTriggerEnter()`中。另外要特别说明的是，由于 Player 脚本中的变量「`Score`」被定义为 `public static`，所以虽然得分的程序代码是在 Projectile 脚本中执行，但还是可以直接以 `Player.Score` 跨脚本存取，以下是我们修改过的 Projectile.cs 的程序代码内容：

```
//Projectile.cs
using UnityEngine;
using System.Collections;

public class Projectile : MonoBehaviour {

        ... (略) ...

    void OnTriggerEnter(Collider otherObject) {
       if(otherObject.tag == "enemy") {
            Instantiate(ExplosionPrefab, enemy.transform.position, enemy.transform.rotation);
            enemy.SetPositionAndSpeed();
            Destroy(gameObject);
            Player.Score+=100;
            if (Player.Score > 1000) Application.LoadLevel(2);
        }
    }
}
```



玩家生命值

玩家生命值存放在变数 `Player.Lives` 里(也就是 Player 脚本中的 `Score` 变量)，目前默认初值是 3，当陨石击中太空战机时就把 `Player.Lives` 减 1，这个指令可以放在处理陨石和太空战机碰撞的 `OnTriggerEnter()`函式里，以下是修改过的 Player 脚本：

```
// Player.cs
using UnityEngine;
using System.Collections;

public class Player : MonoBehaviour {

        ... (略) ...

    void OnTriggerEnter(Collider otherObject) {
        if (otherObject.tag == "enemy" && state == State.Playing) {
            Player.Lives--;
            Instantiate(ExplosionPrefab, enemy.transform.position, enemy.transform.rotation);
            enemy.SetPositionAndSpeed();
            StartCoroutine("DestroyShip");
        }
    }

        ... (略) ...
}
```

现在太空战机被陨石击中后生命值就会减 1，一直减到生命值为 0 时游戏就结束了，也就是将场景切换到 Lose 场景。

现在我们思考一下要在那里判断生命值是否为 0 呢？当太空战机被陨石击中，我们会先产生爆炸，然后飞机消失一下子后重生，重生时还要有几秒的无敌状态，这些动作都在 Player 脚本的 `DestroyShip()`函式中进行，如果玩家生命值已经用完了当然就不需要重生，应该马上出现失败的画面。因此我们决定在太空战机消失后再度出现之前判断游戏是否结束，以下是加上这段判断式 Player 脚本：

```
// Player.cs
using UnityEngine;
using System.Collections;

public class Player : MonoBehaviour {

        ... (略) ...

    IEnumerator DestroyShip() {
        state=State.Explosion;

        gameObject.renderer.enabled = false;
        yield return new WaitForSeconds(shipInvisibleTime);
        transform.position = new Vector3(0.0f, transform.position.y, transform.position.z);

        if (Player.Lives>0) {
            gameObject.renderer.enabled = true;

            state=State.Invincible;
            while (blinkCount < numberOfTimesToBlink) {
                gameObject.renderer.enabled = !gameObject.renderer.enabled;

                if (gameObject.renderer.enabled == true)
                    blinkCount++;
```

```
                yield return new WaitForSeconds(blinkRate);
            }
            blinkCount=0;
            state=State.Playing;
        }
        else {
            Application.LoadLevel(3);
        }
    }

        ... (略) ...

}
```

果然，被陨石击中三次后生命值减为 0，接着游戏就结束并进入了 Lose 场景。

失手次数

所谓失手就是陨石出现但是没被玩家击落，「失手次数」记录在变量 `Player.Missed` (也就是 Player 脚本中的 `Missed` 变量)，是继「生命值」与「得分」后的另一种游戏状态信息。记录「失手次数」可以在处里陨石飞离游戏画面的程序代码之后，陨石既然能够安然飞离，也就是没被玩家击落，所以在此处将 `Player.Missed` 变数加 1：

```
// Enemy.cs
using UnityEngine;
using System.Collections;

public class Enemy : MonoBehaviour {

        ... (略) ...

    void Update () {
        float outToMove = currentSpeed * Time.deltaTime;
        transform.Translate(Vector3.down * outToMove);

        if (transform.position.y <=-5) {
            SetPositionAndSpeed();
            Player.Missed++;
        }
    }

        ... (略) ...

}
```

1-6 细部调整

完整教程请至 http://developer.arcalet.com 进行下载。

教程从「单机模式」改编成「Online」多人实时都有详细的教程，千万别错过。