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

1-5 游戏机制与 GUI 设计

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

1-6 细部调整

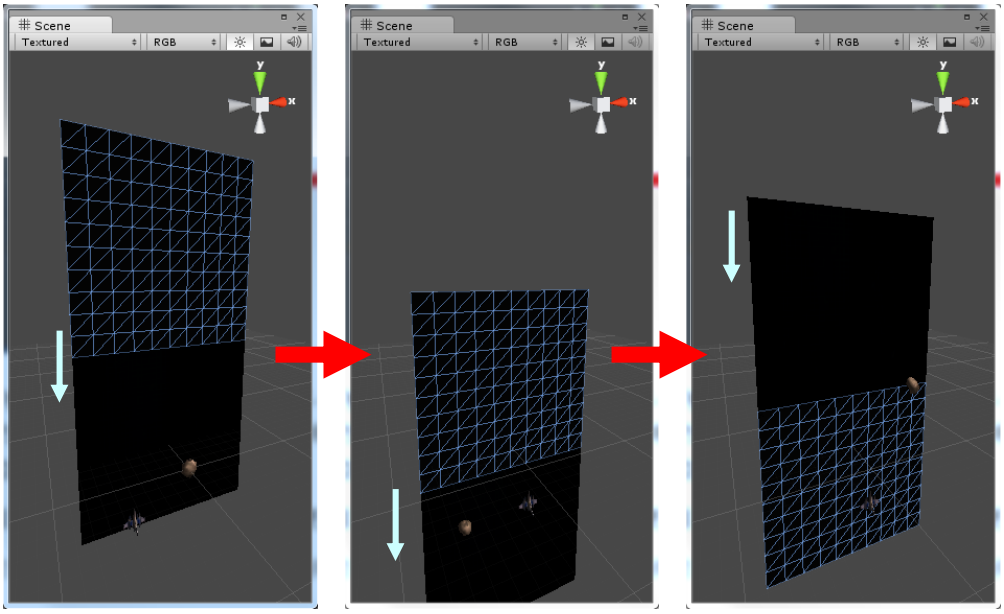
到目前为止游戏制作已进入尾声，但最后我们还是要做一些细部的调整，例如星空背景、动态关卡难度等，当然，细部调整不只这些，如果大家有兴趣可以各自发挥创意，让整个游戏能够更臻完美。

动态星空背景

游戏既然取名为「Space Shooter」，所以我们要让游戏背景呈现星空的样子，而且不能是静态的星空，必须随着飞机往前飞而向后移动。

为了实现这样的效果，我们计划制作一个平面物体，在表面贴上星空图，然后把它放在游戏场景的一侧，平面则与太空战机行进的方向平行。当游戏进行时，这个平面物体会缓缓往下移动，制造出太空战机向前飞行，远处的星星慢慢的往后移动的感觉。

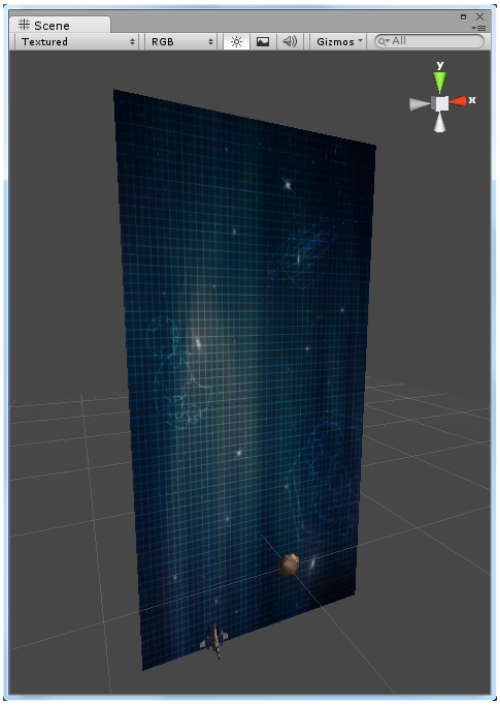
此外我们还要有无限延伸的星空背景，所以使用两张背景图来实现：



两张背景图缓缓往下移动　　飞机快到上方这张背景了　　下面的背景图换到上方

这项技巧是将两张星空图连接在一起并且同步往下方移动，当下方的星空图完全脱离游戏画面后，上方的图位置不变，下方的图移到上方，两张图仍旧连接在一起，然后继续同步缓缓向下移动，就这样不断地移动与交错换图，视觉上就会有无限延长的星空背景了。

了解星空背景运作的方法后，现在我们回到 Level1 场景，依照下列四个步骤建立两张连接起来的星空背景平面：



(1) 在场景中新增一个「Plane」物体，然后更名为 Stars1。

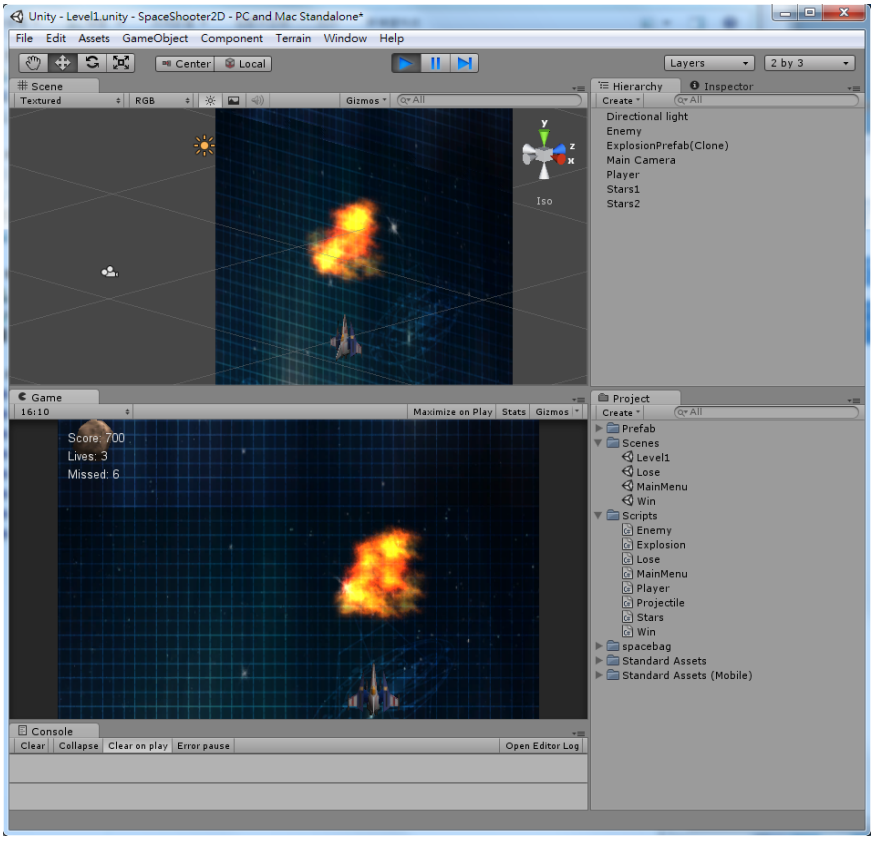
(2) 设定「Stars1」的 Position、Rotation、Scale 分别为 (0,1,3.5)、(270,0,0)、(1.6,1.6,1.6)。

(3) 以键盘快捷键 Ctrl-D 复制「Stars1」，复制后更名为 Stars2，修改「Stars2」的 Position 为(0, 17, 3.5)，也就是把「Stars2」接到「Stars1」的上方。

(4) 从 Spacebag 中将预先做好的星空图「Stars_large00」拖曳到「Stars1」，「Stars_large01」拖曳到「Stars2」。

紧接着新增一个名为 Stars 的脚本，这个脚本程序同时可用来控制「Stars1」与「Stars2」，以下是脚本内容，请将此脚本分别拖曳到「Stars1」与「Stars2」，然后分别到「Stars1」与「Stars2」的 Inspector 窗口中把脚本的公共变量「Speed」设定为 1，星空背景效果就大功告成了：

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

public class Stars : MonoBehaviour {
    public float Speed;

    void Update () {
        float outToMove = Speed * Time.deltaTime;
        transform.Translate(Vector3.down * outToMove, Space.World);

        if (transform.position.y < -12) {
            transform.position =
                        new Vector3(transform.position.x,19.5f,transform.position.z);
        }
    }
}
```



关卡难度调整

游戏玩久了之后，玩家就会逐渐习惯一成不变的敌人行为，所以许多游戏都会有关卡越来越难的设计，我们的太空射击游戏虽然还没有多个关卡，但还是可以发挥一点巧思，在同一个关卡中让游戏越来越难。譬如我们可以让陨石大小不再是固定不变，或是随着时间过去，新陨石的速度也会越来越快。

陨石的大小变化主要是为了让游戏更加有趣，每次出现的陨石有大有小，玩家在视觉上比较不会枯燥乏味；而陨石的速度越来越快则会让玩家的反应时间变短，新陨石一出现，玩家得快速移动太空飞机来发射炮弹或避开陨石，操作的难度也就会越来越高了。

一、陨石速度

陨石速度是以随机数决定的，程序代码如下：

```
currentSpeed = Random.Range(MinSpeed,MaxSpeed);
```

所以只要在陨石被炮弹摧毁之后把 MinSpeed 与 MaxSpeed 的数值调高，新陨石就有机会以更高的速度出现，因此我们修改 Projectile.cs 程序，每次陨石被炮弹击中后就把 MinSpeed 增加 1.2，MaxSpeed 增加 1.6：

```
//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.MinSpeed += 1.2f;
            enemy.MaxSpeed += 1.6f;          } 加大新陨石可能的速度值区间
            enemy.SetPositionAndSpeed();
            Destroy(gameObject);
            Player.Score+=100;
        }
    }
}
```

二、陨石大小

原本我们并没有处理陨石变大变小，它固定的 Scale 就是(1.3, 1.3, 1.3)，为了让它可以以随机数改变大小，我们要先设置两个变量 MinScale、MaxScale，做为以随机数生成陨石 Scale 值的上下限，另外还要三个变量 CurrentScaleX、CurrentScaleY、CurrentScaleZ，配合 MinScale 与 MaxScale 就可以决定新陨石的 Scale 值：

```
CurrentScaleX = Random.Range(MinScale,MaxScale);
CurrentScaleY = Random.Range(MinScale,MaxScale);
CurrentScaleZ = Random.Range(MinScale,MaxScale);
```

把 *x,y,z* 三个方向的 scale 值分开以随机数决定还有个好处，这样子陨石外观形状也会有变化，视觉效果更佳。有了 scale 值之后，再加上这行指令就可以改变陨石外观尺寸了：

```
transform.localScale = new Vector3(currentScaleX,currentScaleY,currentScaleZ);
```

综合以上内容，修改过的 Projectile 脚本如下：

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

public class Enemy : MonoBehaviour {
    public float MinSpeed;
```

```
    public float MaxSpeed;
    private float MinScale=0.8f;
    private float MaxScale=2f;
    private float currentScaleX;
    private float currentScaleY;
    private float currentScaleZ;

    ... (略) ...

    public void SetPositionAndSpeed() {
        currentScaleX = Random.Range(MinScale,MaxScale);
        currentScaleY = Random.Range(MinScale,MaxScale);
        currentScaleZ = Random.Range(MinScale,MaxScale);

        currentSpeed = Random.Range(MinSpeed,MaxSpeed);
        x = Random.Range(-7.0f, 7.0f);
        y = 7.0f;
        z = 0.0f;

        transform.position = new Vector3(x, y, z);
        transform.localScale = new Vector3(currentScaleX,currentScaleY,currentScaleZ);
    }
}
```

陨石可能的大小值区间

随机数生成新陨石的尺寸

三、陨石旋转

为了让陨石旋转，我们先在 Enemy 脚本中设置几个变量：

(1) `MinRotateSpeed`、`MaxRotateSpeed`：做为以随机数生成陨石旋转速度的上下限。

(2) `currentRotationSpeed`：用来决定目前陨石的旋转速度，变量值是在 `SetPositionAndSpeed()` 函式中以随机数决定。

陨石旋转控制必须在 `Update()`函式中进行，为了不让陨石的旋转过于复杂，所以让它只在 *x* 方向旋转即可，程序代码如下：

```
float currentRotation = currentRotationSpeed * Time.deltaTime;
transform.Rotate(new Vector3(-1,0,0) * currentRotation);
```

此处有个地方要特别注意，在加入控制旋转的程序代码之后，等于在 `Update()`中同时执行旋转和移动的指令，但是因为陨石是自转，参考的是相对坐标，移动却是依照 3D 世界坐标移动，所以控制移动的 `transform.Translate()`函式必须指定坐标系统为世界坐标，也就是要多加一个参数「`Space.World`」，否则陨石会变成绕圈圈移动：

```
transform.Translate(Vector3.down * outToMove, Space.World);
```

综合以上内容，修改过的 Enemy 脚本如下：

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

public class Enemy : MonoBehaviour {
    public float MinSpeed;
    public float MaxSpeed;
    private float currentRotationSpeed;
    private float MinRotateSpeed=60.0f;
    private float MaxRotateSpeed=120.0f;
        ... (略) ...
```

陨石可能的旋转速度值区间

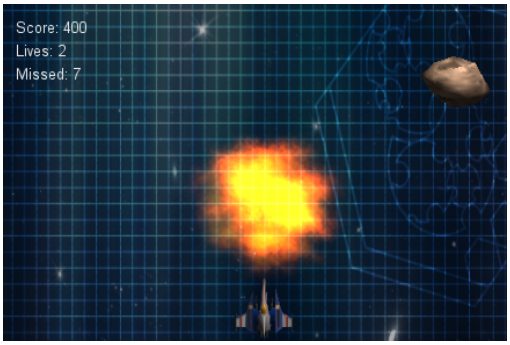
```
    void Update () {
        float currentRotate = currentRotationSpeed * Time.deltaTime;
        transform.Rotate(new Vector3(-1,0,0) * currentRotate);
        float outToMove = currentSpeed * Time.deltaTime;
        transform.Translate(Vector3.down * outToMove, Space.World);
        ... (略) ...
    }

    public void SetPositionAndSpeed() {
        currentRotationSpeed = Random.Range(MinRotateSpeed, MaxRotateSpeed);
        ... (略) ...
    }
}
```

隕石旋转

新隕石的旋转速度值

完工

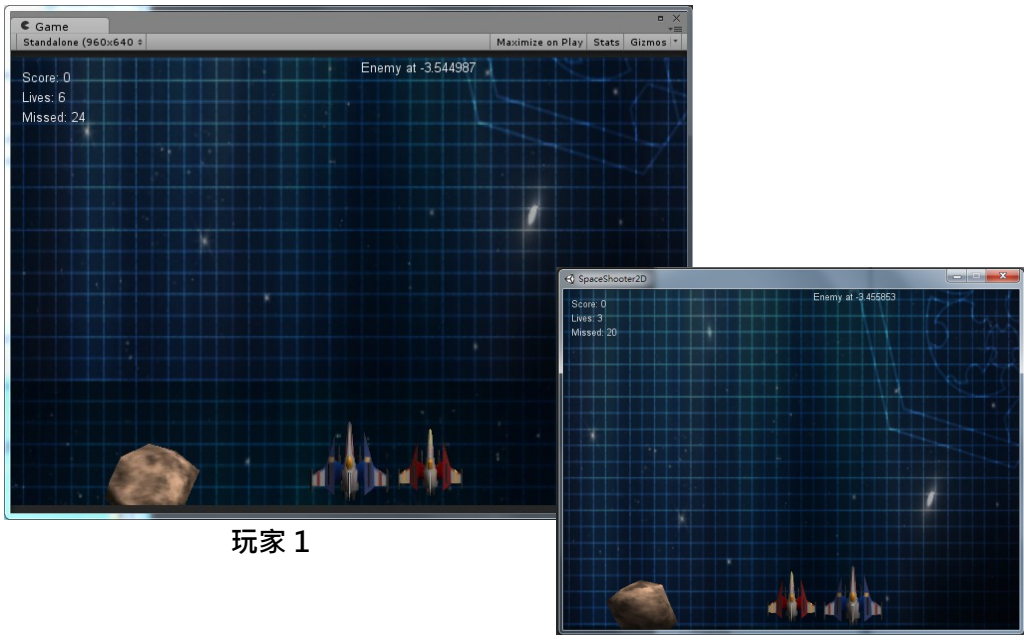
单机版 Space Shooter 终于完工了，虽然这是我们一步一步带领大家完成的游戏，但是藉由这样子抛砖引玉，相信有创意的读者一定能够发挥所学，青出于蓝而胜于蓝，让我们期待更多有趣好玩的游戏问世吧！



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。

从无到有－实作运行结果



玩家 1

玩家 2