# SpaceShooter2D Online 版实作范例



## 八风吹不动，一屁打过江

《佛印禅师公案》

射击游戏是实时要求非常高的游戏，为了让程序设计师很快能学会设计在线游戏程序，所以我们选择这个简单的游戏做为范例。

依续着上一章的单机版太空射击游戏，在改造为多人联机后，比较单机与在线的版本，您会发现用 arcalet 设计在线游戏真的很简单，原本如如不动的远程场景对象，在一弹指间竟然随着远程玩家的动作同步动了起来。

## 1-1 Online 基本概念

## 1-2 arcalet 程序设计基础

## 1-3 登入游戏

## 1-4 处理讯息

## 1-5 玩家操控

玩家操控的脚本程序是 Player GameObject 中的 player.cs，现在我们进行多人联机版的改造后，为了让 player.cs 中的程序代码必须能够使用与存取 AGCC 对象，因此我们先在 Awake()事件中取得 AGCC 对象变量：

```
//player.cs
    ... (略) ...
private AGCC agcc=null;
void Awake()
{
    agcc=(AGCC)FindObjectOfType(typeof(AGCC));
}
    ... (略) ...
```

### 太空战机的移动(本地操控)

要学习在线版太空战机的操控之前，现在先来复习一下单机版 Space Shooter 的玩家操控方式，由于太空战机的动作比较简单，只有左右两个方向，所以在 Player 对象的 Update 事件里可以看到呼叫 Input.GetAxisRaw("Horizontal")来取得操控设备(键盘或游戏杆)目前水平状态的程序代码，Input.GetAxisRaw("Horizontal")传回的数值只有三种，分别是 0、1、-1，0 表示静止，1 表示往右，-1 表示往左，所以在 Update()事件中随着 frame 的更新重新计算太空战机的位移值，玩家在游戏画面便可以看到移动的太空战机。

要完成在线版 Space Shooter 的玩家操控，除了原先就有的本地操控外，我们还得把自己所操控的太空战机的位置传递到主大厅给在线的玩家们，让其它玩家游戏画面里所对应属于自己的太空战机都能够实时同步出现在相同的位置。

有些程序设计师可能会在每次 Update()被触发时就送出自己的太空战机的坐标数据，不过一想到游戏程序一秒钟会执行 Update()高达 60 次，如果每次都把自己的坐标数据送出去，这样的做法并不切实际。

将游戏中移动的物体在每一个玩家的设备上同步显示是多人联机实时互动游戏中一项非常重要的

课题，在此我们暂时不讨论太复杂的算法。由于 Space Shooter 太空战机是以定速移动，同步问题相对简单许多，我们只须在方向改变时将太空战机的数据送出即可，前面提到操控设备传回 0、1、-1 三种数值，我们可以把这三个数值做为太空战机的方向值，当数值改变了也就是太空战机的方向改变了，此时我们传送这个玩家的太空战机新的方向值与目前的坐标资料到主大厅，其它玩家收到后立刻重设对应的远程玩家太空战机的方向值与坐标值，便可让游戏中每一艘太空战机的位置都能实时同步。以下是我们所定义的讯息格式：

playermove:poid/方向值/X,Y,Z

其中方向值是 0、1、-1，分别代表静止、右移、左移；而 X,Y,Z 则是目前太空飞机在 3D 世界中的坐标值。

这是修改 Update()事件程序代码后的 player.cs 脚本：

```
//Player.cs

    ... (略) ...


private float lastInputAxis=-100;
private AGCC agcc=null;

void Awake()
{
    agcc=(AGCC)FindObjectOfType(typeof(AGCC));
}

void Update ()
{
    if(state !=State.Explosion) {
        float thisInputAxis=Input.GetAxisRaw("Horizontal");

        // keyboard或joystick输入的状态变了
        if (thisInputAxis!=lastInputAxis) {
            agcc.ag.Send("playermove:"+agcc.ag.poid.ToString()+"/"+
                        thisInputAxis.ToString()+"/"+
                        transform.position.x.ToString()+","+
                        transform.position.y.ToString()+","+
                        transform.position.z.ToString());
            lastInputAxis=thisInputAxis;
        }

        // Amount to move
        float outToMove = thisInputAxis * PlayerSpeed * Time.deltaTime;

        // Move the Player
        transform.Translate(Vector3.right * outToMove);

        // ScreenWrap
        if (transform.position.x <= -8.25f)
            transform.position =
                    new Vector3(8.25f, transform.position.y, transform.position.z);
        else if (transform.position.x >= 8.25f)
            transform.position =
                    new Vector3(-8.25f, transform.position.y, transform.position.z);
    }
}

    ... (略) ...
```

### 太空战机的移动(远程操控)

收到其它玩家传来的太空战机的「playermove」讯息时，本地的游戏程序必须做出反应，步骤如下：

1. 依照讯息中的 poid 到 RegPlayers 中找到对应的 Game Object

   远程玩家的 Game Object 可以在 RegPlayers List 中被找到，为了方便使用，我们在 AGCC 对象中设计了一个函式 FindPlayer()，依照 poid 找到对应玩家的 GameObject 对象，然后把它传回来，如果找不到就传回 null :

```
// agcc.cs
    ... (略) ...
GameObject FindPlayer(int poid)
{
    lock (RegPlayers) {
        foreach (Hashtable np in RegPlayers) {
            if ((int)np["poid"]==poid) {
                return (GameObject)np["gameobject"];
            }
        }
    }
    return null;
}
    ... (略) ...
```

2. 设定该部太空战机新的移动方向与坐标

   由于 RemotePlayer.cs 中的 Update()事件使用变量 inputAxis 来控制方向，所以游戏程序从主大厅收到远程传来要改变其对应的太空战机的方向时，只要直接改变 inputAxis 的值就可以改变远程太空战机的移动方向。

   至于坐标部份，透过重新指定太空战机 GameObject 的 transform.position，它在 3D 世界中的位置就会改变。

   接着我们在 RemotePlaye 对象中设计一个新的函式 SetPlayerMotionStatus()，用来设定远程太空战机的移动方向与坐标位置。为了方便使用，我们直接传递「PlayerMove」讯息的参数字符串给 SetPlayerMotionStatus()，进入函式后再解析参数字符串，得到方向与坐标值后重新设定 inputAxis 变量与太空战机的位置坐标，以下是 SetPlayerMotionStatus()函式的程序代码：

```
// RemotePlayer.cs
    ... (略) ...
void SetPlayerMotionStatus(string param)
{
```

```
    Debug.Log("SetPlayerMotionStatus() " + param);
    string[] p  =param.Split('/');
    float   move    =float.Parse(p[0]);
    string[] a      =p[1].Split(',');

    inputAxis   =move;
    x           =float.Parse(a[0]);
    y           =float.Parse(a[1]);
    z           =float.Parse(a[2]);


    transform.position=new Vector3(x,y,z);

}

    ... (略) ...
```

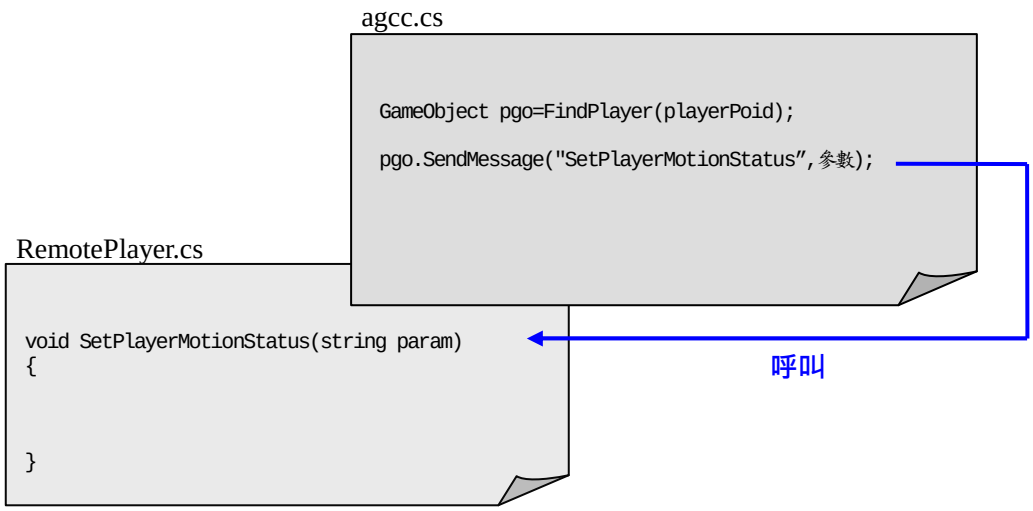
最后，我们要用到 UNITY3D 一个相当重要的程序技巧：呼叫其它 GameObject 脚本里的某个函式」。

许多刚接触 UNITY3D 程序设计的初学者常会有一个错误观念，以为 GameObject 就是面向对象程序概念里的 Object，所以会以「`GameObject.Method`」这样的语法来呼叫他所写的函式。

事实上，GameObject 控制程序中所定义对象(也就是.cs 文件里面所定义的继承自 MonoBehavior 的对象类别)才是程序设计师所认知的对象，「`GameObject.Method`」只能呼叫 `GameObject` 类别所定义的函式，不能呼叫 GameObject 控制程序中的函式。

为什么要用到这个程序技巧呢？这是因为主大厅的讯息接收是在 AGCC 对象的 `OnMessageIn()`事件中处理，当它接收到「PlayerMove」讯息后就得呼叫 RemotePlayer.cs 里 `SetPlayerMotionStatus()`，因此我们在取得远程太空战机的 GameObject 实例时，只能以 `GameObject.SendMessage()`呼叫 RemotePlayer.cs 中 `SetPlayerMotionStatus()`函式。

下图是跨 GameObject 脚本的函数调用的关系图，说明了从 agcc.cs 找到目标 `GameObject` 后，再呼叫它的控制脚本 RemotePlayer.cs 中的 `SetPlayerMotionStatus()` 函式：



了解了这样的概念后，我们着手在 agcc.cs 的 OnMessagieIn 事件里面加上处理「PlayerMove」讯息的程序代码：

```
// agcc.cs

    ... (略) ...

void OnMessageIn(string msg,int delay,ArcaletGame game)
{

    ... (略) ...

    else if (s[0]=="playermove") { // 玩家移动状态改变了
        // 讯息格式: "playermove:/poid/move/x,y,z"
        // 其中  poid:   谁的移动状态改变了
        //       move:   介于-1,1之间的值，代表玩家移动的变量，-1往左，+1往右，0停止
        //       (x,y,z):    新的坐标
        string[] p=s[1].Split('/');
        int playerPoid         =int.Parse(p[0]);

        if (playerPoid != ag.poid) {
            // 找到远程玩家的GameObject
            GameObject pgo=FindPlayer(playerPoid);
            if (pgo != null) {
                pgo.SendMessage("SetPlayerMotionStatus",
                                p[1]+"/"+p[2]+"/"+delay.ToString());
            }
            else {
                Debug.Log("Err: no such player("+playerPoid.ToString()+")");
            }
        }
    }
}

    ... (略) ...
```

### 太空战机被陨石击中

玩家所控制的太空战机被陨石击中后，除了陨石会立刻消失不见外，同时也会在这颗陨石的位置上产生爆炸效果，稍候则在别的地方产生一块大小、飞行速度、旋转速度和原陨石都不相同的新陨石，另外太空战机也要重生，重生的过程会有一段时间处于无敌状态，就算再被新的陨石撞到也没关系。无敌状态期间太空战机会一直闪烁，直到无敌状态时间到了，太空战机才会恢复正常。

我们来思考一下，当玩家控制的太空战机被陨石击中后，远程玩家游戏程序里对应的太空战机是否也应该呈现和本地玩家画面一样的效果呢？

当远程玩家专注在控制他自己的太空战机时，别人的太空战机被陨石撞到后只要也能出现爆炸火球，然后太空战机重生，感觉上这样就足够了。

远程太空战机的重生过程中，远程无敌状态对本地玩家来说是没有意义的，所以我们可以让太空战机被陨石击中时，远程的太空战机就马上消失，但爆炸的火球还是一样出现，直到本地太空战机的无敌状态结束，我们再让远程太空战机出现，这样就有相当真实的同步效果了。

接着我们进行修改 player.cs 中的碰撞侦测，当碰撞的物体是陨石时，就立刻送出讯息到主大厅，让别的玩家知道陨石打到本地的太空战机了。讯息格式如下：

```
hitplayer:no/poid/x,y,z
```

其中 no 是陨石编号，当玩家被击中时，在讯息中识别陨石代号；poid 是被击中的玩家的识别代号 poid；(x,y,z)是陨石的坐标，也就是要产生爆炸效果的坐标位置。

这是修改后的 player.cs 的 OnTrigger()事件内容如下：

```
// player.cs

    ... (略) ...

// 本地玩家被陨石撞到
void OnTriggerEnter(Collider otherObject)
{
    if(otherObject.tag == "enemy" && state == State.Playing) {
        if (enemy==null) enemy=(Enemy)FindObjectOfType(typeof(Enemy));

        // 传送飞机位置出去，让其它玩家的画面中的对应此地的太空战机产生爆炸效果
        agcc.ag.Send("hitplayer:"+enemy.enemyNo.ToString()+"/"+agcc.ag.poid+"/"+
                    transform.position.x.ToString()+","+
                    transform.position.y.ToString()+","+
                    transform.position.z.ToString());

        if (agcc.isMaster) {
            Player.Lives++;
        }

        enemy.SetPositionAndSpeed();
        StartCoroutine("DestroyShip");
    }
}

    ... (略) ...
```

*Note*

上面这段程序代码我们做了一点手脚，把被陨石撞击后的生命值改为加一，让玩家不会进入 Lose 状态，这是因为为了方便测试所做的修改，开发者可以在使用多台计算机进行测试时，不会因为手忙脚乱导致游戏很快就结束，而无法正常的测试。

由于 player.cs 中会参考到变量 enemy.enemyNo，所以要在 Enemy.cs 中再加上这个变数：

```
// Enemy.cs

    ... (略) ...

public int  enemyNo = 0;

    ... (略) ...
```

其它玩家收到远程玩家传来「hitplayer」讯息后有三个动作要做：

一、 在指定的坐标上面产生爆炸效果

二、 让这个远程玩家所控制的太空战机消失

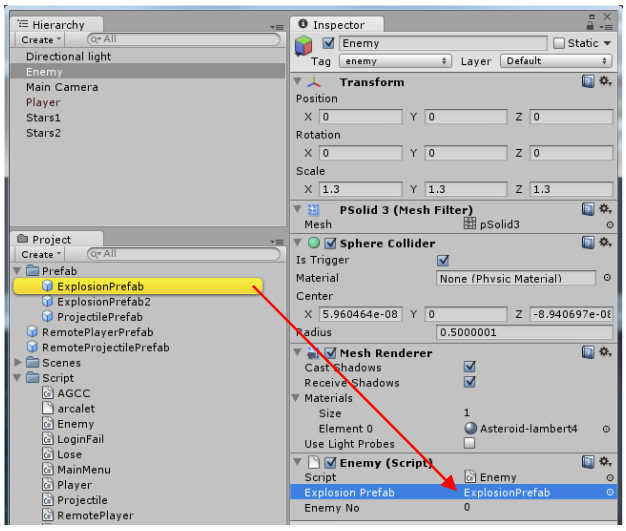
三、 收到讯息的玩家是游戏主控者，就要负责产生一个陨石，并通知所有在线的玩家；如果不是游戏的主控者，就直接让陨石消失

现在介绍第一个动作：在指定的坐标位置产生爆炸效果：

我们把这个动作写在 Enemy.cs 里，另外设计一个 `ExplosionAt()`函式，函式的参数则是远程所传过来的「hitplayer」讯息参数，经过参数解析后获得要产生爆炸效果的位置坐标，然后在此坐标位置实例化 `ExplosionPrefab` Prefab，以产生爆炸效果。

```
// Enemy.cs

    ... (略) ...
public GameObject ExplosionPrefab;

    ... (略) ...

public void ExplosionAt(string[] pos)
{
    float px=float.Parse(pos[0]);
    float py=float.Parse(pos[1]);
    float pz=float.Parse(pos[2]);
    Instantiate(ExplosionPrefab, new Vector3(px,py,pz), Quaternion.identity);
}

    ... (略) ...
```

不要忘了指定 `ExplosionPrefab` 的初值，作法大家应该都很熟悉了：




第二个动作：要先依照远程讯息的 poid 找到储存于 `RegPlayers` List 里的 GameObject，然后呼叫其控制脚本 RemotePlayer.cs 里的 `SetPlayerVisible()`函式。`SetPlayerVisible()`函式是我们要设计的新函式，它有一个 `bool` 参数，用来决定脚本所属的太空战机是消失还是显现。函式程序代码如下：

```
// RemotePlayer.cs
```

```
... (略) ...

void SetPlayerVisible(bool visible)
{
    if (visible)    gameObject.renderer.enabled =true;
    else            gameObject.renderer.enabled =false;
}

... (略) ...
```

*Note*

这里要提醒大家，此处消失的太空战机只是远程玩家操控的太空战机的同步分身，所以我们在此控制让它消失即可，至于远程的太空战机本尊在被陨石击中后的重生无敌状态并不需要同步到在线的所有玩家，直到无敌状态结束，远程玩家会再送出讯息通知在线玩家，到时候再让消失的太空战机出现就可以了。

第三个动作：要让陨石消失并不是真的删除陨石 Game Object，而是把它的位置移到目前 camera 看不到的地方，依照这个原则我们在 Enemy.cs 中定义一个让陨石消失的函式：

```
// Enemy.cs

    ... (略) ...

// 让enemy消失
public void Disapear()
{
    x = 0.0f;
    y = -7.0f;
    z = 0.0f;
    transform.position = new Vector3(x, y, z);
}

    ... (略) ...
```

如果是游戏主控者，产生新陨石的动作可以呼叫 `Enemy.SetPositionAndSpeed()`，不过我们还得修改这个 `SetPositionAndSpeed()`函式，让它可以把新产生的陨石资料传送给在线玩家，好让其它玩家也同步产生新的陨石，这个部份我们稍后再讨论。

最后，我们把处理「`hitplayer`」讯息的程序代码加到 agcc.cs 的 `OnMessageIn()`事件里：

```
// agcc.cs

    ... (略) ...

public Enemy enemy=null;

    ... (略) ...

void OnMessageIn(string msg,int delay,ArcaletGame game)
{

    ... (略) ...

    else if (s[0]=="hitplayer") { // 陨石打到了玩家
        // 讯息格式: "hitplayer:no/poid/x1,y1,z1"
        // 其中    no:  enemyNo (陨石编号，当玩家被击中时，在讯息中识别陨石代号)
        //        poid:  谁被此陨石打到
        //  (x1,y1,z1): position (玩家被击中时的位置)
```

```
        string[] p=s[1].Split('/');
        int hiterPoid =int.Parse(p[1]);
        if (hiterPoid!=ag.poid) { // 别人被陨石打到，然后传讯息过来
            if (enemy==null) enemy=(Enemy)FindObjectOfType(typeof(Enemy));

            // 产生爆炸
            enemy.ExplosionAt(p[2].Split(','));

            // 玩家暂时消失
            GameObject rp=FindPlayer(hiterPoid);
            if (rp!=null) rp.SendMessage("SetPlayerVisible",false);


            if (isMaster) {
                enemy.SetPositionAndSpeed();
            }
            else {
                // 陨石消失
                enemy.Disapear();
            }
        }
    }
}

    ... (略) ...
```

### 远程太空战机的重生

远程玩家接收到「hitplayer」讯息时会让其对应的太空战机消失，然后要再等通知让消失的太空战机再度出现，由于本地玩家会在 player.cs 的 DestroyShip()进行太空战机重生，最后当无敌状态结束时就可以发出讯息通知在线玩家同步让消失的太空战机再度出现，我们定义讯息格式如下：

```
playerreborn:poid/x,y,z
```

其中 poid 是被陨石击中的太空战机所属玩家之 poid，(x,y,z)则是太空战机重新出现时的坐标位置。

送出此讯息的程序代码就放在 DestroyShip()的最后面：

```
// player.cs

    ... (略) ...

IEnumerator DestroyShip()
{
    state = State.Explosion;

    // 被陨石撞到，所以目前玩家的飞机要爆炸
    Instantiate(ExplosionPrefab, transform.position, Quaternion.identity);

    gameObject.renderer.enabled = false;
    transform.position = new Vector3(0f, transform.position.y, transform.position.z);
    yield return new WaitForSeconds(shipInvisibleTime);
    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;
        agcc.ag.Send("playerreborn:"+agcc.ag.poid.ToString()+"/"+
                    transform.position.x.ToString()+","+
                    transform.position.y.ToString()+","+
                    transform.position.z.ToString());
    }
    else
        Application.LoadLevel("Lose");
}
    ... (略) ...
```

另外，远程玩家收到「playerreborn」讯息后则依照参数中的 poid 找到所属太空战机的 GameObject，设定好新坐标后再让它出现，为了重设远程太空飞机的位置，我们先在 RemotePlayce.cs 中加上一个新的函式 `SetPlayerPosition()`：

```
// RemotePlayer.cs

    ... (略) ...

void SetPlayerPosition(string param)
{
string[] p=param.Split(',');

transform.position=new Vector3(float.Parse(p[0]),float.Parse(p[1]),float.Parse(p[2]));
}
    ... (略) ...
```

紧接着修改 agcc.cs：

```
// agcc.cs

    ... (略) ...

void OnMessageIn(string msg,int delay,ArcaletGame game)
{

    ... (略) ...
        else if (s[0]=="playerreborn") { // 玩家重生再度出现
        // 讯息格式: "playerreborn:poid/x,y,z"
        // 其中
        //      poid:  谁重生
        //     x,y,z:  重生后的坐标
            string[] p=s[1].Split('/');

            int rebornPoid =int.Parse(p[0]);

            if (rebornPoid!=ag.poid) { // 是别人
                GameObject go=(GameObject)FindPlayer(rebornPoid);
                go.SendMessage("SetPlayerPosition",p[1]);
                go.SendMessage("SetPlayerVisible",true);
            }
        }
```

```
    ... (略) ...
}
```

**游戏主控者产生新的陨石**

不论陨石击中玩家操控的太空战机，或是后面要介绍的玩家发射炮弹击中陨石，陨石都得在爆炸之后重新产生。单机版的 Space Shooter 中，产生新陨石是由 Enemy.cs 中的 `SetPositionAndSpeed()` 函式负责，它利用随机数决定新的陨石诸元，再让它从游戏画面上方出现，非常简单。

由于陨石在在线游戏中属于「非玩家角色」(简称 NPC)，NPC 在整个游戏系统里面只能有一个控制程序，因此我们就让游戏主控者来负责产生新的陨石，然后将新陨石的数据传送给在线的玩家。

以下是我们定义的产生新陨石的讯息格式：

`enemy:/no/s1/s2/x1,y1,z1/x2,y2,z2`

其中

`no` 为 EnemyNo，新陨石的编号，当玩家命中时，在讯息中识别陨石代号

`s1` 为 currentRotationSpeed，陨石的旋转速度

`s2` 为 currentSpeed，陨石的速度

(`x1,y1,z1`)表 transform.position，陨石出现的位置

(`x2,y2,z2`)表 transform.localScale，陨石的大小形状

为了在 Enemy 对象里能够传送讯息，所以我们也得先在 `Awake()`里取得 AGCC 对象：

```
// Enemy.cs
    ... (略) ...
private AGCC agcc=null;

void Awake()
{
    // 取得AGCC对象，以便之后的讯息传送作业
    agcc=(AGCC)FindObjectOfType(typeof(AGCC));
    enemyNo=0;
}
    ... (略) ...
```

接着修改 `SetPositionAndSpeed()`：

```
// Enemy.cs
```

```
    ... (略) ...

// 产生新的enemy，并且将其数据传给其它玩家
public void SetPositionAndSpeed()
{
    // 如果本机是游戏主控者，就用随机数生成新的enemy 的参数资料
    // 如果本机不是游戏主控者，enemy 的参数数据已经被设好了(从远程的游戏主控者传过来的)
    if (agcc.isMaster) {
      currentRotationSpeed = Random.Range(MinRotateSpeed, MaxRotateSpeed);

       currentScaleX = Random.Range(MinScale, MaxScale);
       currentScaleY = Random.Range(MinScale, MaxScale);
       currentScaleZ = Random.Range(MinScale, MaxScale);

       currentSpeed = Random.Range(MinSpeed, MaxSpeed);
       x = Random.Range(-7f ,7f);
       y = 7.0f;
       z = 0.0f;
    }

    // 设定enemy 的位置
    transform.position = new Vector3(x, y, z);

    // 设定enemy 的大小
    transform.localScale = new Vector3(currentScaleX, currentScaleY, currentScaleZ);

    // 如果本地玩家是游戏主控者，就要把enemy 的数据传送出去
    // 讯息格式: "enemy:no/s1/s2/x1,y1,z1/x2,y2,z2"
    // 其中  s1:     currentRotationSpeed          (陨石的旋转速度)
    //       s2:     currentSpeed                  (陨石的速度)
    //  (x1,y1,z1): transform.position         (陨石出现的位置)
    //  (x2,y2,z2): transform.localScale           (陨石的大小形状)
    if (agcc.isMaster) {
        agcc.ag.Send("enemy:"+enemyNo+"/"+
                  currentRotationSpeed.ToString()+"/"+
                  currentSpeed.ToString()+"/"+
                  x.ToString()+","+y.ToString()+","+z.ToString()+"/"+
                  currentScaleX.ToString()+","+
                  currentScaleY.ToString()+","+
                  currentScaleZ.ToString());
        enemyNo++;
        if (enemyNo>10000) enemyNo=0;
    }
}

    ... (略) ...
```

远程玩家收到「enemy」讯息后，我们要让 AGCC 能够呼叫 Enemy.cs 里的函式来产生新的陨石，所以我们在 Enemy.cs 中实作一个称为 `ResetEnemyParam()`的函式来完成这项作业，以便让 AGCC 处理「enemy」讯息时直接呼叫使用：

```
// Enemy.cs

    ... (略) ...
private float remoteDeltaTime=0.0f;
private bool forceToResetEnemy=false;

    ... (略) ...

// 由AGCC 呼叫来产生新的enemy (也就是远程游戏主控者产生的enemy)
public void ResetEnemyParam(int enemyNo,float rotationSpeed,float speed,
                        float px,float py,float pz,
                        float sx,float sy,float sz,float delay)
{
    this.enemyNo              =enemyNo;
```

```
    currentRotationSpeed        =rotationSpeed;
    currentSpeed                =speed;
    currentScaleX               =sx;
    currentScaleY               =sy;
    currentScaleZ               =sz;
    x                           =px;
    y                           =py;
    z                           =pz;

    remoteDeltaTime             =(float)delay/1000.0f;
    forceToResetEnemy           =true;
}

    ... (略) ...
```

上面的程序最后一行将变量 forceToResetEnemy 被设定为 true，为什么要这个变数呢？

这是要用来控制 Update()事件的程序流程用的，原本单机版 Update()程序代码只是很单纯的控制陨石的移动，它是一个很简单的 NPC AI(非玩家角色人工智能)，如果陨石没有击中任何在线玩家所操控的太空战机，也没有被任何炮弹击中，最后陨石移动到游戏画面之外后，Update()程序就会呼叫 SetPositionAndSpeed()产生新的陨石。

至于在线版作业则要考虑到陨石随时都有可被任何一个玩家撞到或是击中，既然 ResetEnemyParam()是在接收到游戏主控者传来的「enemy」讯息而要产生新的陨石，它就只能透过 forceToResetEnemy 变量让 Update()程序呼叫 SetPositionAndSpeed()来产生新的陨石，也因为在这个时候，陨石的相关变量都已经变更为新的数值，SetPositionAndSpeed()就会直接依照新的数值设定陨石的参数，新陨石就这样诞生了。

接着我们照着上面的思考逻辑来改写 Update()事件的程序代码：

```
// Enemy.cs

    ... (略) ...

void Update ()
{
    float deltaTime;

    // 本地玩家若是游戏主控者，就由本地游戏程控Enemy

    if (agcc.isMaster || !forceToResetEnemy ) {
        // 若本地玩家若是游戏主控者，
        // 或虽然不是游戏主控者，但远程游戏主控者也没有传来"enemy"讯息要求产生新的enemy
        // 计算位移的时间差就要用本地与上个frame 的时间差
        deltaTime=Time.deltaTime;
    }
    else {
        // 本地玩家是游戏主控者，又收到自己送出的"enemy"讯息要产生新的enemy
        // 我们把封包延迟时间拿来做位移调整，以便让同步更接近真实状况
        deltaTime=(remoteDeltaTime*4+Time.deltaTime);
    }

    // 计算旋转值
    float rotationSpeed = currentRotationSpeed * deltaTime;
    transform.Rotate(new Vector3(-1, 0, 0) * rotationSpeed);


    // 计算位移值
    float outToMove = currentSpeed * deltaTime;
    transform.Translate(Vector3.down * outToMove, Space.World);
```

```
    // Enemy 已经跑离画面，或是游戏主控者要求产生新的enemy
    if ((agcc.isMaster && transform.position.y <=-5) || (forceToResetEnemy) ) {
        SetPositionAndSpeed();
        Player.Missed++;
    }

    forceToResetEnemy=false;

}

    ... (略) ...
```

最后，完成所有工作之后，我们来让 AGCC 对象的 OnMessageIn()事件可以处理「enemy」讯息：

```
// agcc.cs

    ... (略) ...

void OnMessageIn(string msg,int delay,ArcaletGame game)
{

    ... (略) ...

    else if (s[0]=="enemy" && !isMaster) { // 从游戏主控者传来的陨石资料
        // 讯息格式: "enemy:/no/s1/s2/x1,y1,z1/x2,y2,z2"
        // 其中      no:     enemyNo            (陨石编号，当玩家命中时，在讯息中识别陨石代号)
        //       s1:     currentRotationSpeed   (陨石的旋转速度)
        //      s2:      currentSpeed           (陨石的速度)
        // (x1,y1,z1): transform.position (陨石出现的位置)
        // (x2,y2,z2): transform.localScale   (陨石的大小形状)
        ResetEnemy(s[1],delay);
    }

    ... (略) ...

}

void ResetEnemy(string param,int delay)
{
    string[] p=param.Split('/');

    int  enemyNo              =int.Parse(p[0]);
    float rotationSpeed    =float.Parse(p[1]);
    float speed            =float.Parse(p[2]);
    string[] pos              =p[3].Split(',');
    string[] scale         =p[4].Split(',');
    float px                  =float.Parse(pos[0]);
    float py                  =float.Parse(pos[1]);
    float pz                  =float.Parse(pos[2]);
    float sx                  =float.Parse(scale[0]);
    float sy                  =float.Parse(scale[1]);
    float sz                  =float.Parse(scale[2]);

    if (enemy==null) enemy=(Enemy)FindObjectOfType(typeof(Enemy));

    if (enemy!=null)
        enemy.ResetEnemyParam(enemyNo,rotationSpeed,speed,px,py,pz,sx,sy,sz,delay);
}

    ... (略) ...
```

### 发射炮弹

炮弹发射之后，我们也要立刻送出「炮弹发射了」讯息到主大厅，让其它的玩家也在他的游戏画面上面出现炮弹，讯息格式定义如下：

```
fire:no/poid/x,y,z
```

其中 no 是炮弹编号；(x,y,z)是炮弹发射时的坐标位置。定义炮弹编号是为了让(*no*,*poid*)成为炮弹的唯一标识符，炮弹编号可以在玩家控制程序里依流水号产生，配合 poid 就可以成为唯一的标识符了。

接下来修改 player.cs，在按下发射键后的处理程序中加上送出「fire」讯息的程序代码：

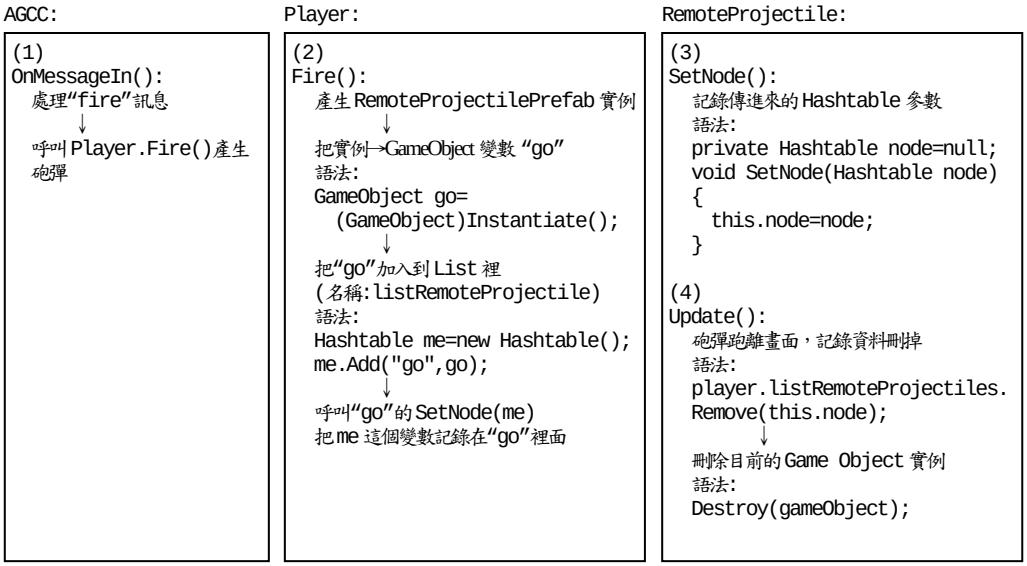
```
// player.cs

    ... (略) ...

private int projectileNo=1;
private float ProjectileOffset = 1.2f;
    ... (略) ...

void Update ()
{

    ... (略) ...

        if (Input.GetKeyDown("space")) { // 按下发射键
            // 将炮弹信息传给其他玩家
            agcc.ag.Send("fire:"+projectileNo.ToString()+"/"+
                        agcc.ag.poid.ToString()+"/"+
                        transform.position.x.ToString()+","+
                        transform.position.y.ToString()+","+
                        ProjectileOffset.ToString());
            projectileNo++;

            //在目前太空战机的位置产生炮弹
            Vector3 position = new Vector3(transform.position.x, transform.position.y +
                            ProjectileOffset);
            Instantiate(ProjectilePrefab, position, Quaternion.identity);
        }


    ... (略) ...

}
```

把「fire」讯息传送至主大厅，目的是让其它玩家的游戏画面也要出现同样的炮弹，为此我们先仿照本地的炮弹制作另一个 prefab，名称为 RemoteProjectilePrefab。

当我们呼叫 `Instantiate()`生成 RemoteProjectile 的 GameObject 实例后就把它储存在 LIST 里，等到炮弹击中目标或是飞到游戏画面之外，再从 LIST 里把这个炮弹 GameObject 删除，让不再使用的对象不会继续占用内存资源。

现在我们要一次处理好几个程序，这些程序分散于多个对象之中，为了方便大家理解，我们在此

先整理一下：




(1) 收到远程传来炮弹发射的讯息

```
// agcc.cs

    ... (略) ...
public Player player=null;
    ... (略) ...

void OnMessageIn(string msg,int delay,ArcaletGame game)
{
    ... (略) ...

    else if (s[0]=="fire") { // 远程玩家发射炮弹
        // 讯息格式: "fire:no/poid/x,y,z"
        // 其中       no:  炮弹编号, (poid,no)可做为炮弹的唯一标识符
        //          poid:   谁发射炮弹
        //      (x,y,z):  position
        string[] p=s[1].Split('/');
        int fireNo     =int.Parse(p[0]);
        int firePoid   =int.Parse(p[1]);

        if (firePoid!=ag.poid) {
            if (player==null) player=(Player)FindObjectOfType(typeof(Player));
            player.Fire(firePoid,fireNo,p[2]);
        }
    }
}

    ... (略) ...
```

由于讯息送至主大厅是包括自己在内的所有人都会收到，但是自己本身的炮弹已经产生了，所以要先判断讯息的 poid 是不是自己的，如果不是，才是远程玩家传送过来要我们在游戏画面产生他所发射的炮弹，之后便呼叫 `player.cs` 中的 `Fire()` 函式来产生远程玩家所发射的炮弹。

*Note*

这个步骤有个呼叫 `player.Fire()`的程序代码，但是我们要在下个步骤才会在 `player.cs` 中完成 `Fire()`函

式，所以现在 Console 窗口出现编译错误讯息是正常的。

(2) 产生远程玩家所发射的炮弹

```
// player.cs

using UnityEngine;
using System.Collections;
using System.Collections.Generic;

    ... (略) ...

public GameObject  RemoteProjectilePrefab;
public List<Hashtable> listRemoteProjectiles=new List<Hashtable>();

    ... (略) ...

// 远程玩家发射炮弹，在本地产生炮弹
public void Fire(int poid,int fireno,string pos)
{
    string[] p=pos.Split(',');
    float x=float.Parse(p[0]);
    float y=float.Parse(p[1]);
    float z=float.Parse(p[2]);

    // 产生本地画面的炮弹
    Vector3 position = new Vector3(x, y + z);
    GameObject go=(GameObject)Instantiate(RemoteProjectilePrefab, position,
                                  Quaternion.identity);

    // 把远程炮弹的Game Object实例记录在List中
    Hashtable me=new Hashtable();
    me.Add("poid",poid);
    me.Add("fireno",fireno);
    me.Add("go",go);
    lock (listRemoteProjectiles) {
        listRemoteProjectiles.Add(me);
    }

    // 在go的控制程序设定刚才储存在List中的记录数据，稍后会用到
    go.SendMessage("SetNode",me);
}

    ... (略) ...
```

远程炮弹是预先做好的 prefab，呼叫 `Instantiate()`产生其 GameObject 实例，然后把它指定给 `GameObject` 型态的变量 `go`。

接着使用 `Hashtable` 型态的变量 `me` 来存放 `poid`、`fireno`、`go` 这三个变量，然后把它加到 `listRemoteProjectiles` List 里。

最后一行呼叫 `go.SendMessage("SetNode",me)`，这是因为稍后介绍的炮弹控制程序 RemoteProjectile.cs 里会用到变量 `me`，所以要把这个它传给「`go`」的控制程序。

(3) 建立 RemoteProjectile.cs

一开始的 RemoteProjectile.cs 很简单，只有短短几行，SetNode()把传进来的 Hashtable 参数存放在本地变数里，稍后的炮弹控制程序会用到这些资料：

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

public class RemoteProjectile : MonoBehaviour {

    private Hashtable node=null;        // 在Player 中记录远程玩家发射的炮弹对象=本炮弹对象

    void SetNode(Hashtable node)
    {
        this.node=node;
    }
}
```

(4) 在 RemotePeojectile.cs 里的 `Update()`事件里控制炮弹的移动，当炮弹离开游戏画面，就把它删除。

把炮弹删除要做两件事：

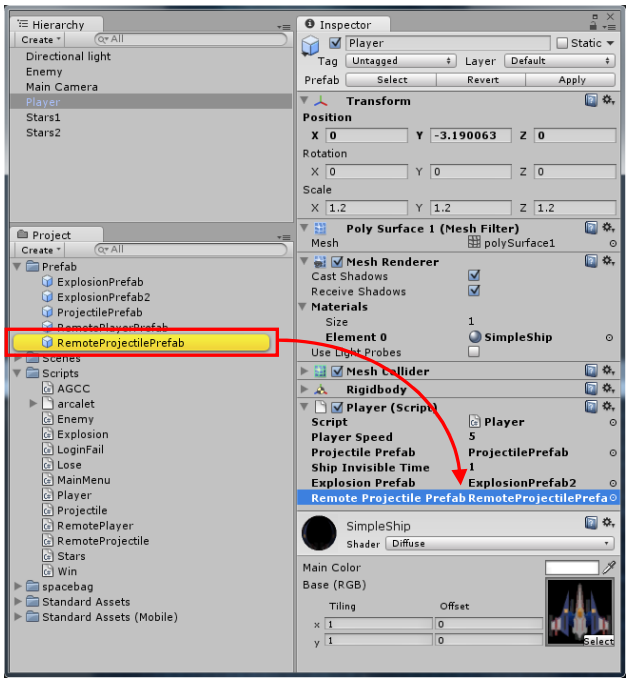
一、 移除跟目前这个远程玩家发射的炮弹有关 `Hashtable` 资料，也就是稍早在呼叫 `SetNode()`时指定给 `this.Node` 的 `Hashtable`。

二、 呼叫 `Destroy(gameObject)`，删除执行目前控制程序的 GameObject。

以下是完整 RemotePeojectile.cs 程序代码内容：

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

public class RemoteProjectile : MonoBehaviour {
    public float ProjectileSpeed;
    public GameObject ExplosionPrefab; // 记得要给初值
    private Transform myTransform;
    private Player player;
    private Hashtable node=null;       // 在Player 中记录远程玩家发射的炮弹对象=本炮弹对象

    void Awake()
    {
        // 取得Player 对象
        player=(Player)FindObjectOfType(typeof(Player));

    }

    // Use this for initialization
    void Start ()
    {
        myTransform = transform;
    }

    // Update is called once per frame
    void Update ()
    {
        float outToMove = ProjectileSpeed * Time.deltaTime;
        myTransform.Translate(Vector3.up * outToMove);

        if (myTransform.position.y > 6.25f) {
            if (this.node != null) {
                lock (player.listRemoteProjectiles) {
                    player.listRemoteProjectiles.Remove(this.node);
                }
            }
            Destroy(gameObject);
        }
    }

    void SetNode(Hashtable node)
    {
        this.node=node;
    }
}
```

(5) 指定变量 RemoteProjectile 的初值，直接从「Project」窗口把 RemoteProjectilePrefab 拖曳到 Player 中的 RemoteProjectile 里：




## 炮弹击中目标

到目前为止，我们已经可以将本地玩家的炮弹发射、移动、消失等动作与远程玩家的游戏画面同步，现在要进行最后一个步骤，让炮弹击中目标也要与远程玩家的游戏画面同步。

原本在单机版的 Space Shooter 只有一个玩家，发射的炮弹也只有一种，要决定炮弹有没有击中目标，只要在炮弹的控制程序里的 `OnTriggerEnter()`事件判断和它发生碰撞的对象是否是陨石(也就是 enemy 对象)，如果是的话，那就是击中目标了。

由于炮弹与陨石的碰撞是由 Unity3D 的碰撞侦测系统处理，在进行在线版改造作业时千万不要忘了在线游戏场景里的每个物体只能由一个主控程控，所以炮弹与陨石的碰撞侦测也只能从 `Projectile` 与 `RemoteProjectile` 选择一个控制程序来进行。

既然 Projectile.cs 在单机版时已经负责这项工作，在此我们决定仍旧由它继续担任炮弹与陨石的碰撞处理作业，当碰撞发生后再送出讯息通知在线玩家，其它玩家收到讯息后，必须把击中陨石的炮弹删除，并在陨石所在的位置产生爆炸效果，最后再让游戏主控者产生一个新的陨石，这样就可以达到炮弹与陨石碰撞的同步效果。也因为我们做了这样的规划，所以这个讯息参数就得包括「炮弹编号」与「陨石的坐标」。

接着我们定义如下的讯息格式：

```
hitenemy:eno/poid/fno/x,y,z
```

其中

eno 表 EnemyNo，也就是陨石编号，当玩家命中时，在讯息中识别陨石代号

poid 为击中此陨石的玩家之 poid

fno 表 fireNo，也就是炮弹编号

(x,y,z)表坐标，也就是陨石被命中时的位置

如果这个玩家是游戏主控者，它得要在传出「hitenemy」讯息之后负责产生一个新的陨石，由于产生新的陨石得呼叫 enemy.SetPositionAndSpeed()，这个函式已经具有同步功能了，所以在线其它玩家也会同步产生新的陨石。

如果玩家不是游戏主控者，就直接让自己本机里的陨石消失，因为前面已经送出「hitenemy」讯息，游戏主控者收到后会再负责产生新的陨石，再同步到其它玩家，这时候陨石就会再度出现。

以下修改过的 Projectile.cs 的 OnTriggerEnter()事件：

```
// Projectile.cs

    ... (略) ...
public int fireNo=0;
private AGCC agcc=null;

void Awake()
{
    // AGCC: arcalet game control center
    // 取得 AGCC 对象，以便之后要使用 isMaster 变量确定本地玩家是否为 game master
    agcc=(AGCC)FindObjectOfType(typeof(AGCC));
}

    ... (略) ...

// 本地玩家打到陨石了
void OnTriggerEnter(Collider otherObject)
{
    if(otherObject.tag == "enemy") {

        // 在目前的陨石位置上产生爆炸
        Instantiate(ExplosionPrefab, enemy.transform.position, enemy.transform.rotation);

        // 传送讯息，讯息格式为「hitenemy:eno/poid/fno/x,y,z」
        agcc.ag.Send("hitenemy:"+enemy.enemyNo.ToString()+"/"+
                agcc.ag.poid+"/"+fireNo.ToString()+"/"+
                enemy.transform.position.x.ToString()+","+
                enemy.transform.position.y.ToString()+","+
                enemy.transform.position.z.ToString());

        if (agcc.isMaster) {
            // 游戏主控者要负责产生一个新的陨石
            enemy.SetPositionAndSpeed();
        }
        else {
            // 非游戏主控者，就直接让本地的陨石消失
            enemy.Disapear();
        }

        Destroy(gameObject);
        Player.Score += 100;
```

```
    }
}

    ... (略) ...
```

AGCC 收到「hitenemy」后要做三件事：

一、把打到陨石的远程炮弹从记录它的 List 里删除

二、让被打到的陨石产生爆炸效果

三、如果自己是游戏主控者，就产生一个新的陨石，并同步到在线玩家；如果不是游戏主控，直接让陨石消失即可

为了让打到陨石的远程炮弹从记录它的 List 删除，我们另外在 Player.cs 中定义一个新的函式 RemoveFile()来完成它：

```
// player.cs

    ... (略) ...

public void RemoveFire(int poid,int fireNo)
{
    lock (listRemoteProjectiles) {
        foreach (Hashtable a in listRemoteProjectiles) {
            if ((int)(a["poid"])==poid && (int)(a["fireno"])==fireNo) {
                Destroy((GameObject)a["go"]);
                listRemoteProjectiles.Remove(a);
                break;
            }
        }
    }
}

    ... (略) ...
```

最后则是接收与处理「hitenemy」讯息的处理程序：

```
// agcc.cs

    ... (略) ...

void OnMessageIn(string msg,int delay,ArcaletGame game)
{

    ... (略) ...

    else if (s[0]=="hitenemy") {   // 玩家打到陨石
        // 讯息格式: "hitenemy:eno/poid/fno/x1,y1,z1"
        // 其中      eno:    enemyNo (陨石编号，当玩家命中时，在讯息中识别陨石代号)
        //      poid:          (谁打到了此陨石)
        //      fno:    fireNo  (炮弹编号)
        //   (x1,y1,z1): position   (陨石被命中时的位置)

        string[] p=s[1].Split('/');

        int hiterPoid  =int.Parse(p[1]);
```

```
        int fireNo     =int.Parse(p[2]);

        if (hiterPoid!=ag.poid) { // 是别人打到陨石，然后传讯息过来
            if (enemy==null) enemy=(Enemy)FindObjectOfType(typeof(Enemy));
            if (player==null) player=(Player)FindObjectOfType(typeof(Player));

            // 把远程炮弹在 List 中的数据删掉
            player.RemoveFire(hiterPoid,fireNo);

            // 产生爆炸
            enemy.ExplosionAt(p[3].Split(','));

            if (isMaster) {
                //自己是游戏主控者,则产生一个新的陨石
                enemy.SetPositionAndSpeed();
            }
            else {
                //自己不是游戏主控者，让陨石消失
                enemy.Disapear();
            }
        }
    }
}

    ... (略) ...
```

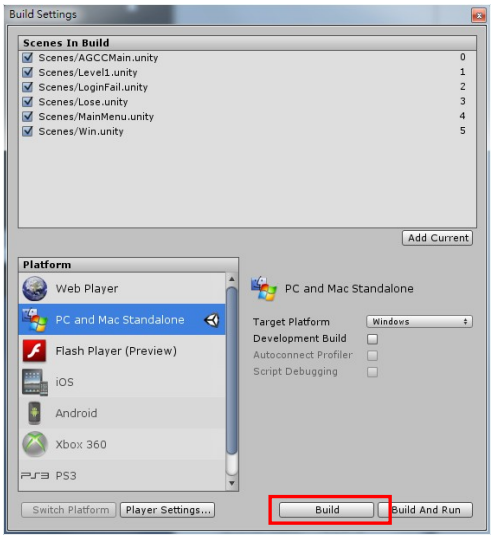
在线版 Space Shooter 改造到此告一段落，现在我们可以开始来试玩多人联机共同击退陨石的乐趣了。

## 1-6 执行与测试

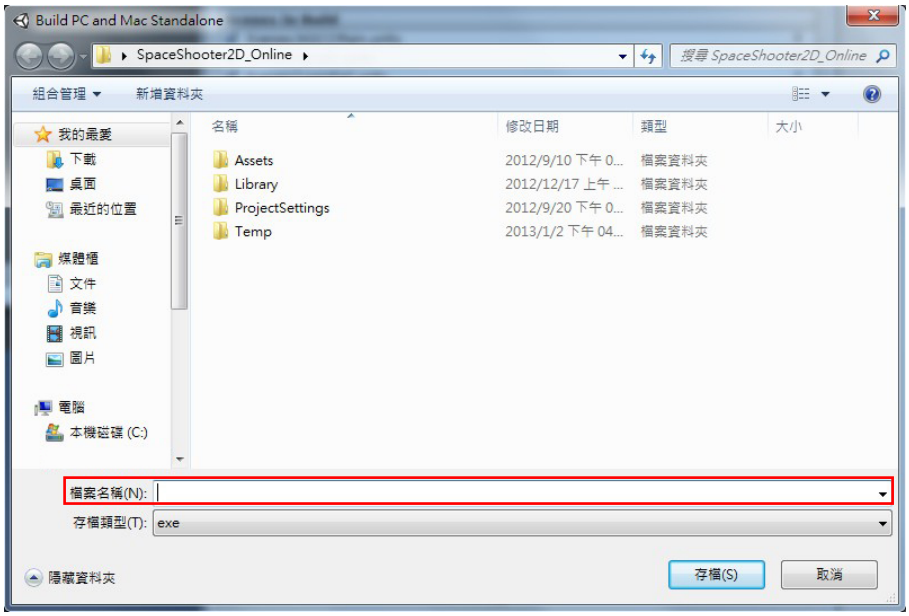
要测试多人联机游戏，首先要先有多个玩家账号才能进行，所以请先至 **arcalet** 网站注册会员账号，注册账号时填写如下的窗体：


在这个窗体里面有两个密码栏，「网站密码」就是登入 **arcalet** 网站要用的密码，而「游戏密码」则是让玩家用来登入游戏的密码，也就是要登入我们所开发的「Space Shooter Online」游戏时所需要的密码。

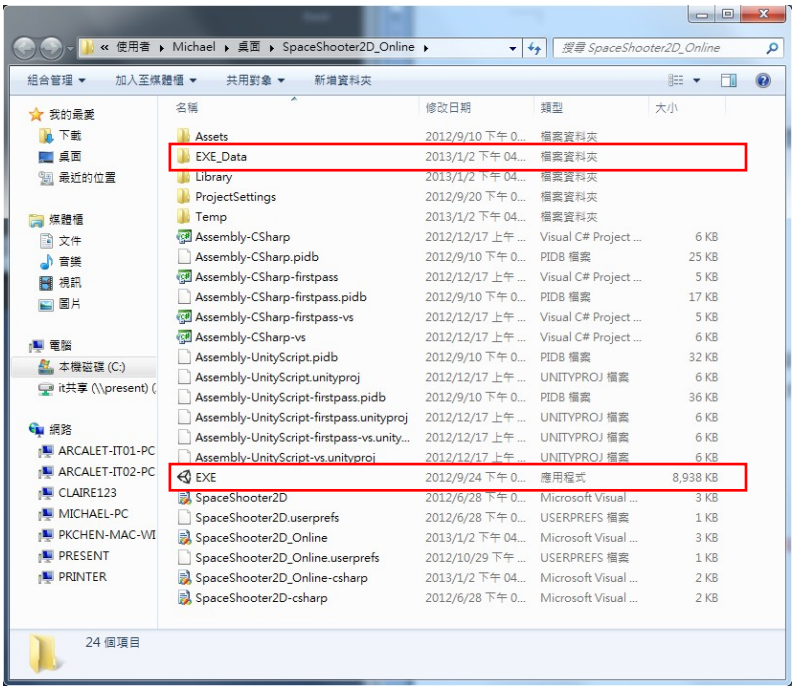
另外，游戏程序也必须在不同的计算机上执行，所以要将项目制作成 EXE 执行档：




按下「Build」钮后出现档案窗口，在此输入执行档的档名：

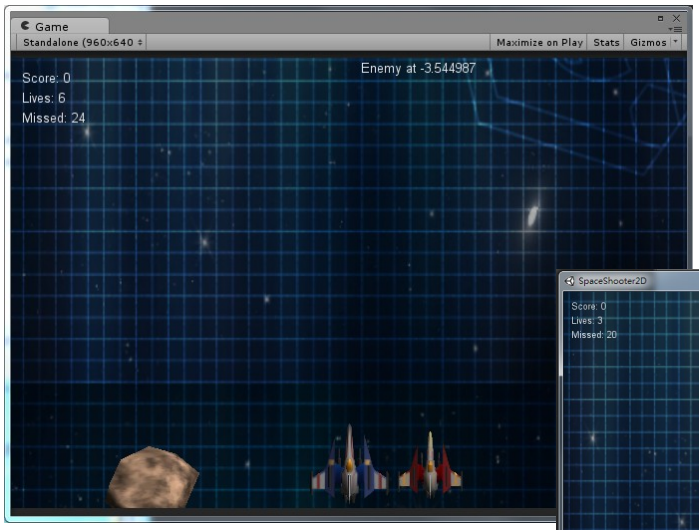



执行档制作完成后，要把 EXE 档和「文件名_Data」数据匣一起复制到另一台计算机上，不能只复制执行文件，否则游戏程序无法执行。




准备好了之后，分别在两台计算机上面输入不同的玩家账号与密码，游戏就可以开始进行了。

这是第一台计算机进入游戏场景执行的画面，本机太空飞机是蓝白色的：




玩家 1


玩家 2

第二台 Space Shooter 的玩家(玩家 2)登入后，本机(玩家 1)游戏画面出现了另外一台红色的太空飞机，红色飞机代表远程玩家所操控的太空飞机，上面右方的画面是第二个玩家的画面，他所看到的远程太空飞机在左边，刚好跟第一台游戏的画面相反。

**完整专案下载请至：**

http://developer.arcalet.com