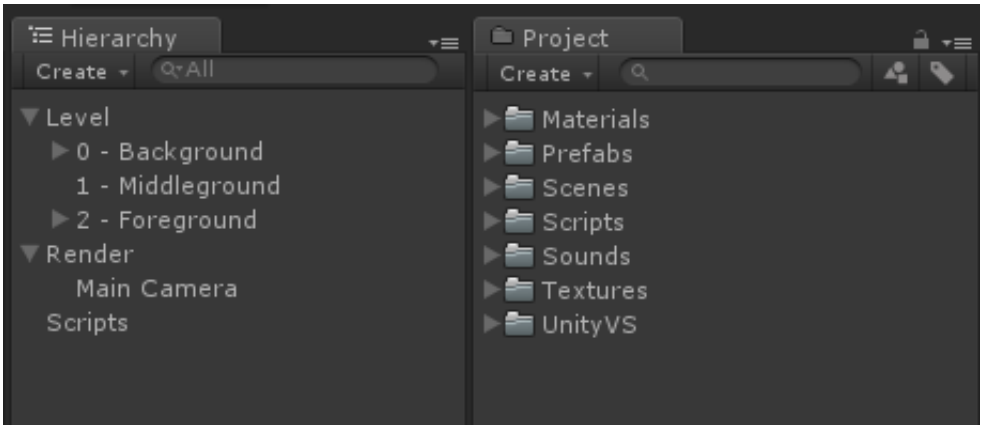
Unity3D开发一个2D横版射击游戏 - 姑苏慕容复

教程基于http://pixelnest.io/tutorials/2d-game-unity/ ，这个例子感觉还是比较经典的，网上转载的也比较多．刚好最近也在学习U3D，做的过程中自己又修改了一些地方，写篇文和大家一起分享下，同时也加深记忆．有什么纰漏的地方还请大家多包涵．

1.创建第一个场景

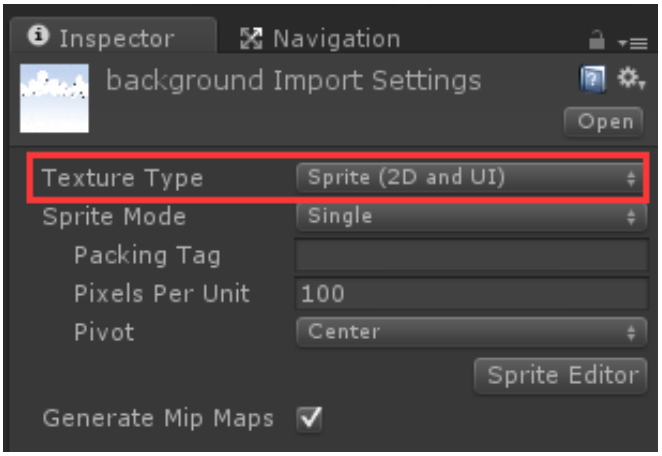


新建工程,在Project面板创建文件夹，是为了更好的规划管理资源文件．

接着在Hierarchy面板上创建多个空对象(这样的结构也是清晰了整个游戏的层次，对象之间的关系一目了然)，这些空对象的Position保持(0,0,0)即可．保存场景到Scenes文件夹中，名称为Stage1．

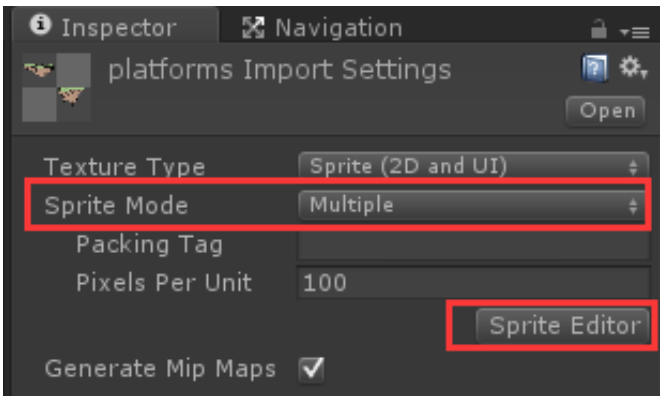
2.添加显示背景

将背景图片放入Textures文件夹，确认这张图片的纹理类型Texture Type为Sprite．

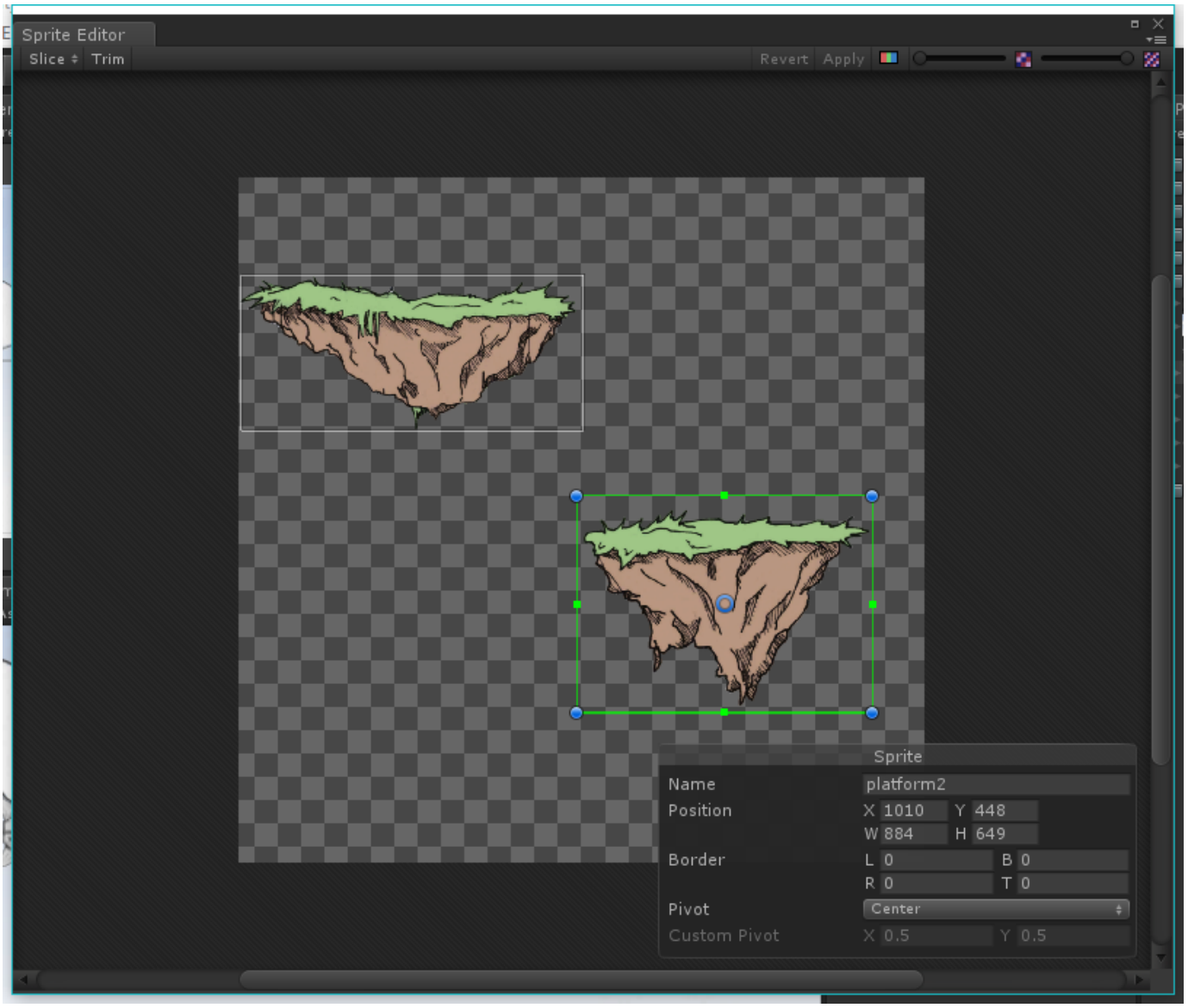


在场景中添加一个Sprite游戏对象,命名为Background1,设置Sprite属性为刚才导入的背景图片，将它移动到0 - Background中，设置Position为(0,0,0)．

接着添加背景元素．导入平台岛屿图片到Textures文件夹，选中Platforms图片，设置它的Sprite Mode为Multiple，然后点击Sprite Editor，如下图:



在弹出的Sprite Editor窗口中，进行绘制每个平台岛屿的包围矩形，以便将纹理分隔成更小的部分．然后分别命名为platform1和platform2.

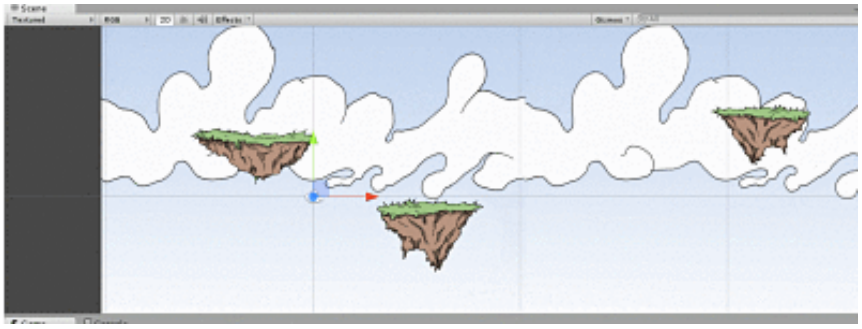


创建一个新的Sprite对象，设置它的Sprite为platform1．然后同样再创建一个Sprite对象，设置Sprite为platform2．将它们放置到1 - Middleground对象里，并且确认他们的Z坐标为0．设置完成后，将这两个对象从Hierarchy面板拖动到Project面板下的Prefabs文件夹，保存为预制对象．接着，为了避免显示顺序问题，修改下游戏对象的Z坐标，如下所示:

Layer	Z Position
0 - Background	10

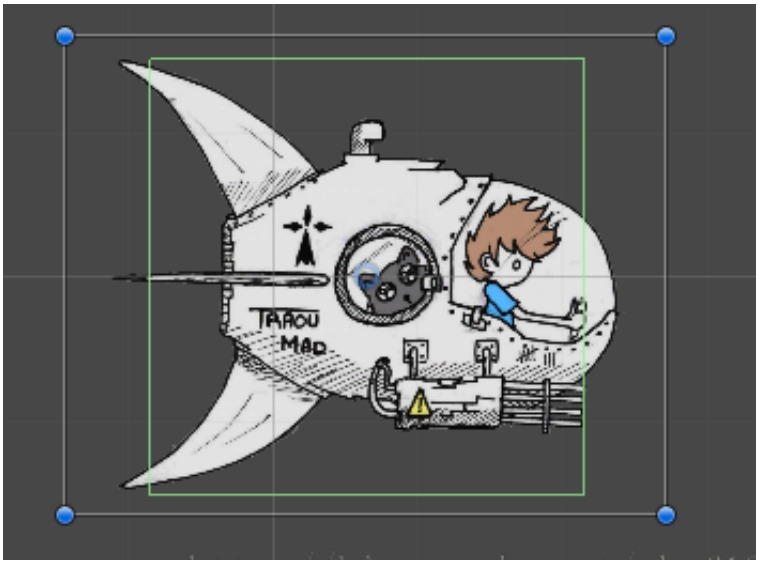
1 - Middleground	5
2 - Foreground	0

此时，点击Scene面板上的2D到3D视图切换，可以清除的看到层次：

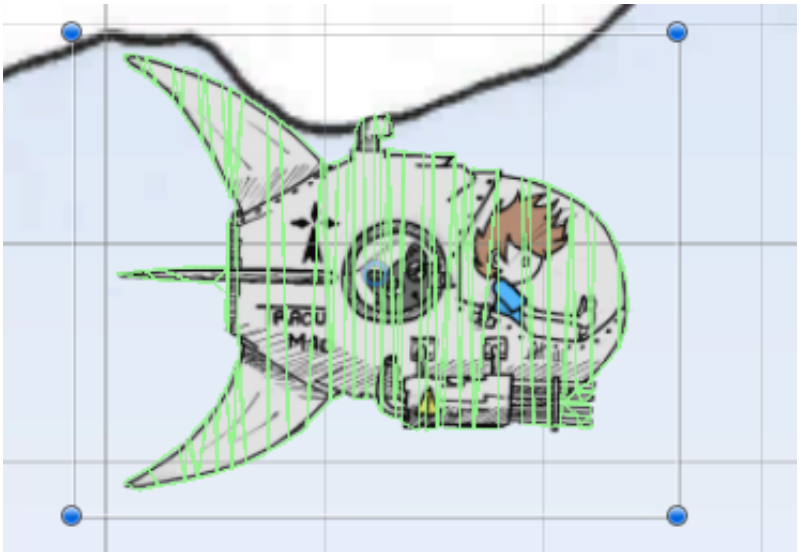


3.创建玩家和敌人

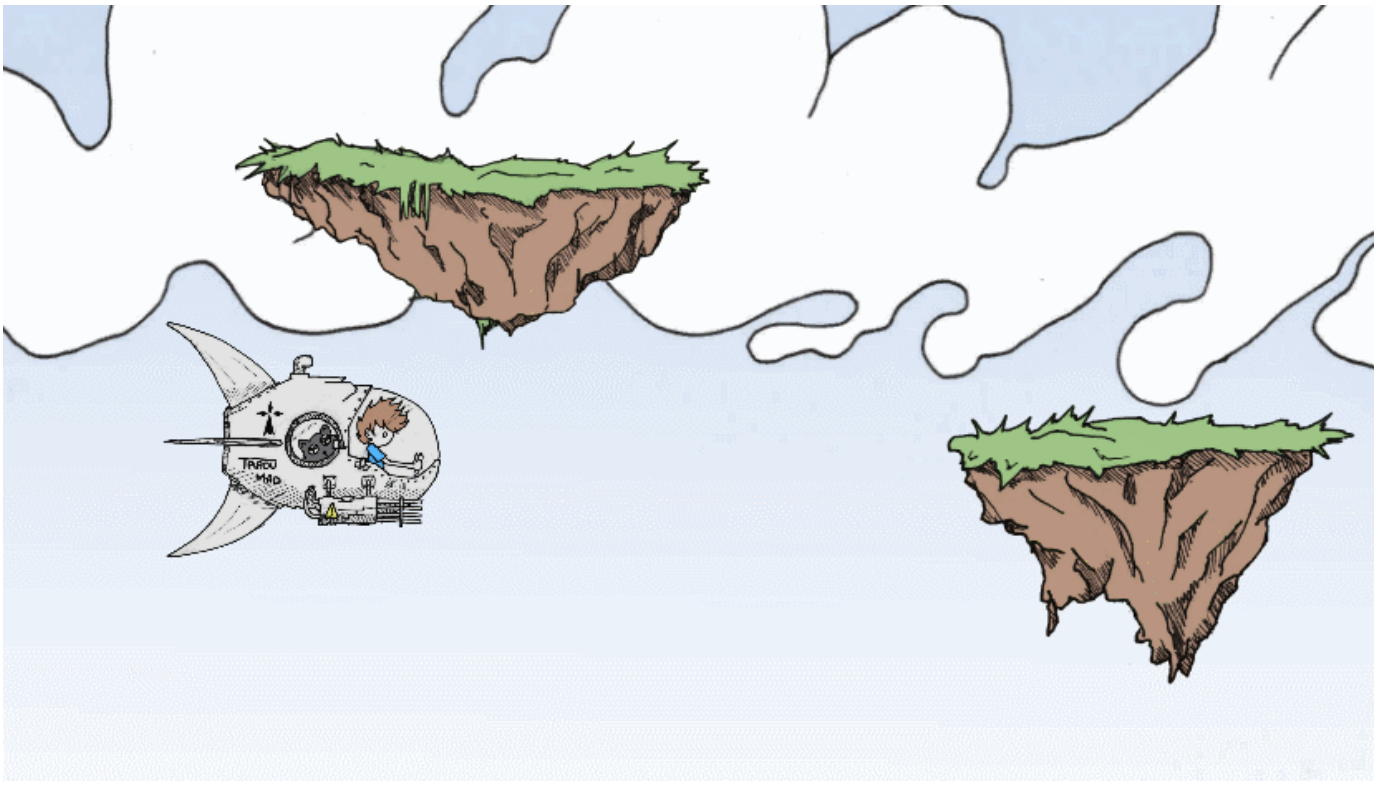
导入主角图片到Textures文件夹，创建一个Sprite对象，命名为Player，设置其Sprite属性为刚才导入的主角图片．将它放入2 - Foreground中，设置Scale为(0.2,0.2,1)．接着，为主角添加碰撞机，点击Add Component按钮，选择Box Collider 2D，设置Size为(10,10)，虽然大于实际区域，但是已经比图片小多了．



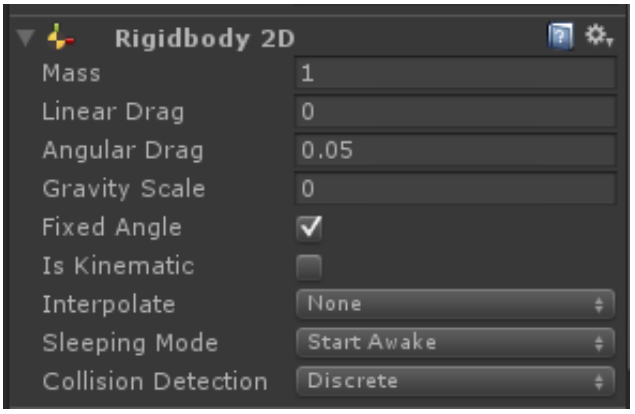
但是我更愿意去使用Polygon Collider 2D来达到更精致的效果，这里只是个例子，大家可以自由选择．



接着，再为主角对象添加Rigidbody 2D刚体组件，现在运行可以看到如下结果：



可以看到主角往下落了，这是因为刚体带有重力，但在这个游戏中我们用不到重力，将Gravity Scale设置为0即可．另外，不想因为物理而引起的主角旋转，则将Fixed Angles勾选上．



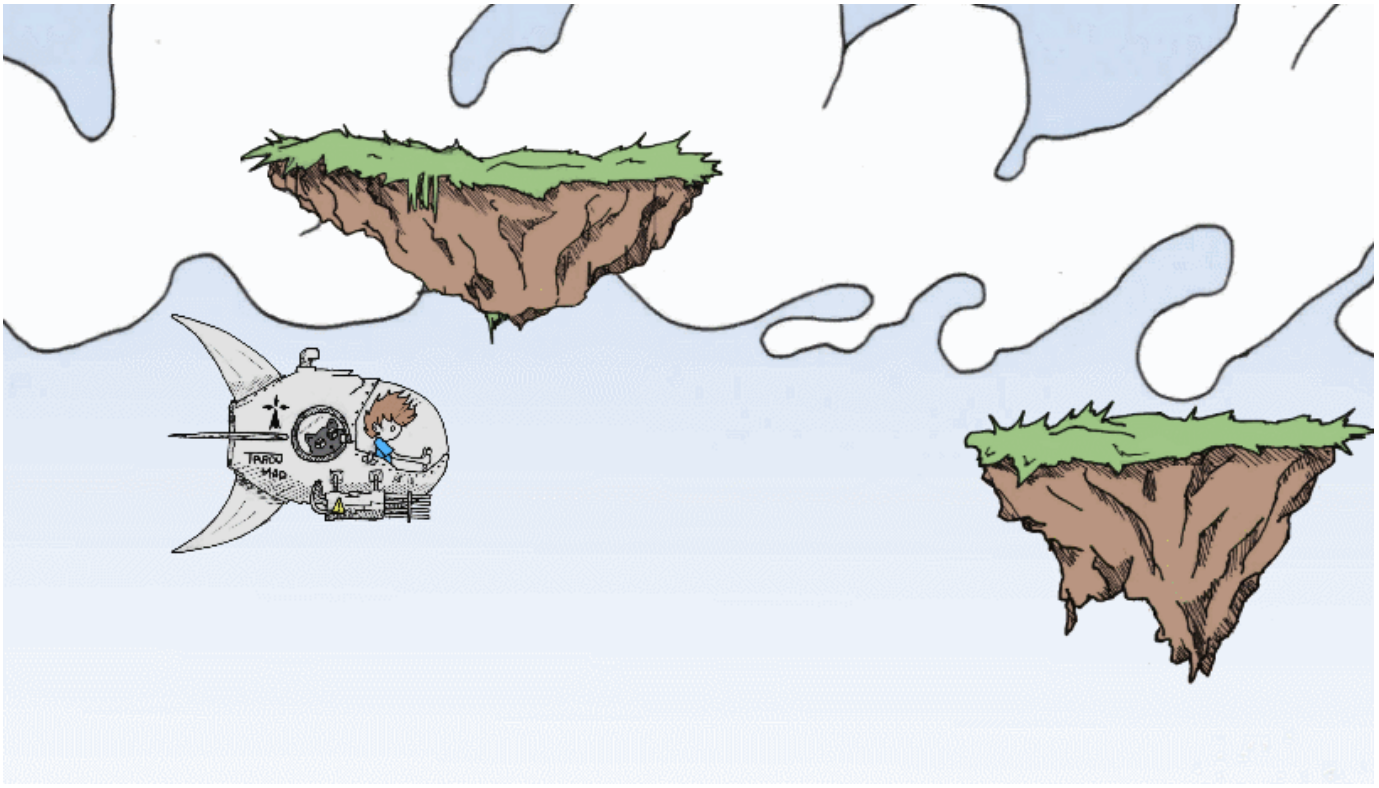
开始准备让主角移动．在Scripts文件夹中，创建一个C#脚本，名称为PlayerScript，实现让方向键移动主角，代码如下：

```
using UnityEngine;
using System.Collections;

/// <summary>
/// 玩家控制器和行为
/// </summary>
public class PlayerScript : MonoBehaviour
{
    #region 1 - 变量
```

```
    /// <summary>
    /// 飞船移动速度
    /// </summary>
    private Vector2 speed = new Vector2(50, 50);

    // 存储运动
    private Vector2 movement;

    #endregion

    // Update is called once per frame
    void Update()
    {
        #region 运动控制

        // 2 - 获取轴信息
        float inputX = Input.GetAxis("Horizontal");
        float inputY = Input.GetAxis("Vertical");

        // 3 - 保存运动轨迹
        movement = new Vector2(speed.x * inputX, speed.y * inputY);

        #endregion
    }

    void FixedUpdate()
    {
        // 4 - 让游戏物体移动
        rigidbody2D.velocity = movement;
    }
}
```

这里以改变刚体的速率来达到主角移动的效果，而不是通过直接改变transform.Translate，因为那样的话，可能会不产生碰撞. 另外， 这里有人可能会疑问为什么实现移动的代码要写在FixedUpdate而不是Update中，请看Update和FixedUpdate的区别：传送门.

现在将此脚本附加到主角对象上，点击运行，方向键来控制移动.

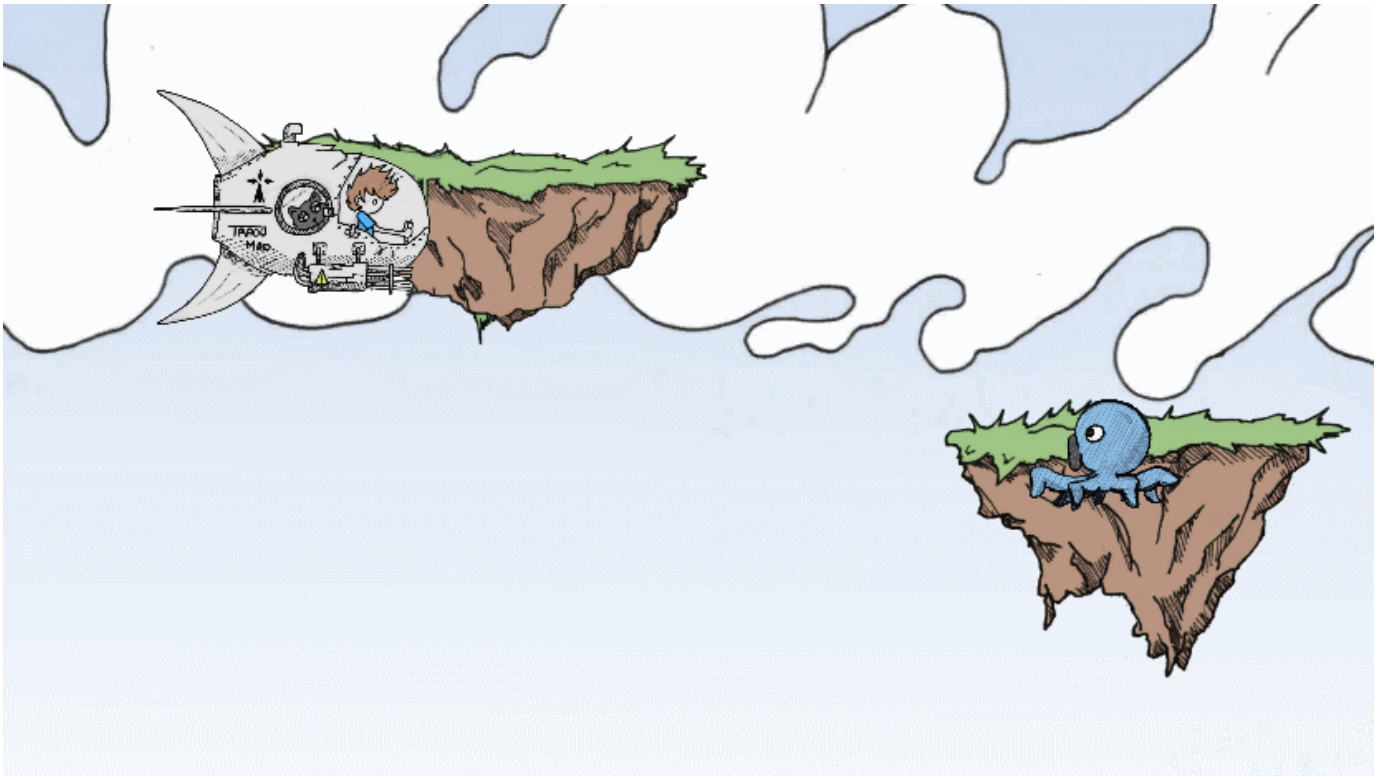


接下来，添加第一个敌人. 导入章鱼敌人图片到Textures文件夹，创建一个Sprite对象，命名为Poulpi，设置Sprite为刚才导入的章鱼图片，设置Scale为(0.4,0.4,1)，添加碰撞机(Polygon Collider 2D或Box Collider 2D都可以)，如果是Box Collider 2D，设置Size为(4,4)，添加Rigidbody 2D组件，设置Gravity Scale为0，并且勾选Fixed Angles属性. 设置完成后，将对象保存为预制. 在这里只让章鱼简单的往前行走，创建一个脚本，命名为MoveScript，代码如下:

```
using UnityEngine;
using System.Collections;

/// <summary>
///  当前游戏对象简单的移动行为
/// </summary>
public class MoveScript : MonoBehaviour
{
    #region 1 - 变量

    /// <summary>
    /// 物体移动速度
    /// </summary>
    public Vector2 speed = new Vector2(10, 10);

    /// <summary>
    /// 移动方向
    /// </summary>
    public Vector2 direction = new Vector2(-1, 0);
```

```
    private Vector2 movement;

    #endregion

    // Use this for initialization
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        // 2 - 保存运动轨迹
        movement = new Vector2(speed.x * direction.x, speed.y * direction.y);
    }

    void FixedUpdate()
    {
        // 3 - 让游戏物体移动
        rigidbody2D.velocity = movement;
    }
}
```

将此脚本附加到章鱼对象上，现在运行吗可以看到章鱼往前移动，如图:



此时如果主角和章鱼发生碰撞，会互相阻塞对方的移动.

4.射击

导入子弹图片到"Textures"文件夹，创建一个Sprite游戏对象，命名为"PlayerShot"，设置其"Sprite"属性为刚才导入的图片，设置"Scale"属性为(0.75, 0.75, 1)，添加"Rigidbody 2D"组件，其"Gravity Scale"属性为0，并且勾选"Fixed Angles"属性框，添加"Box Collider 2D"组件，其Size为(1, 1)，并且勾选"IsTrigger"属性。勾选"IsTrigger"属性表示该碰撞体用于触发事件，并将被物理引擎所忽略。意味着，子弹将穿过触碰到的对象，而不会阻碍对象的移动，触碰的时候将会引发"OnTriggerEnter2D"事件。创建一个脚本，命名为"ShotScript"，代码如下：

```
using UnityEngine;
using System.Collections;

/// <summary>
/// 子弹行为
/// </summary>
public class ShotScript : MonoBehaviour
{
    #region 1 - 变量

    /// <summary>
    /// 造成伤害
    /// </summary>
    public int damage = 1;

    /// <summary>
    /// 子弹归属 , true-敌人的子弹, false-玩家的子弹
    /// </summary>
    public bool isEnemyShot = false;

    #endregion

    // Use this for initialization
    void Start()
    {
        // 2 - 为避免任何泄漏,只给予有限的生存时间.[20秒]
        Destroy(gameObject, 20);
    }
}
```

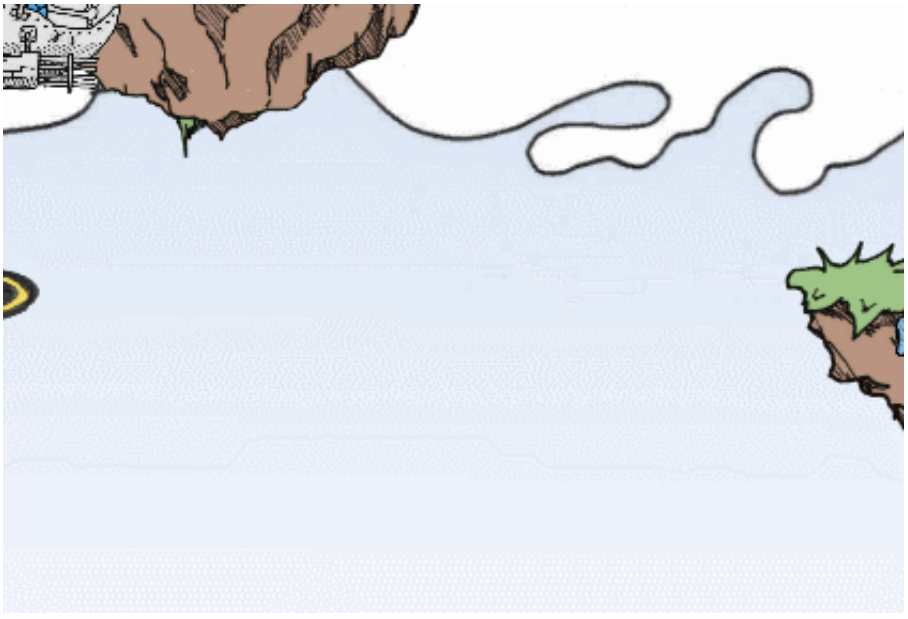
将此脚本附加到子弹对象上，然后将"MoveScript"脚本也附加到子弹对象上以便可以移动。保存此对象

为预制。接着，让碰撞产生伤害效果。创建一个脚本，命名为"HealthScript"，代码如下：

```
using UnityEngine;
using System.Collections;

/// <summary>
/// 处理生命值和伤害
/// </summary>
public class HealthScript : MonoBehaviour
{
    #region 1 - 变量

    /// <summary>
    /// 总生命值
    /// </summary>
    public int hp = 1;

    /// <summary>
    /// 敌人标识
    /// </summary>
    public bool isEnemy = true;

    #endregion

    /// <summary>
    /// 对敌人造成伤害并检查对象是否应该被销毁
    /// </summary>
    /// <param name="damageCount"></param>
    public void Damage(int damageCount)
    {
        hp -= damageCount;

        if (hp <= 0)
        {
            // 死亡! 销毁对象!
            Destroy(gameObject);
        }
    }

    void OnTriggerEnter2D(Collider2D otherCollider)
    {
        ShotScript shot = otherCollider.gameObject.GetComponent<ShotScript>();
        if (shot != null)
```

```
        {
            // 判断子弹归属,避免误伤
            if (shot.isEnemyShot != isEnemy)
            {
                Damage(shot.damage);

                // 销毁子弹
                // 记住，总是针对游戏的对象，否则你只是删除脚本
                Destroy(shot.gameObject);
            }
        }
    }
}
```

将此脚本附加到Poulpi预制体上。现在运行，让子弹和章鱼碰撞，可以看到如下结果：



如果章鱼的生命值大于子弹的伤害值，那么章鱼就不会被消灭，可以试着通过改变章鱼对象的"HealthScript"的hp值。

接着，我们来准备射击。创建一个脚本，命名为"WeaponScript"，代码如下：

```
using UnityEngine;
using System.Collections;

/// <summary>
/// 发射子弹
/// </summary>
public class WeaponScript : MonoBehaviour
{
```

```
    #region 1 - 变量

    /// <summary>
    /// 子弹预设
    /// </summary>
    public Transform shotPrefab;

    /// <summary>
    /// 两发子弹之间的发射间隔时间
    /// </summary>
    public float shootingRate = 0.25f;

    /// <summary>
    /// 当前冷却时间
    /// </summary>
    private float shootCooldown;

    #endregion

    // Use this for initialization
    void Start()
    {
        // 初始化冷却时间为0
        shootCooldown = 0f;
    }

    // Update is called once per frame
    void Update()
    {
        // 冷却期间实时减少时间
        if (shootCooldown > 0)
        {
            shootCooldown -= Time.deltaTime;
        }
    }

    /// <summary>
    /// 射击
    /// </summary>
    /// <param name="isEnemy">是否是敌人的子弹</param>
    public void Attack(bool isEnemy)
    {
        if (CanAttack)
        {
```

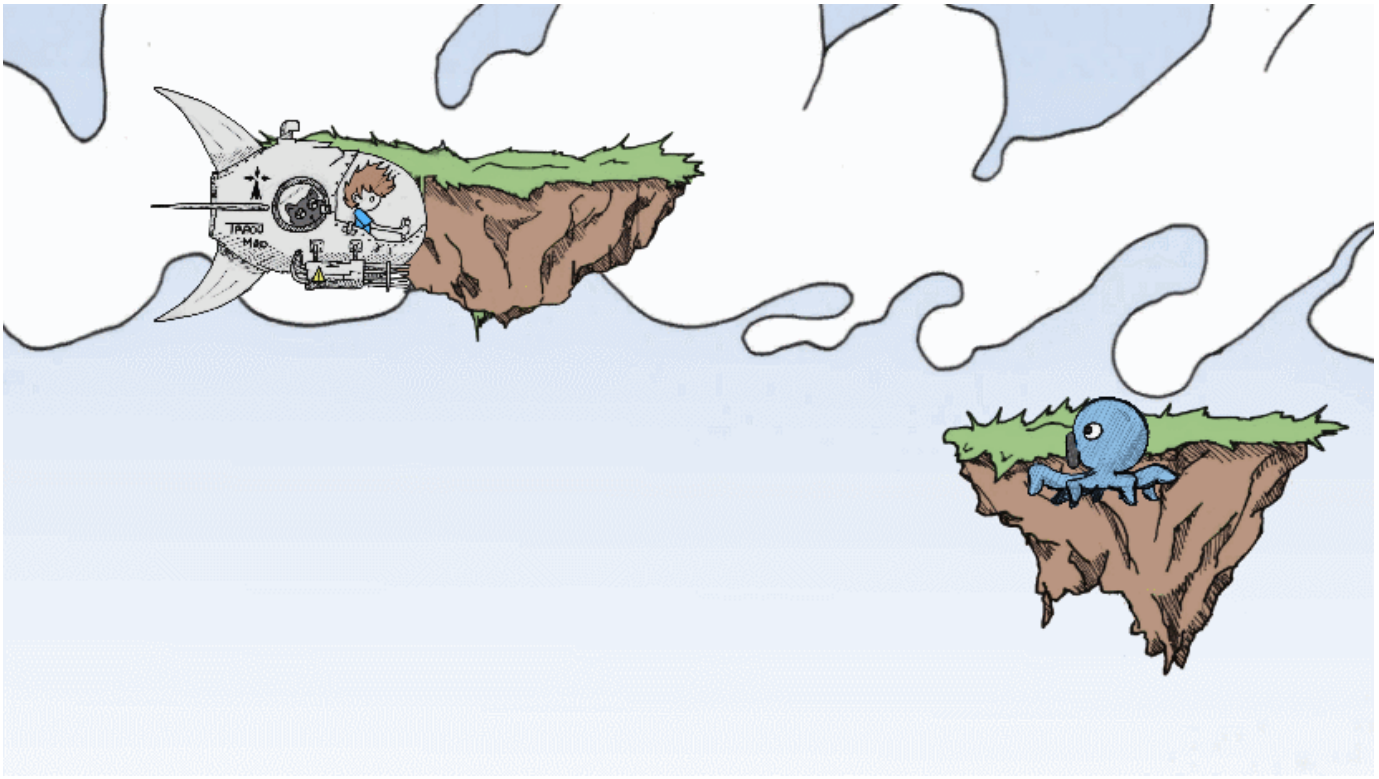
```
            if (isEnemy)
            {
                SoundEffectsHelper.Instance.MakeEnemyShotSound();
            }
            else
            {
                SoundEffectsHelper.Instance.MakePlayerShotSound();
            }

            shootCooldown = shootingRate;

            // 创建一个子弹
            var shotTransform = Instantiate(shotPrefab) as Transform;

            // 指定子弹位置
            shotTransform.position = transform.position;

            // 设置子弹归属
            ShotScript shot = shotTransform.gameObject.GetComponent<ShotScript>();
            if (shot != null)
            {
                shot.isEnemyShot = isEnemy;
            }

            // 设置子弹运动方向
            MoveScript move = shotTransform.gameObject.GetComponent<MoveScript>();
            if (move != null)
            {
                // towards in 2D space is the right of the sprite
                move.direction  =  this.transform.right;
            }
        }
    }

    /// <summary>
    /// 武器是否准备好再次发射
    /// </summary>
    public bool CanAttack
    {
        get
        {
            return shootCooldown <= 0f;
        }
    }
```

```
}
```

将这个脚本附加到主角对象上，设置其"Shot Prefab"属性为"PlayerShot"预制体。打开"PlayerScript"脚本，在Update()方法里面，加入以下片段：

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

        #region 射击控制

        // 5 - 射击
        bool shoot = Input.GetButtonDown("Fire1");
        shoot |= Input.GetButtonDown("Fire2");
        // 小心：对于Mac用户，按Ctrl +箭头是一个坏主意

        if (shoot)
        {
            WeaponScript weapon = GetComponent<WeaponScript>();
            if (weapon != null)
            {
                weapon.Attack(false);
            }
        }

        #endregion
    }
```

当收到射击的按钮状态，调用Attack(false)方法。现在运行，可以看到如下结果：

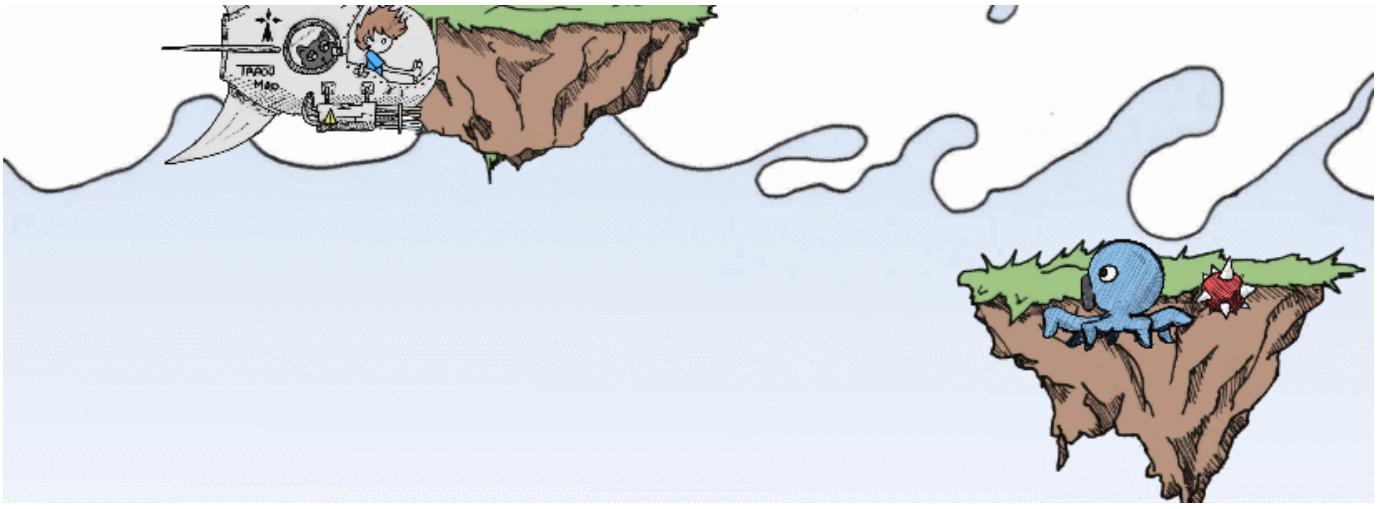


接下来，准备创建敌人的子弹。导入敌人子弹图片到"Textures"文件夹，选中"PlayerShot"预制体，按下Ctrl+D进行复制，命名为"EnemyShot1"，然后改变其Sprite为刚才导入的图片，设置其Scale为(0.35, 0.35, 1)。接着，让章鱼可以射击。将"WeaponScript"脚本附加到章鱼对象上，拖动"EnemyShot1"预制体到其"Shot Prefab"属性，创建一个脚本，命名为"EnemyScript"，用来简单地每一帧进行自动射击，代码如下：

```
using UnityEngine;
using System.Collections;

/// <summary>
/// 敌人通用行为
/// </summary>
public class EnemyScript : MonoBehaviour
{
    #region 变量

    private WeaponScript weapon;

    #endregion

    void Awake()
    {
        // 只检索一次武器
        weapon = GetComponent<WeaponScript>();
    }
```

```
    // Update is called once per frame
    void Update()
    {
            // 自动开火
            if (weapon != null && weapon.CanAttack)
            {
                weapon.Attack(true);
            }
    }
}
```

将此脚本附加到章鱼对象上，现在运行，可以看到如下结果：



可以看到章鱼向右射击了子弹，这是因为代码就是让它那么做的。实际上，应该做到可以朝向任何方向射击子弹。修改"WeaponScript"中的Attack方法，代码为如下：

```
    /// <summary>
    /// 射击
    /// </summary>
    /// <param name="isEnemy">是否是敌人的子弹</param>
    public void Attack(bool isEnemy)
    {
        if (CanAttack)
        {
            if (isEnemy)
            {
                SoundEffectsHelper.Instance.MakeEnemyShotSound();
            }
            else
            {
                SoundEffectsHelper.Instance.MakePlayerShotSound();
```

```
            }

            shootCooldown = shootingRate;

            // 创建一个子弹
            var shotTransform = Instantiate(shotPrefab) as Transform;

            // 指定子弹位置
            shotTransform.position = transform.position;

            // 设置子弹归属
            ShotScript shot = shotTransform.gameObject.GetComponent<ShotScript>();
            if (shot != null)
            {
                shot.isEnemyShot = isEnemy;
            }

            // 设置子弹方向
            MoveScript move = shotTransform.gameObject.GetComponent<MoveScript>();
            if (move != null)
            {
                // towards in 2D space is the right of the sprite
                //move.direction = move.direction;

                // 如果是敌人的子弹，则改变方向和移动速度
                if (shot.isEnemyShot)
                {
                    move.direction.x = -1f;
                    move.speed = new Vector2(15, 15);
                }
                else
                {
                    move.direction.x = 1f;
                    move.speed = new Vector2(10, 10);
                }
            }
        }
    }
```

可以适当调整子弹的移动速度，它应该快于章鱼的移动速度。现在运行，如下图所示：

目前，当主角和章鱼碰撞时，仅仅只是阻碍对方的移动而已，在这里改成互相受到伤害。打开"PlayerScript"文件，添加如下代码：

```
void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.ToString().IndexOf("Poulpi") >= 0)
    {
        bool damagePlayer = false;

        // 与敌人发生碰撞
        EnemyScript enemy = collision.gameObject.GetComponent<EnemyScript>();
        if (enemy != null)
        {
            // 杀死敌人
            HealthScript enemyHealth = enemy.GetComponent<HealthScript>();
            if (enemyHealth != null)
            {
                enemyHealth.Damage(enemyHealth.hp);
            }

            damagePlayer = true;
        }

        // 玩家也受到伤害
        if (damagePlayer)
        {
            HealthScript playerHealth = this.GetComponent<HealthScript>();
            if (playerHealth != null)
            {
                playerHealth.Damage(1);
            }
        }
    }
}
```

5.视差卷轴效果

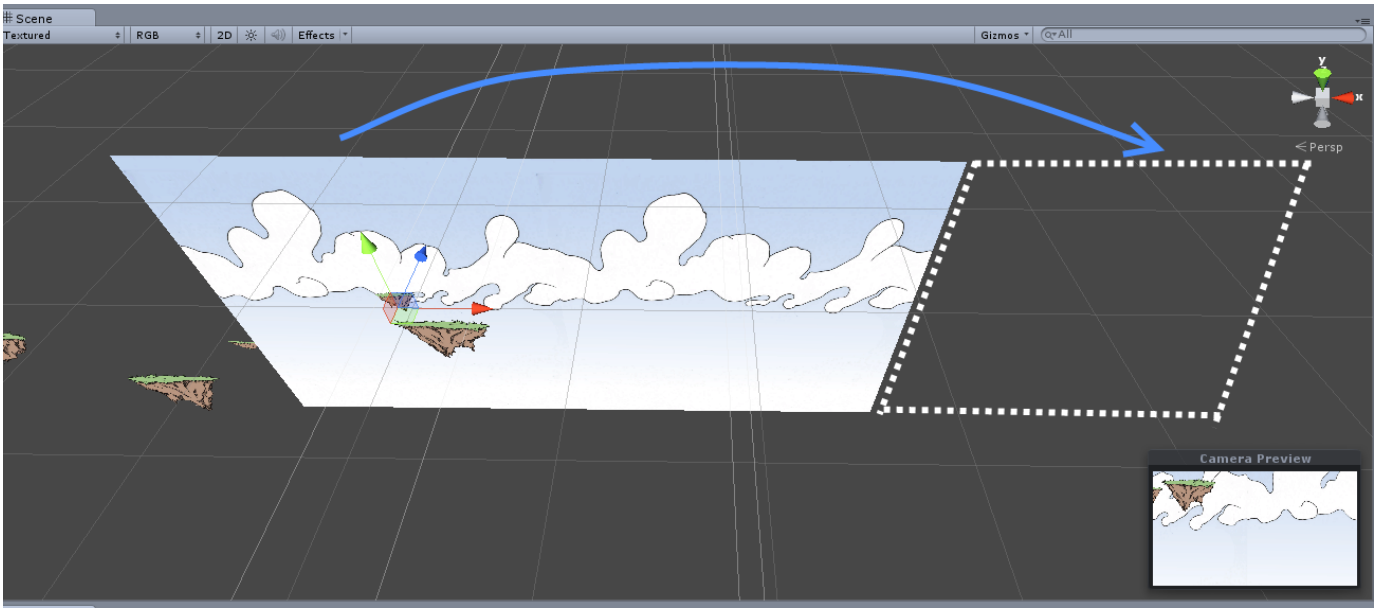
为了达到这种视差卷轴的效果，可以让背景层以不同的速度进行移动，越远的层，移动地越慢。如果操作得当，这可以造成深度的错觉，这将很酷，又是可以容易做到的效果。在这里存在两个滚动：

- 主角随着摄像机向前推进
- 背景元素除了摄像机的移动外，又以不同的速度移动

一个循环的背景将在水平滚动的时候，一遍又一遍的重复进行显示。现有的层如下：

Layer	Loop	Position
0 - Background	Yes	(0, 0, 10)
1 - Middleground	No	(0, 0, 5)
2 - Foreground	No	(0, 0, 0)

接下来，实现无限背景。当左侧的背景对象远离了摄像机的左边缘，那么就将它移到右侧去，一直这样无限循环，如下图所示：



要做到检查的对象渲染器是否在摄像机的可见范围内，需要一个类扩展。创建一个C#文件，命名为"RendererExtensions.cs"，代码如下：

```
using UnityEngine;
using System.Collections;

/// <summary>
/// 渲染扩展
/// </summary>
```

```
public static class RendererExtensions
{
    /// <summary>
    /// 检查对象渲染器是否在摄像机的可见范围内
    /// </summary>
    /// <param name="renderer">渲染对象</param>
    /// <param name="camera">摄像机</param>
    /// <returns></returns>
    public static bool IsVisibleFrom(this Renderer renderer, Camera camera)
    {
        Plane[] planes = GeometryUtility.CalculateFrustumPlanes(camera);
        return GeometryUtility.TestPlanesAABB(planes, renderer.bounds);
    }
}
```

接下来，可以开始实现不带背景循环的滚动。创建一个脚本，命名为"ScrollingScript"，代码如下：

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

/// <summary>
/// 背景视差滚动脚本
/// </summary>
public class ScrollingScript : MonoBehaviour
{
    #region 变量

    /// <summary>
    /// 滚动速度
    /// </summary>
    public Vector2 speed = new Vector2(2, 2);

    /// <summary>
    /// 移动方向
    /// </summary>
    public Vector2 direction = new Vector2(-1, 0);

    /// <summary>
    /// 相机是否运动
    /// </summary>
```

```
    public bool isLinkedToCamera = false;

    /// <summary>
    /// 背景是否循环
    /// </summary>
    public bool isLooping = false;

    /// <summary>
    /// 渲染对象名单
    /// </summary>
    private List<Transform> backgroundPart;

    #endregion

    // Use this for initialization
    void Start()
    {
        // 只循环背景
        if (isLooping)
        {
            // 获取该层渲染器的所有子集对象
            backgroundPart = new List<Transform>();

            for (int i = 0; i < transform.childCount; i++)
            {
                Transform child = transform.GetChild(i);

                // 只添加可见子集
                if (child.renderer != null)
                {
                    backgroundPart.Add(child);
                }
            }

            // 根据位置排序
            // Note: 根据从左往右的顺序获取子集对象
            // 我们需要增加一些条件来处理所有可能的滚动方向。
            backgroundPart = backgroundPart.OrderBy(t => t.position.x).ToList();
        }
    }

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

```
        // 创建运动状态
        Vector3 movement = new Vector3(speed.x * direction.x, speed.y * direction.y, 0);

        movement *= Time.deltaTime;
        transform.Translate(movement);

        // 移动相机
        if (isLinkedToCamera)
        {
            Camera.main.transform.Translate(movement);
        }

        // 循环
        if (isLooping)
        {
            // 获取第一个对象
            // 该列表的顺序是从左往右(基于x坐标)
            Transform firstChild = backgroundPart.FirstOrDefault();

            if (firstChild != null)
            {
                // 检查子集对象(部分)是否在摄像机前已准备好.
                // We test the position first because the IsVisibleFrom
                // method is a bit heavier to execute.
                if (firstChild.position.x < Camera.main.transform.position.x)
                {
                    // 如果子集对象已经在摄像机的左侧,我们测试它是否完全在外面,以及是否需要被回收.
                    if (firstChild.renderer.IsVisibleFrom(Camera.main) == false)
                    {
                        // 获取最后一个子集对象的位置
                        Transform lastChild = backgroundPart.LastOrDefault();
                        Vector3 lastPosition = lastChild.transform.position;
                        Vector3 lastSize = (lastChild.renderer.bounds.max - lastChild.renderer.bounds.min);

                        // 将被回收的子集对象作为最后一个子集对象
                        // Note: 当前只横向滚动.
                        firstChild.position = new Vector3(lastPosition.x + lastSize.x, firstChild.position.y, firstChild.position.z);

                        // 将被回收的子集对象设置到backgroundPart的最后位置.
                        backgroundPart.Remove(firstChild);
                        backgroundPart.Add(firstChild);
```

```
                    }
                }
            }
        }
    }
}
```

在Start方法里，使用了LINQ将它们按X轴进行排序。

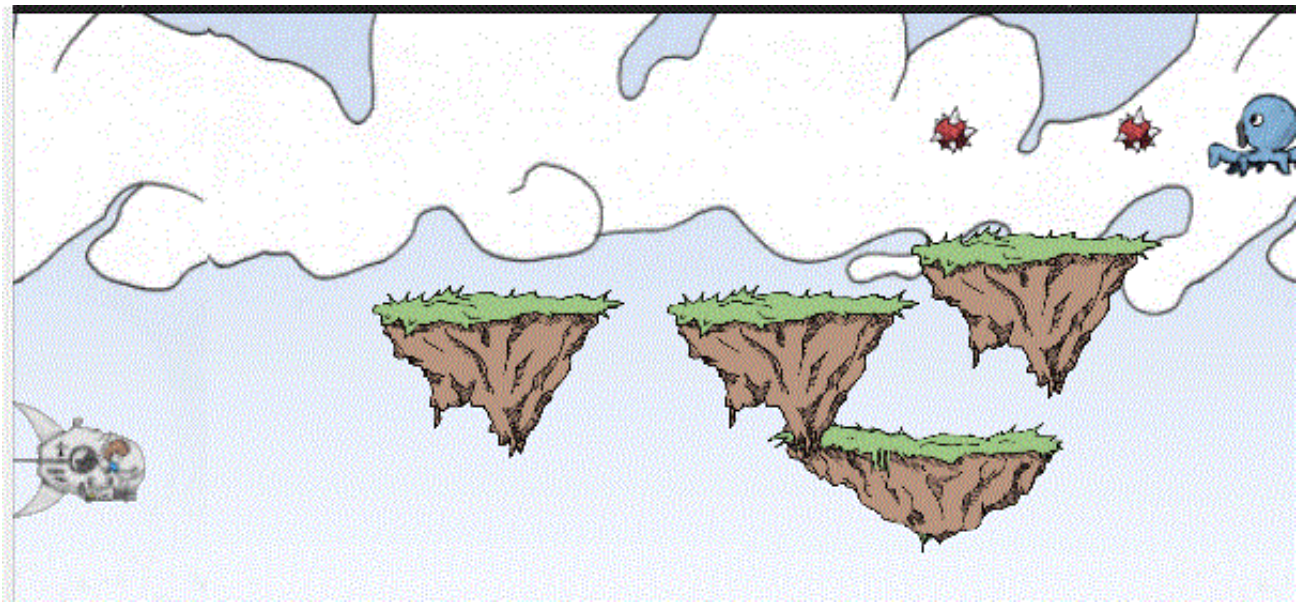
将这个脚本附加到以下对象上，并且设置好其属性值，如下：

Layer	Speed	Direction	Linked to Camera
0 - Background	(1, 1)	(-1, 0, 0)	No
1 - Middleground	(2.5, 2.5)	(-1, 0, 0)	No
Player	(1, 1)	(1, 0, 0)	Yes

现在添加更多的元素到场景上：

- 添加两个天空背景到0 - Background
- 添加一些平台到2 - Middleground
- 添加更多的敌人到3 - Foreground，放置在摄像机的右边

将"0 - Background"对象里的"ScrollingScript"组件的"Is Looping"属性勾选，现在运行，就可以看到视差卷轴的效果，如下图所示：



接下来，修改敌人脚本，让敌人静止不动，且无敌，直到摄像机看到它们。另外，当它们移出屏幕时，则立刻移除它们。修改"EnemyScript"脚本，代码为如下：

```
using UnityEngine;
```

```
using System.Collections;

/// <summary>
/// 敌人通用行为
/// </summary>
public class EnemyScript : MonoBehaviour
{
    #region 变量

    /// <summary>
    /// 是否登场
    /// </summary>
    private bool hasSpawn;
    private MoveScript moveScript;
    private WeaponScript weapon;

    #endregion

    void Awake()
    {
        // 只检索一次武器
        weapon = GetComponent<WeaponScript>();
        // 当未登场的时候检索脚本以禁用
        moveScript = GetComponent<MoveScript>();
    }

    // Use this for initialization
    void Start()
    {
        hasSpawn = false;

        // 禁止一切
        // -- 碰撞机
        collider2D.enabled = false;
        // -- 移动
        moveScript.enabled = false;
        // -- 射击
        weapon.enabled = false;
    }

    // Update is called once per frame
    void Update()
    {
        // 检查敌人是否登场
```

```
        if (hasSpawn == false)
        {
            if (renderer.IsVisibleFrom(Camera.main))
            {
                Spawn();
            }
        }
        else
        {
            // 自动开火
            if (weapon != null && weapon.enabled && weapon.CanAttack)
            {
                weapon.Attack(true);
            }

            // 超出摄像机视野,则销毁对象
            if (renderer.IsVisibleFrom(Camera.main) == false)
            {
                Destroy(gameObject);
            }
        }
    }

    /// <summary>
    /// 激活自身
    /// </summary>
    private void Spawn()
    {
        hasSpawn = true;

        // 启用一切
        // -- 碰撞机
        collider2D.enabled = true;
        // -- 移动
        moveScript.enabled = true;
        // -- 射击
        weapon.enabled = true;
    }
}
```

在游戏过程中，可以发现主角并不是限制在摄像机区域内的，可以随意离开摄像机，现在来修复这个问题。打开"PlayerScript"脚本，在Update方法里面添加如下代码：

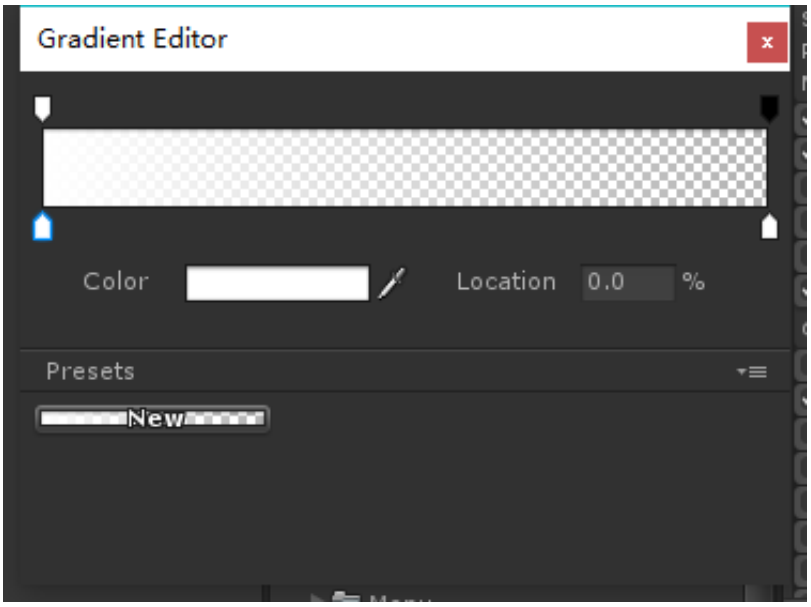
```
        #region 确保没有超出摄像机边界

        var dist = (transform.position - Camera.main.transform.position).z;
        var leftBorder = Camera.main.ViewportToWorldPoint(new Vector3(0, 0, dist)).x;
        var rightBorder = Camera.main.ViewportToWorldPoint(new Vector3(1, 0, dist)).x;
        var topBorder = Camera.main.ViewportToWorldPoint(new Vector3(0, 0, dist)).y;
        var bottomBorder = Camera.main.ViewportToWorldPoint(new Vector3(0, 1, dist)).y;

        transform.position = new Vector3(
          Mathf.Clamp(transform.position.x, leftBorder, rightBorder),
          Mathf.Clamp(transform.position.y, topBorder, bottomBorder),
          transform.position.z
        );

        #endregion
```

6.粒子效果

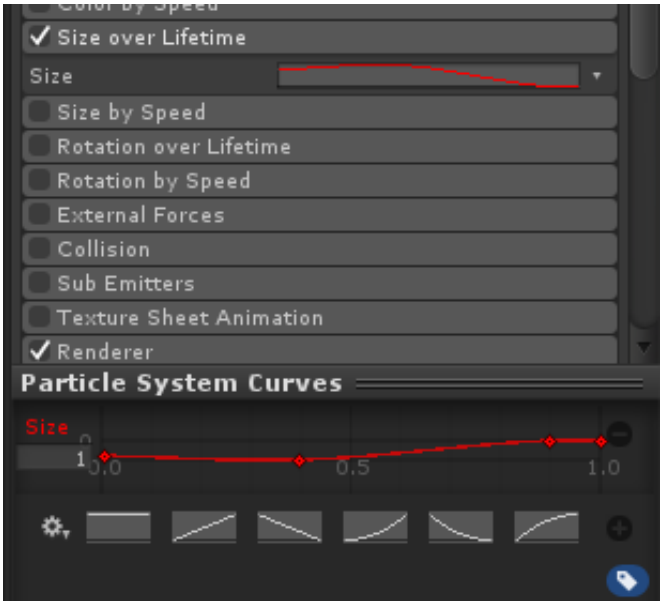
制作一个爆炸的粒子，用于敌人或者主角被摧毁时进行显示。创建一个"Particle System"，导入烟图片到"Textures"文件夹，改变其"Texture Type"为"Texture"，并且勾选"Alpha Is Transparent"属性，附加这个纹理到粒子上，将其拖动到粒子对象上，更改其Shader为"Particles"→"Alpha Blended"，接着更改一些属性，如下所示：

Category	Parameter name	Value
General	Duration	1
General	Max Particles	15
General	Start Lifetime	1
General	Start Color	Gray
General	Start Speed	3
General	Start Size	2
Emission	Bursts	0 : 15
Shape	Shape	Sphere
Color Over Lifetime	Color	见下图
Size Over Lifetime	Size	见下图

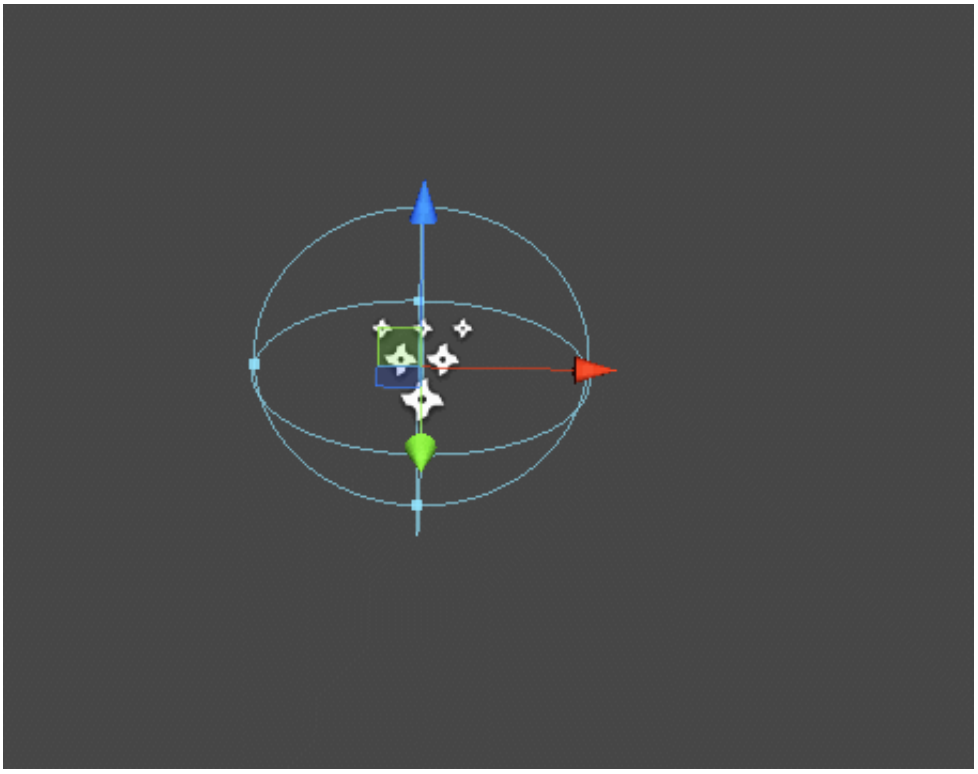
其中Color Over Lifetime要设置成在结束时，有个淡出的效果，如下图所示：



Size Over Lifetime选择一个递减曲线，如下图所示：



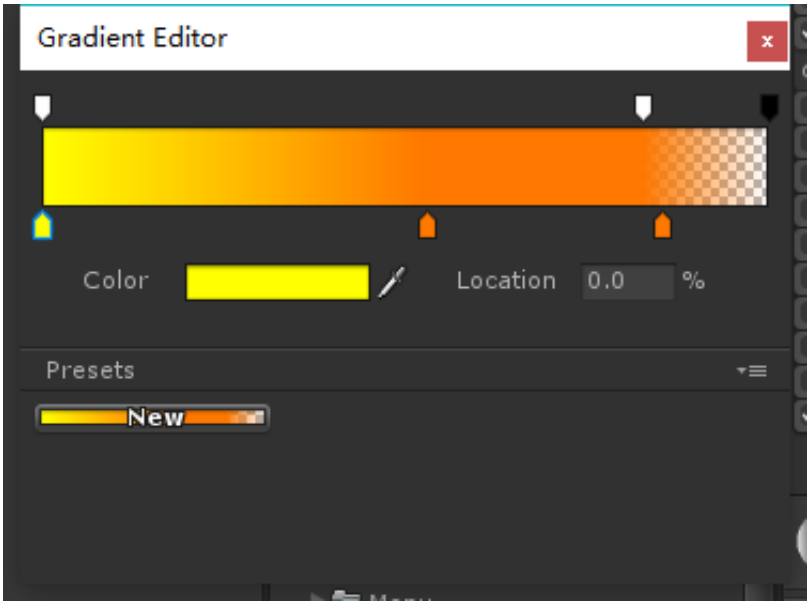
当调整完成后，取消勾选"Looping"，现在粒子效果为如下：



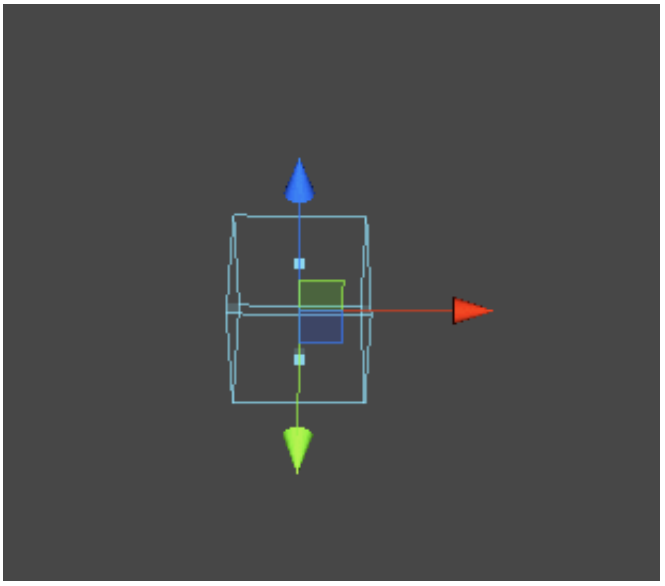
保存成预制，命名为"SmokeEffect"，放在"Prefabs/Particles"文件夹下。现在创建另一个粒子，火焰效果，使用默认材质即可。其他设置如下：

Category	Parameter name	Value
General	Looping	false
General	Duration	1
General	Max Particles	10
General	Start Lifetime	1
General	Start Speed	0.5
General	Start Size	2
Emission	Bursts	0 : 10
Shape	Shape	Box
Color Over Lifetime	Color	见下图

其中Color Over Lifetime要设置成有一个黄色到橙色的渐变，最后淡出，如下图所示：



粒子效果为：



保存成预制，命名为"FireEffect"。创建一个脚本，命名为"SpecialEffectsHelper"，代码如下：

```
using UnityEngine;
using System.Collections;

/// <summary>
/// 从代码中创建粒子特效
/// </summary>
public class SpecialEffectsHelper : MonoBehaviour
{
    /// <summary>
    /// Singleton
    /// </summary>
    public static SpecialEffectsHelper Instance;

    public ParticleSystem smokeEffect;
```

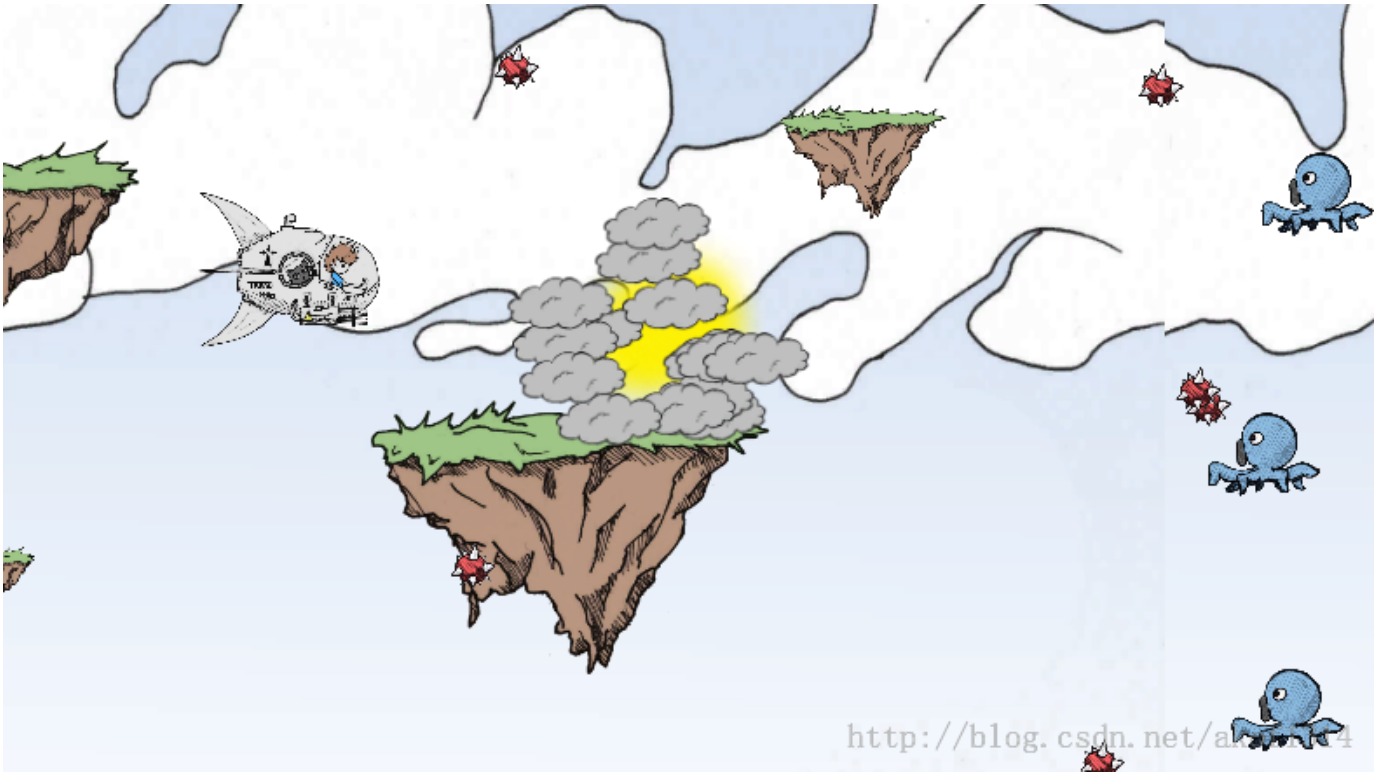
```
    public ParticleSystem fireEffect;

    void Awake()
    {
        // Register the singleton
        if (Instance != null)
        {
            Debug.LogError("Multiple instances of SpecialEffectsHelper!");
        }

        Instance = this;
    }

    // Use this for initialization
    void Start()
    {

    }

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

    }

    /// <summary>
    /// 在给定位置创建爆炸特效
    /// </summary>
    /// <param name="position"></param>
    public void Explosion(Vector3 position)
    {
        // 烟雾特效
        instantiate(smokeEffect, position);
        // 火焰特效
        instantiate(fireEffect, position);
    }

    /// <summary>
    /// 从预制体中实例化粒子特效
    /// </summary>
    /// <param name="prefab"></param>
    /// <returns></returns>
    private ParticleSystem instantiate(ParticleSystem prefab, Vector3 position)
    {
```

```
        ParticleSystem newParticleSystem = Instantiate(prefab, position,
Quaternion.identity) as ParticleSystem;

        // 确保它会被销毁
        Destroy(newParticleSystem.gameObject, newParticleSystem.startLifetime);

        return newParticleSystem;
    }
}
```

这里创建了一个单例，可以让任何地方都可以产生烟和火焰的粒子。将这个脚本附加到"Scripts"对象，设置其属性"Smoke Effect"和"Fire Effect"为对应的预制体。现在是时候调用这个脚本了，打开"HealthScript"脚本文件，在OnTriggerEnter方法里面，更新成如下代码：

```
    /// <summary>
    /// 对敌人造成伤害并检查对象是否应该被销毁
    /// </summary>
    /// <param name="damageCount"></param>
    public void Damage(int damageCount)
    {
        hp -= damageCount;

        if (hp <= 0)
        {
            // 爆炸特效
            SpecialEffectsHelper.Instance.Explosion(transform.position);
            // 播放音效
            SoundEffectsHelper.Instance.MakeExplosionSound();
            // 死亡! 销毁对象!
            Destroy(gameObject);
        }
    }
```

现在运行，射击敌人，可以看到如下效果：



7.游戏音效

现在来添加一些声音。将声音资源放入"Sounds"文件夹，取消勾选每一个声音的"3D sound"属性，因为这是2D游戏。准备播放背景音乐，创建一个游戏对象，命名为"Music"，其Position为(0, 0, 0)，将背景音乐拖到这个对象上，然后勾选"Mute"属性。因为音效总是在一定的时机进行播放，所以创建一个脚本文件，命名为"SoundEffectsHelper"，代码如下：

```
using UnityEngine;
using System.Collections;

/// <summary>
/// 创建音效实例
/// </summary>
public class SoundEffectsHelper : MonoBehaviour
{
    /// <summary>
    /// 静态实例
    /// </summary>
    public static SoundEffectsHelper Instance;

    public AudioClip explosionSound;
    public AudioClip playerShotSound;
    public AudioClip enemyShotSound;

    void Awake()
    {
        // 注册静态实例
```

```
        if (Instance != null)
        {
            Debug.LogError("Multiple instances of SoundEffectsHelper!");
        }
        Instance = this;
    }

    public void MakeExplosionSound()
    {
        MakeSound(explosionSound);
    }

    public void MakePlayerShotSound()
    {
        MakeSound(playerShotSound);
    }

    public void MakeEnemyShotSound()
    {
        MakeSound(enemyShotSound);
    }

    /// <summary>
    /// 播放给定的音效
    /// </summary>
    /// <param name="originalClip"></param>
    private void MakeSound(AudioClip originalClip)
    {
        // 做一个非空判断，防止异常导致剩余操作被中止
        if (Instance.ToString() != "null")
        {
            // 因为它不是3D音频剪辑，位置并不重要。
            AudioSource.PlayClipAtPoint(originalClip, transform.position);
        }
    }
}
```

这里我做了一个非空判断，因为按照原例中的程序，当章鱼死亡在摄像机边界时，会导致Script对象被销毁！从而引发程序异常．我没有找到被销毁的准确原因，所以暂时折中一下，加了个判断，以确保程序能照常运行．如果哪位大神有知道原因的话也欢迎告诉我来更新此文．

将此脚本附加到"Scripts"对象上，然后设置其属性值，如下图所示：

接着，在"HealthScript"脚本文件里，播放粒子效果后面，添加代码：

```
SoundEffectsHelper.Instance.MakeExplosionSound();
```

在"WeaponScript"脚本文件里，Attack方法中，添加代码：

```
            if (isEnemy)
            {
                SoundEffectsHelper.Instance.MakeEnemyShotSound();
            }
            else
            {
                SoundEffectsHelper.Instance.MakePlayerShotSound();
            }
```

现在运行，就可以听到声音了。

8.菜单

创建简单的菜单，以便游戏可以重新开始。导入背景图片和LOGO图片到"Textures"文件夹的子文件夹"Menu"。创建一个新的场景，命名为"Menu"。添加背景Sprite对象，其Position为(0, 0, 1)，Size为(2, 2, 1)。添加LOGO的Sprite对象，其Position为(0, 2, 0)，Size为(0.75, 0.75, 1)。添加一个空对象，命名为"Scripts"，用来加载脚本。现在为这个启动画面，添加一个开始按钮。创建一个脚本文件，命名为"MenuScript"，代码如下：

```
using UnityEngine;
using System.Collections;

/// <summary>
/// Title screen script
/// </summary>
public class MenuScript : MonoBehaviour
{

    // Use this for initialization
    void Start()
    {

    }

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

    }

    void OnGUI()
    {
        const int buttonWidth = 84;
        const int buttonHeight = 60;

        // 在开始游戏界面绘制一个按钮
        if (
            // Center in X, 2/3 of the height in Y
          GUI.Button(new Rect(Screen.width / 2 - (buttonWidth / 2), (2 * Screen.height /
3) - (buttonHeight / 2), buttonWidth, buttonHeight), "开始游戏")
        )
        {
            // On Click, load the first level.
            // "Stage1" is the name of the first scene we created.
            Application.LoadLevel("Stage1");
        }
    }
}
```

将此脚本附加到"Scripts"对象上。现在运行，可以看到如下效果：

但是，点击按钮会崩溃，因为没有将Stage1场景添加进来。打开"File"→"Build Settings"，将场景"Menu"和"Stage1"拖动到上面的"Scenes In Build"里面。再次运行，就可以看到按钮正常切换场景了。当主角被摧毁时，需要可以重新开始游戏。创建一个脚本文件，命名为"GameOverScript"，代码如下：

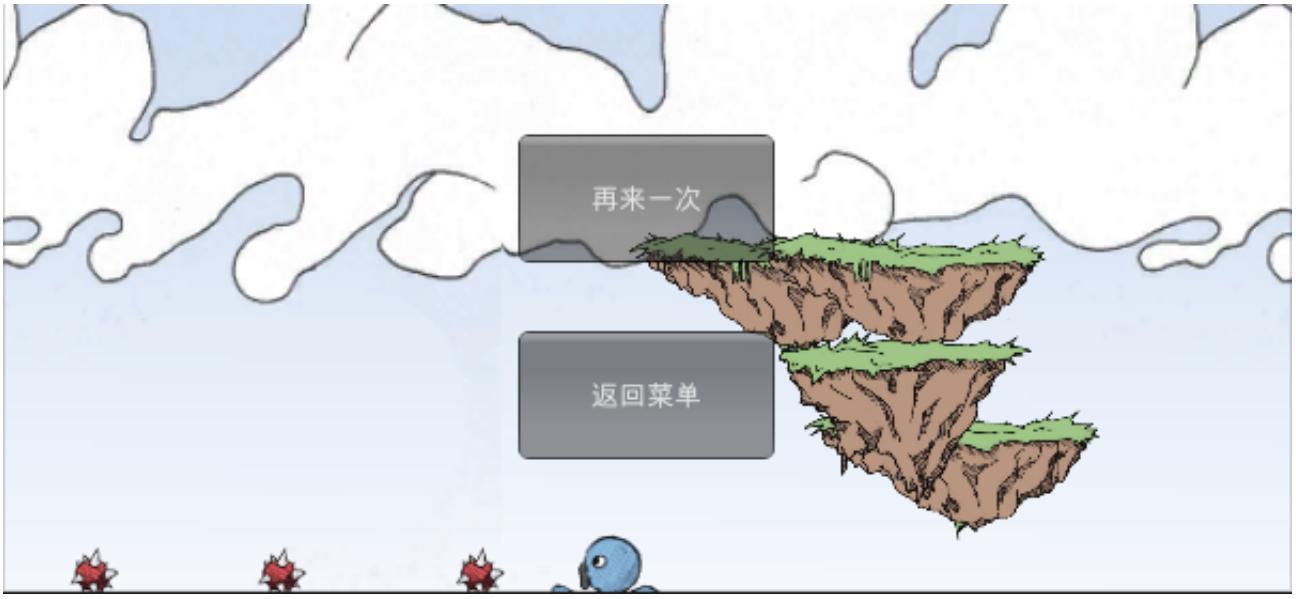
```
using UnityEngine;
using System.Collections;

/// <summary>
/// 开始或退出游戏
/// </summary>
public class GameOverScript : MonoBehaviour
{

    // Use this for initialization
    void Start()
    {

    }

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

    }

    void OnGUI()
    {
        const int buttonWidth = 120;
        const int buttonHeight = 60;
```

```
        // 在x轴中心，y轴1/3处创建"重来"按钮
        if (GUI.Button(new Rect(Screen.width / 2 - (buttonWidth / 2), (1 * Screen.height
/ 3) - (buttonHeight / 2), buttonWidth, buttonHeight), "再来一次"))
        {
            // 重新加载游戏场景
            Application.LoadLevel("Stage1");
        }

        // x轴中心，y轴2/3处创建"返回菜单"按钮
        if (GUI.Button(new Rect(Screen.width / 2 - (buttonWidth / 2), (2 * Screen.height
/ 3) - (buttonHeight / 2), buttonWidth, buttonHeight), "返回菜单"))
        {
            // 加载菜单场景
            Application.LoadLevel("Menu");
        }
    }
}
```

在主角死亡的时候，调用这个脚本。打开"PlayerScript"文件，添加如下代码：

```
    void OnDestroy()
    {
        // Game Over.
        // Add the script to the parent because the current game
        // object is likely going to be destroyed immediately.
        transform.parent.gameObject.AddComponent<GameOverScript>();
    }
```

现在运行，当死亡时，就会出现按钮，如下图所示：



9.代码创建平台岛屿和敌人

现在，一个简单的横版射击小游戏已经有雏形了，然后手动创建有限的岛屿和敌人毕竟有耗尽的时候.这个时候我们可以在代码中动态的去创建敌人和岛屿，这样只要玩家还存活，就会一直有敌人出现，有点无尽版的意思.

创建一个脚本文件，命名为"MakePlatformScript"，代码如下：

```
using UnityEngine;
using System.Collections;

/// <summary>
/// 制造平台
/// </summary>
public class MakePlatformScript : MonoBehaviour
{
    /// <summary>
    /// 平台预设体1
    /// </summary>
    public Transform platform1Prefab;
    /// <summary>
    /// 平台预设体2
    /// </summary>
    public Transform platform2Prefab;

    // Use this for initialization
    void Start()
    {

    }
```

```
    // Update is called once per frame
    void Update()
    {
        if (transform.childCount < 4)
        {
            if (Random.Range(0, 2) > 0)
            {
                CreatePlatform1();
            }
            else
            {
                CreatePlatform2();
            }
        }
    }

    void CreatePlatform1()
    {
        var platformTransform = Instantiate(platform1Prefab) as Transform;

        platformTransform.position = new Vector3(Camera.main.transform.position.x +
Random.Range(14, 23), Random.Range(-3, 3), 5);
        platformTransform.transform.parent = transform;
    }

    void CreatePlatform2()
    {
        var platformTransform = Instantiate(platform2Prefab) as Transform;

        platformTransform.position = new Vector3(Camera.main.transform.position.x +
Random.Range(14, 23), Random.Range(-3, 3), 5);
        platformTransform.transform.parent = transform;
    }
}
```

将MakePlatformScript附加到1 - Middleground, 设置它的预制体为对应的平台岛屿预制体.

接着继续创建一个脚本文件，命名为"MakeEnemyScript"，代码如下：

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

```
/// <summary>
/// 制造敌人
/// </summary>
public class MakeEnemyScript : MonoBehaviour
{
    /// <summary>
    /// 敌人预设体
    /// </summary>
    public Transform enemyPrefab;

    // Use this for initialization
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (transform.childCount < 2)
        {
            CreateEnemy();
        }
    }

    /// <summary>
    /// 创建敌人
    /// </summary>
    void CreateEnemy()
    {
        var enemyTransform = Instantiate(enemyPrefab) as Transform;
        enemyTransform.position = new Vector3(Camera.main.transform.position.x + 15,
Random.Range(-4, 4), 0);
        enemyTransform.transform.parent = transform;
        MoveScript move = enemyTransform.gameObject.GetComponent<MoveScript>();
        if (move != null)
        {
            move.direction.x = -1f;
            move.speed = new Vector2(3, 3);
        }
    }
}
```

将MakeEnemyScript附加到2 - Foreground，同样，设置对应的预制体为章鱼.

现在再次运行，就会看到系统自动创建的章鱼敌人和漂浮岛屿了!

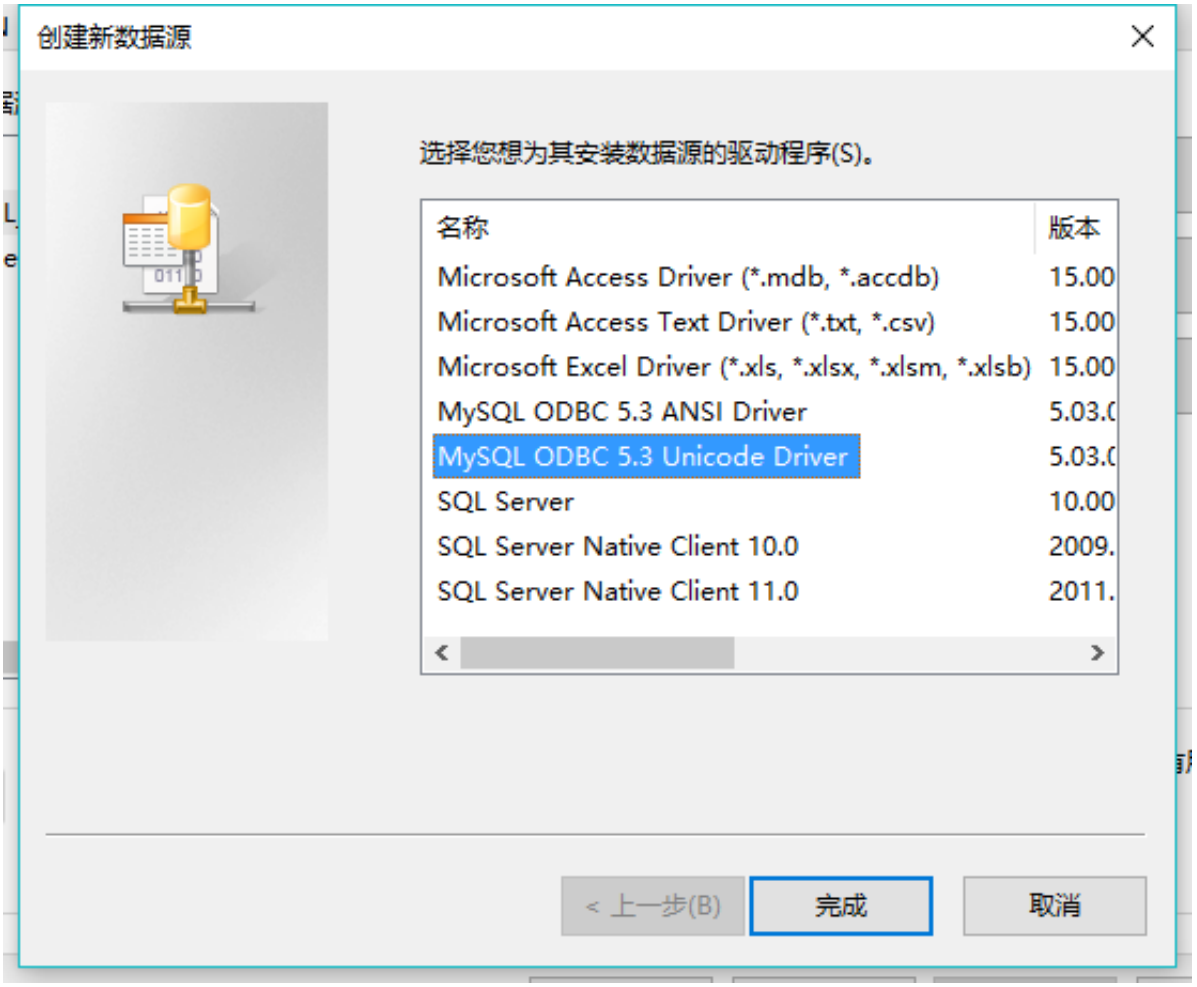
到这里，整个游戏就完成了，推荐大家可以去页首找原文链接去看看，老外的原文解说的更详细，我也是在原文的基础上自己再次整理.

最后附上程序源码，文中有纰漏或看不太明白的地方大家可以对照着源码一起看，欢迎留言指教.

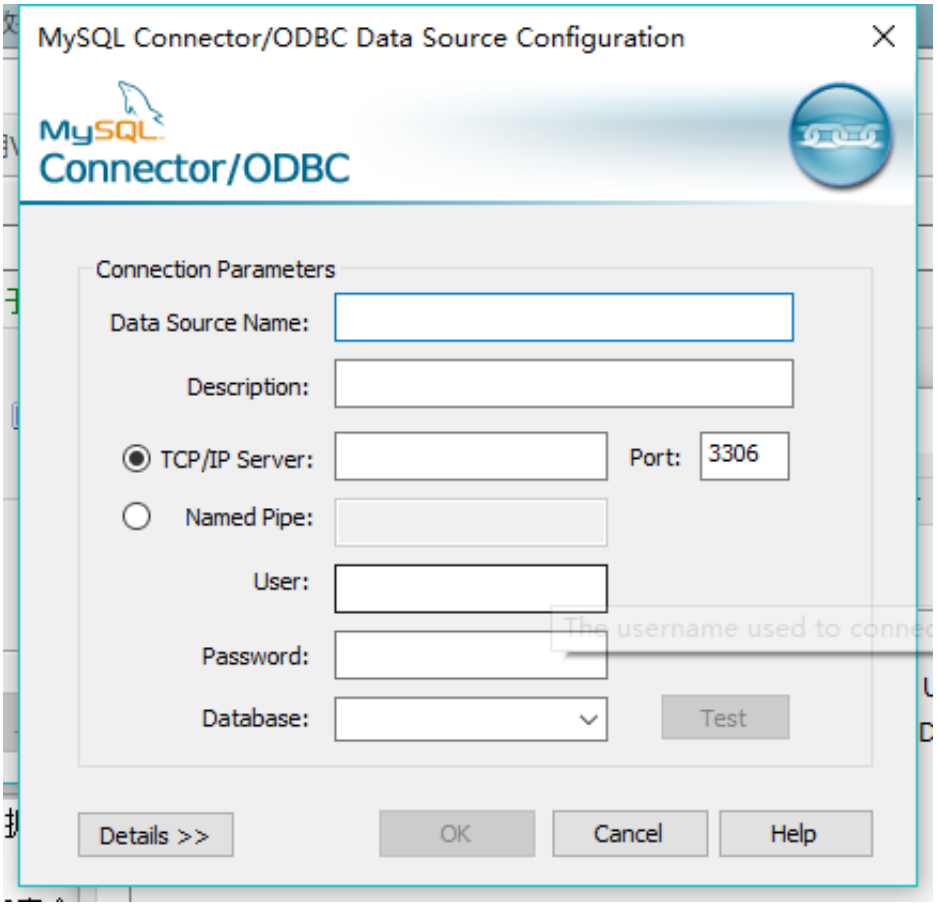
源代码地址： http://pan.baidu.com/s/1b51XmQ

- 能加个QQ不。新手上路。。16769711
- --曲达子
- @曲达子没懂你什么意思...unity直接打开不就好了...
-
- @姑苏慕容复楼主我想问下。我下载了你的源码C#部分现在可以打开了。u3d部分我想问下杂个在untiy中加载出来。。感谢 。...
- --曲达子
- @姑苏慕容复哇哈哈~感谢！...
- --二十二号同学
- @二十二号同学Unity版本4.6.3，5.0没敢用，毕竟还是菜鸟，5里面有一些问题目前不太好解决，资料也没4+的多，所以初学者我觉得还是4+版本开始学好.UI插件楼主也没有，也是刚开始学.....
-

一. 安装MySQL ODBC驱动
为MySQL安装Connector/ODBC驱动。在此需要注意的一点是Connector/ODBC驱动与MySQL Server的版本对应问题。
二. 创建系统DSN
DSN为ODBC定义了一个确定的数据库和必须用到的ODBC驱动程序。每个ODBC驱动程序定义为该驱动程序支持的一个数据库创建DSN需要的信息。
创建系统DSN步骤如下：
开始->设置->控制面板->管理工具->数据源（ODBC），按该流程打开ODBC数据源管理器。
切换至系统DSN选项卡，点击添加按钮。弹出创建新数据源对话框，选择MySQL ODBC 5.3 Unicode Driver驱动程序(根据版本不同，可能名称有出入)，点击完成按钮。



在弹出的链接MySQL对话框中设置MySQL数据库帐号信息。



关键是Login选项卡下的几个参数。

1　Data Source Name，这个根据命名规则任意命名就可以了，最终会显示于ODBC数据源管理器中系统

DSN选项卡下的列表中。

1 Description，对该数据源的描述，可不填写。

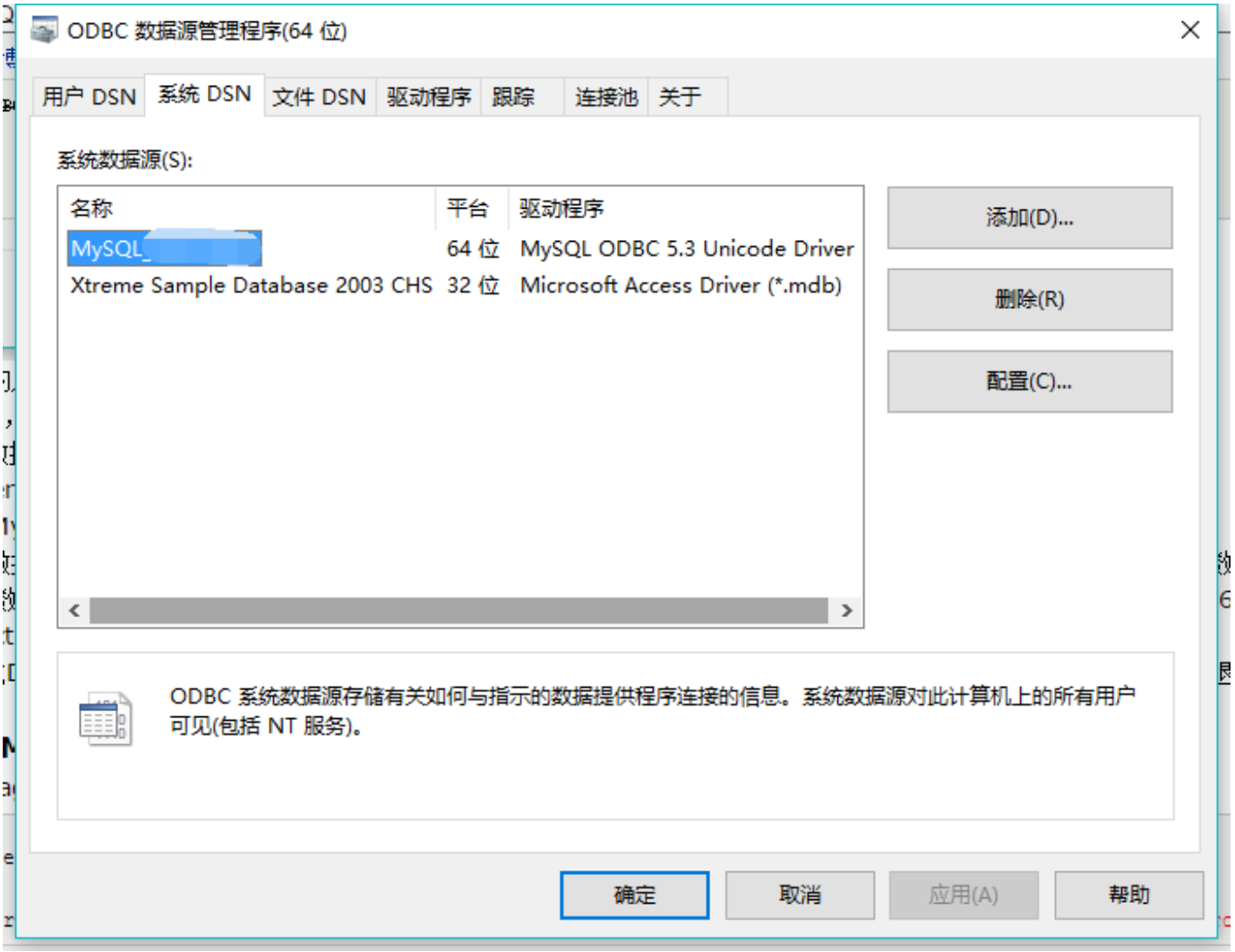
1 Server，MySQL Server的主机名，这里填写计算机主机名或者localhost均可。

1 User和Password是MySQL Server对应的用户名和密码。

1 DataBase，选定该数据源所指向的数据库。在这一里必须要求前面几个参数都正确，否则会提示错误，无法选择MySQL Server中的数据库。

还有两个需要注意的参数是Connect Options选项卡下的Port和Character Set。Port用于设置MySQL Server的通信端口，默认是3306，在安装时候如果没有改动默认端口，这里可以不设置。Character Set用于设置数据库语言编码，这里选择gbk。

点击OK按钮，完成系统DSN的创建，返回到ODBC数据源管理器对话框，在系统DSN选项卡下可查看到刚建立的数据源。点击确定按钮退出。

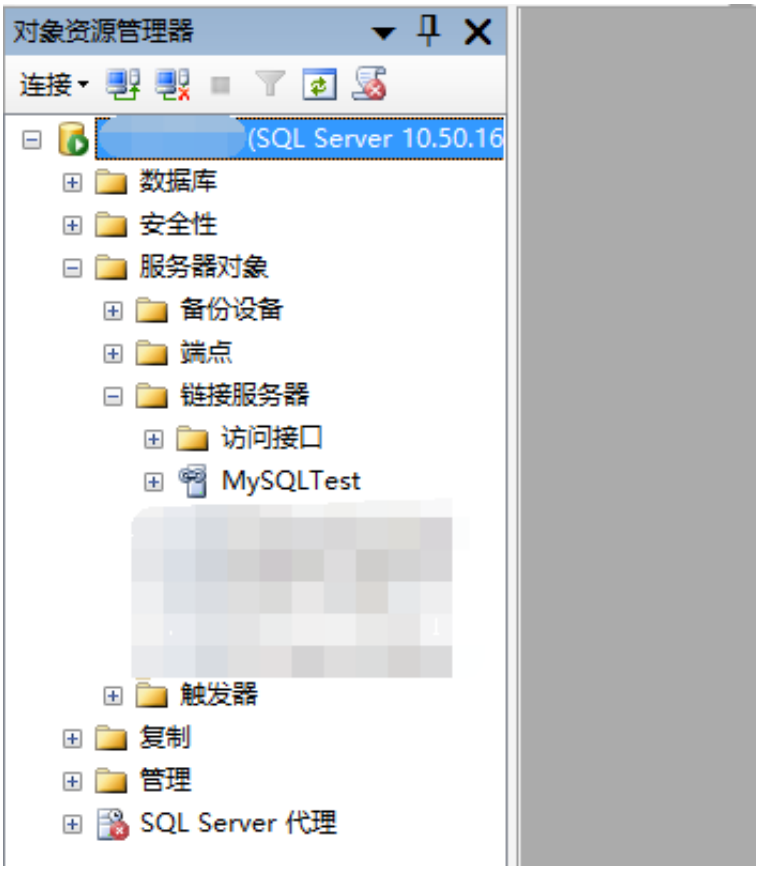


三．创建MSSQL到MySQL的链接服务

打开SQL Server Management Studio，运行下述语句，通过前面新建的ODBC数据源建立与MySQL Server链接服务器。

```
EXEC sp_addlinkedserver  @server = 'MySQLTest', @srvproduct='MySQL', @provider =
'MSDASQL', @datasrc = '数据库名称'
GO
EXEC sp_addlinkedsrvlogin  @rmtsrvname='MySQLTest', @useself='false',
@locallogin='sa',@rmtuser='root',@rmtpassword='root用户密码'
GO
```

刷新下链接服务器节点，既可以看到上述用语句创建的链接。



四. 利用SQL语句转移数据至MSSQL

在Microsoft SQL Server中创建新的数据库，运行如下语句，运行后就可以把MySQL 数据库“tigerdb” 导入到 Microsoft SQL 数据库“testMySQl”中。

```
SELECT * INTO 数据库.dbo.表
FROM OPENQUERY (MySQLtest ,'select * from 数据库.dbo.表' )
```

如此便可完成从MySQL Server到MS SQL Server的数据库移植。PS：结尾再啰嗦两句废话，昨晚开始用的MySQL，从安装到使用，感觉还是有点烦的，视图化操作习惯了， 突然用命令行感觉好麻烦。不得不感叹一下微软的封装做的确实是好，把用户习惯培养了起来。 对于一个合格的it人员来说， 大多数微软的软件完全可以称得上是傻瓜式操作，不看说明全凭自己摸索都可以完美运行。