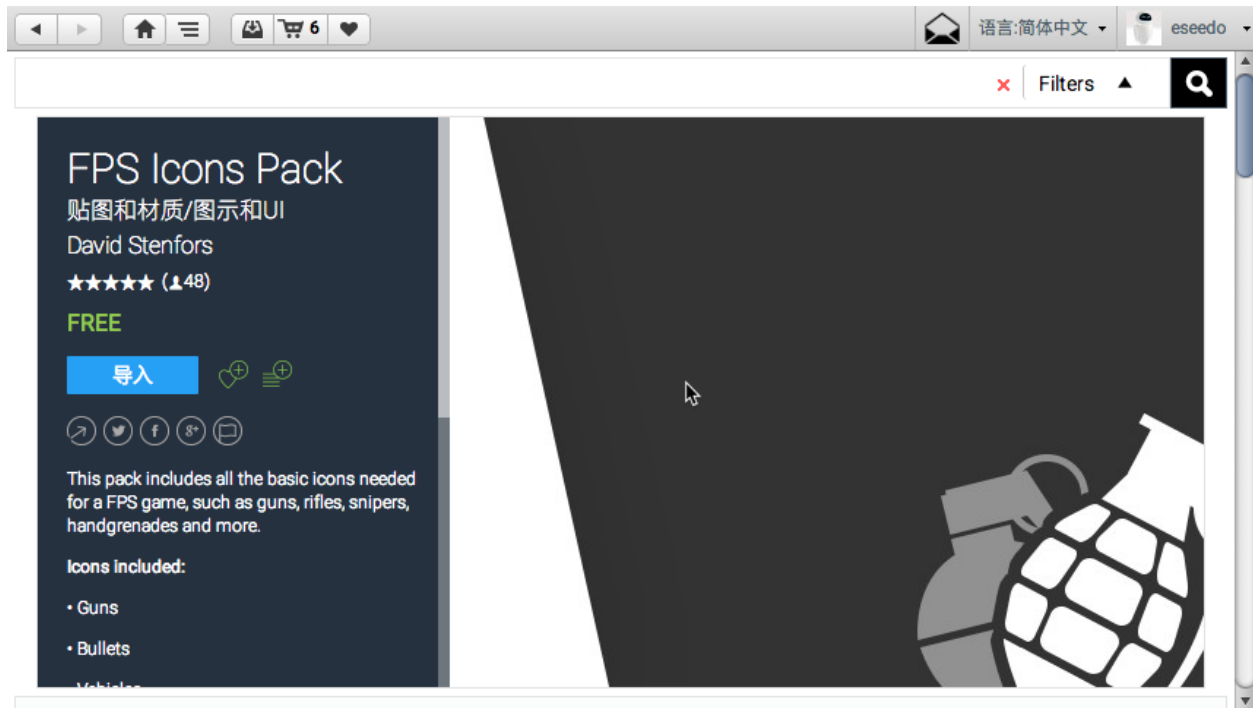


欢迎回到我们的学习。

在本节课程中，我们将给玩家设置生命值。

首先让我们导入一些icon资源。

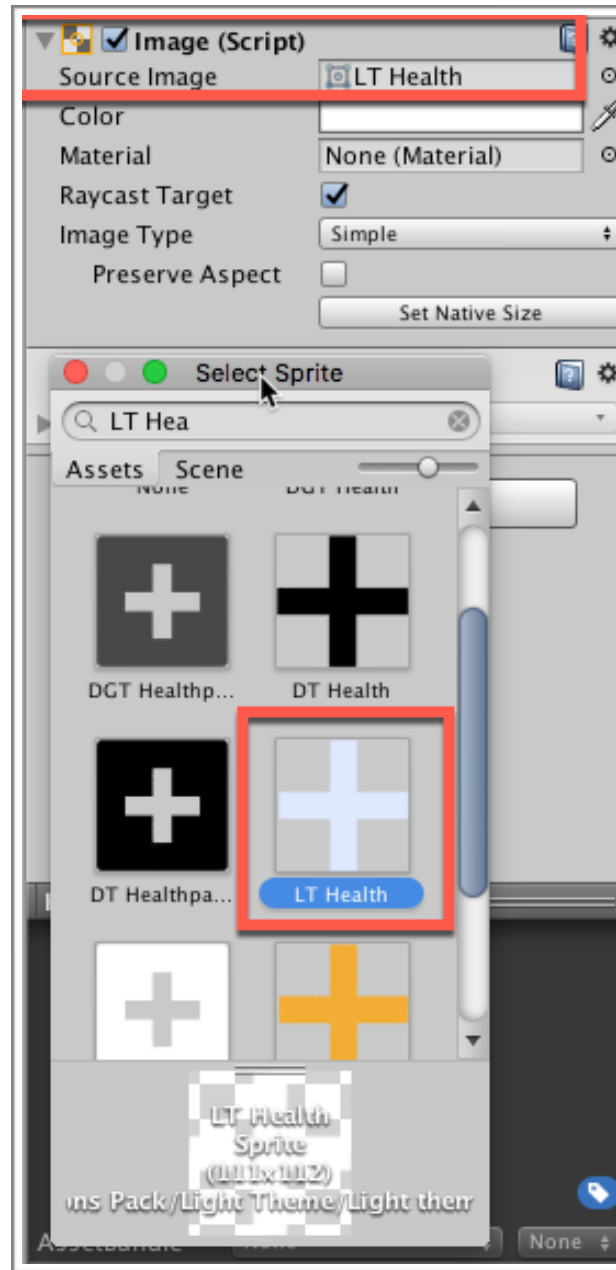
在Unity编辑器中打开Asset Store，搜索fps icons，然后找到下面的这个插件。



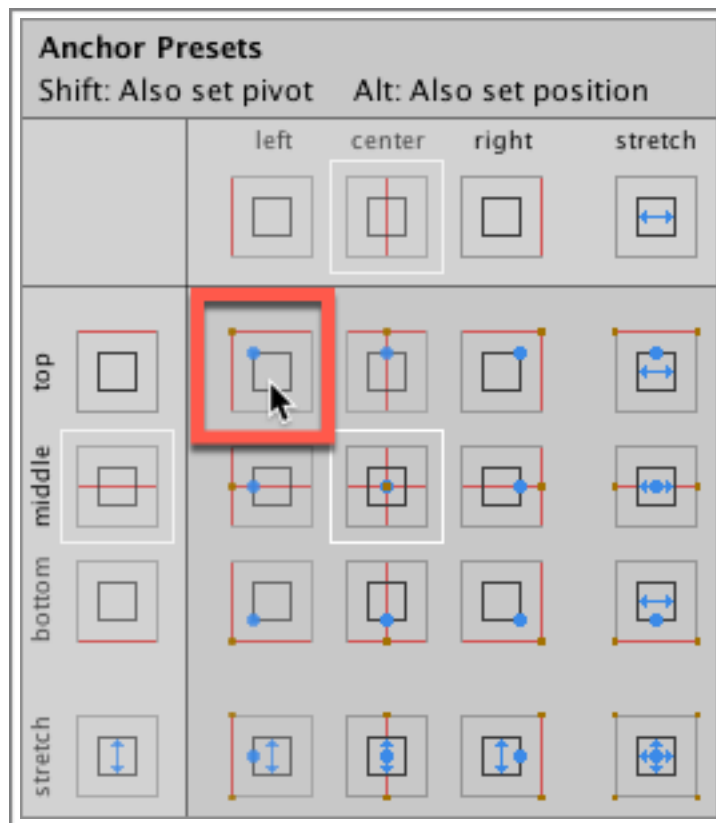
下载并将其导入到项目之中。

然后把下载之后的FPS Icons Pack拖到Project视图的Arts目录中。

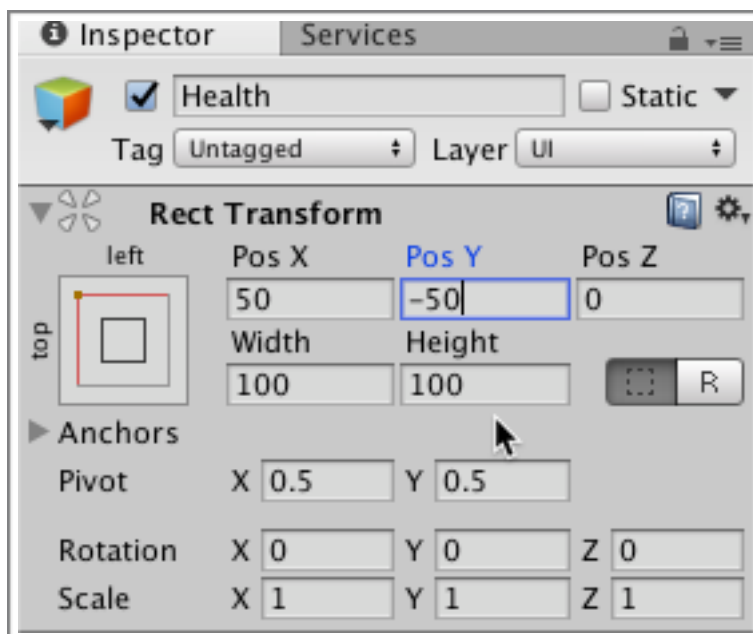
接下来在Hierarchy视图中右键单击Canvas对象，创建一个新的Image，并将其更名为Health，然后在Inspector面板中设置Source Image为LT Health，如下图。



接下来在Inspector面板中设置Health这个UI元素的锚点为Top Left,如下图



如果此时十字标志没有自动切换到Game视图的左上角，那么我们需要手动设置 Rect Transform的属性如下：



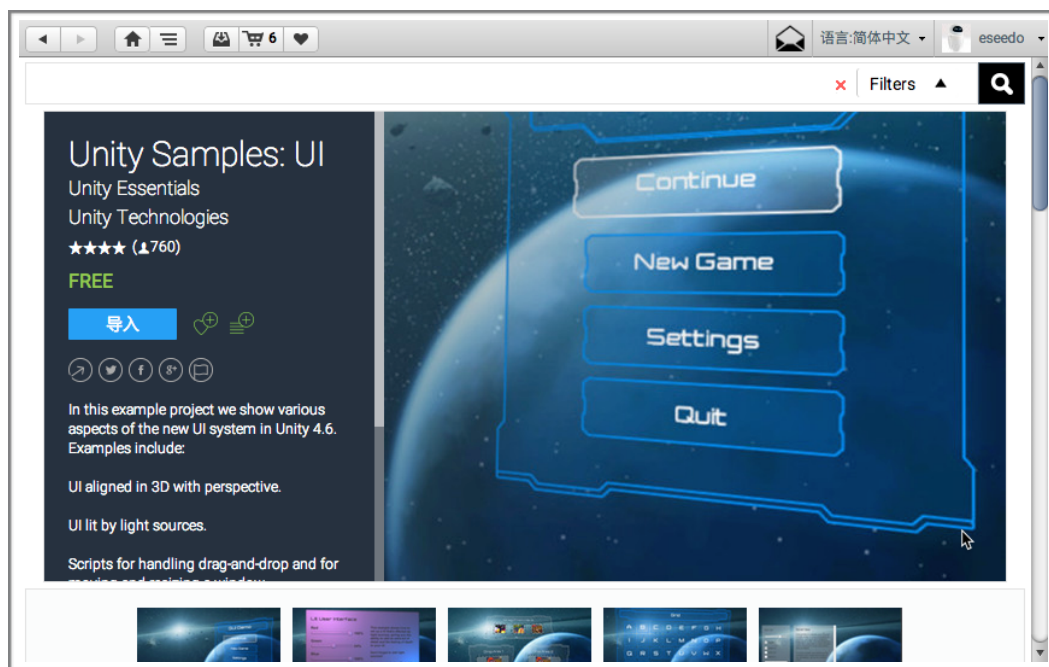
此外我们还需要给Health元素添加文字说明，

在Hierarchy视图中右键单击Health，选择UI-Text，然后在Inspector面板中做以下设置：

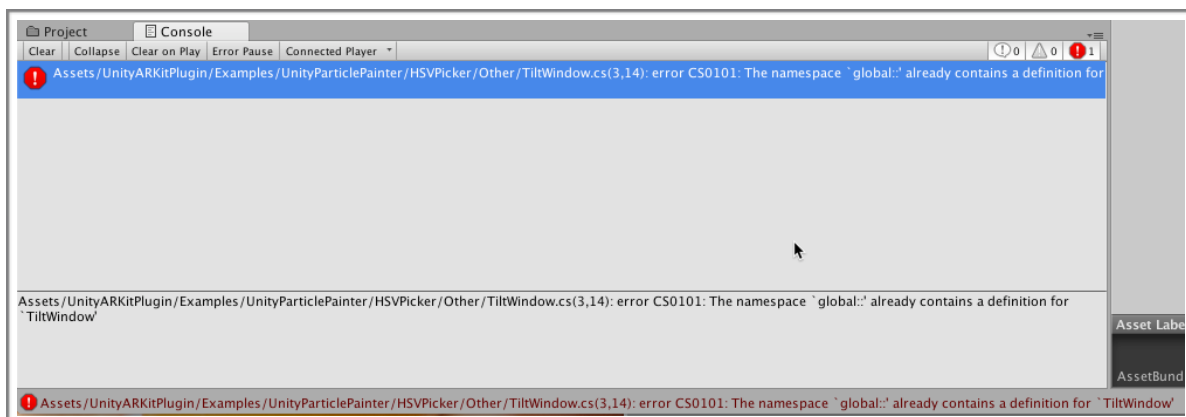
- (1) 设置Text初始内容为100
- (2) 设置Rect Transform中的Width和Height 为250, 200
- (3) 设置Color为纯白色
- (4) 设置Font Size为130
- (5) 设置Rect Transform中的Pos X和Pos Y属性, 使得文本显示在合适的位置

但是现在默认的字體不是很美观, 我们希望设置一种全新的字體。

在Unity中重新切换到Asset Store选项卡, 搜索ui,并找到Unity Samples:UI这个插件。



但是遗憾的是, 当这个插件导入后, 在Console中立即出现了错误提示。

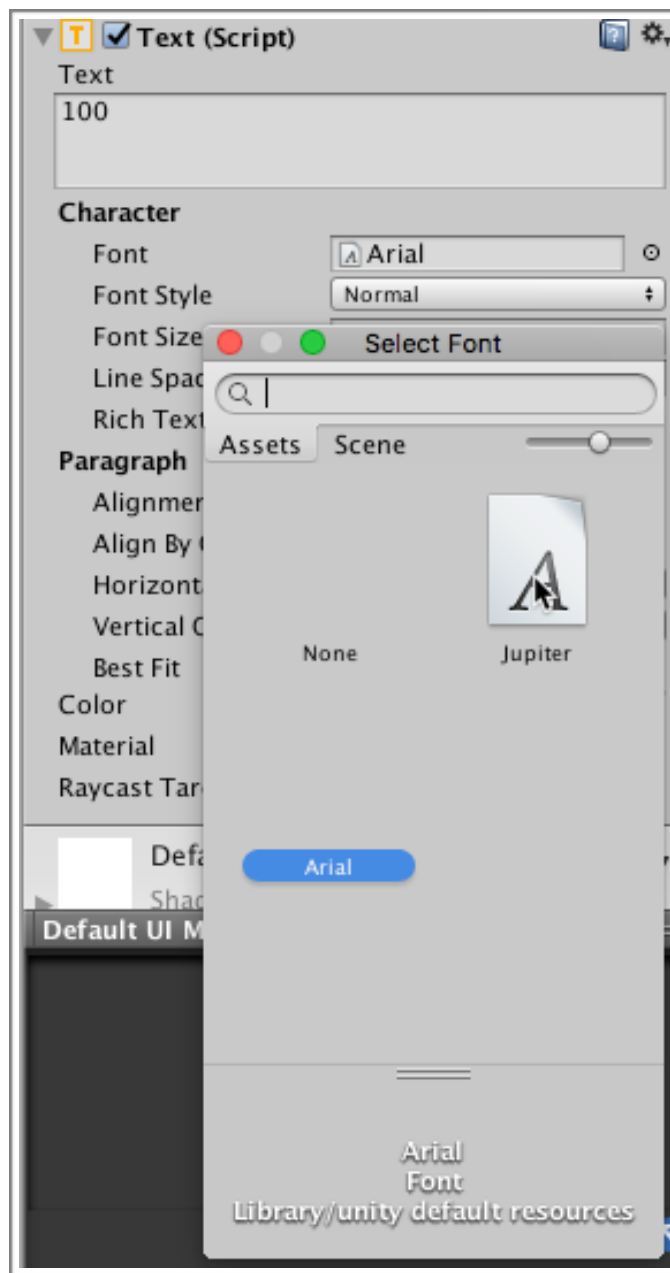


此时我们需要将导入的插件的Scripts文件夹删除。

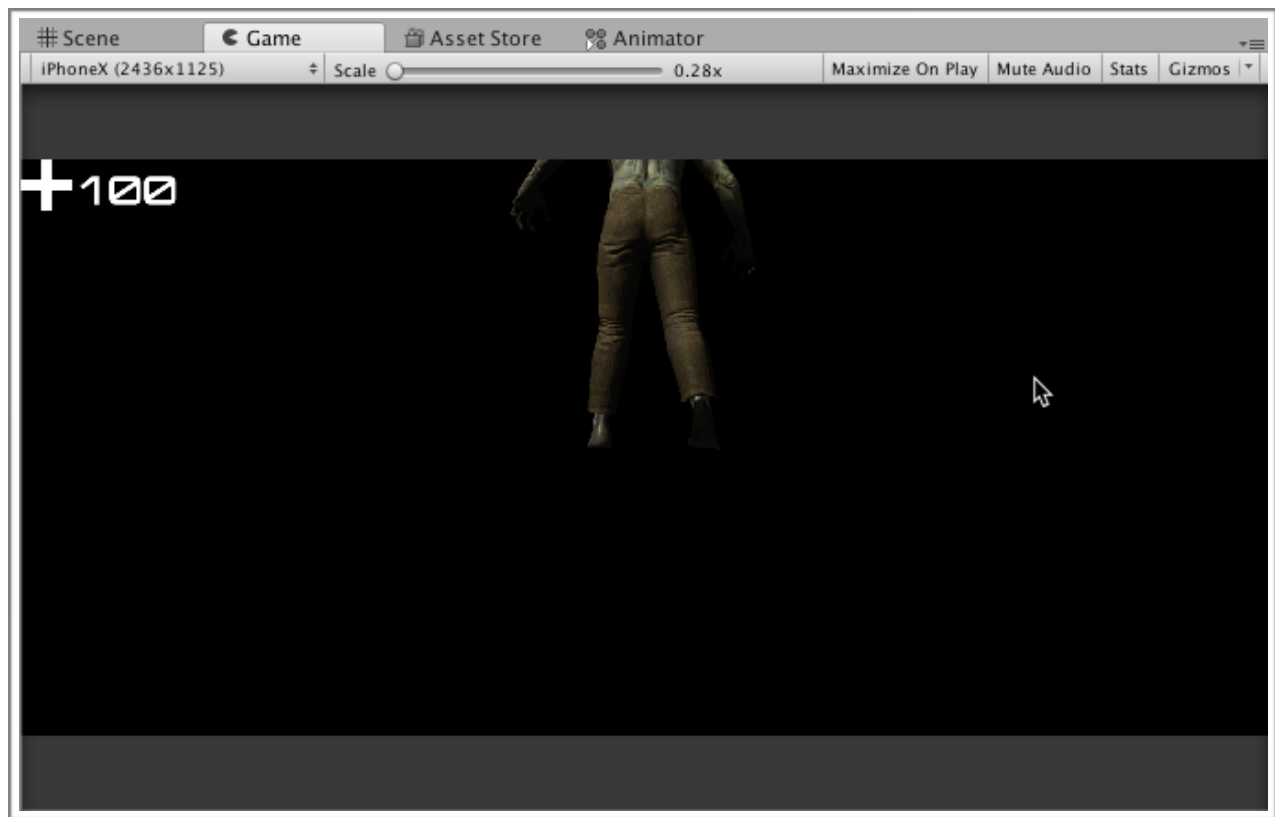
这也是为什么我个人习惯使用_Scripts来命名项目中自己的脚本文件夹，因为这样就不会跟所导入的第三方插件混在一起了。

删除了Scripts文件夹之后（注意不是我们自己创建的_Scripts文件夹！！！），一切恢复正常了。

在Hierarchy视图中选择Text对象，然后设置Font类型为刚刚添加的Jupiter。



此时Game视图中应该如下所示，当然具体的细节可以微调。



接下来就是要实现相关的游戏逻辑了。

在Hierarchy视图中选择GameController，打开GameControllerScript.cs文件，并编辑其中的代码如下：

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

//1.导入UI相关的文件
using UnityEngine.UI;

public class GameControllerScript : MonoBehaviour {

    //添加到bloodyScreen对象的引用
    public GameObject bloodyScreen;

    //2.添加到healthText对象的引用
    public Text healthText;
```

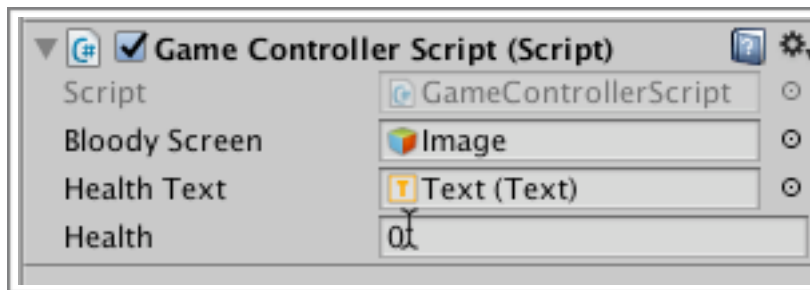
```
//3. 玩家的生命值
public int health;
.....
```

省略号代表后面的代码并未做任何变动。

这里我们还是按照注释行的数字编号来解释。

- (1) 这里我们首先要导入Unity中用来处理UI的命名空间
- (2) 这里添加了到UI元素healthText对象的引用
- (3) 定义了玩家的生命值。

接下来回到Unity主编辑器，在GameController对象的Game Controller Script 组件处，设置Health Text的属性为刚刚创建的Text UI元素，如图所示。



需要注意的是，有时候返回Unity主编辑器的时候并没有看到新出现的属性，这个时候就需要在visual studio中使用快捷键command +B手动编译一次（可能是Unity某些版本的bug）。

然后回到GameControllerScript.cs文件，继续修改其中的代码。

首先我们要更改其中的Start方法，并设置玩家的初始生命值为100。

```
void Start () {
    //4. 设置玩家的初始生命值为100
    health = 100;
}
```

然后在zombieAttack方法体中添加一行代码，使得每次攻击时玩家的生命值减少5点。

同时我们还需要把玩家的生命值转化为字符串，并赋值给healthText对象。

```
public void zombieAttack(bool zombieIsThere){  
  
    //激活bloodyScreen对象  
    bloodyScreen.gameObject.SetActive (true);  
  
    //添加协程，在两秒后禁用bloodyScreen  
    StartCoroutine(WaitTwoSeconds());  
  
    //5.玩家的生命值减少5点  
    health -= 5;  
  
    //6.将玩家生命值转换为字符串  
    string stringHealth = (health).ToString();  
  
    //7.将生命值赋值给healthText文本对象  
    healthText.text = "" + stringHealth;  
  
}
```

接下来要更改Update方法体中的代码，当玩家的生命值减少为0时，我们将加载新的游戏场景，并显示“游戏结束”。

```
void Update () {  
  
    //8.当玩家生命值为0时，切换到游戏结束的场景  
    if (health <= 0) {  
  
        SceneManager.LoadScene ("GameOver");  
  
    }  
  
}
```

当然此时我们还需要在顶部导入场景管理的命名空间。

```
//9.导入场景管理相关的文件  
using UnityEngine.SceneManagement;
```

而且我们还需要创建一个新的GameOver场景。

此时GameControllerScript.cs中的完整代码如下所示：

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

//1.导入UI相关的文件
using UnityEngine.UI;

//9.导入场景管理相关的文件
using UnityEngine.SceneManagement;

public class GameControllerScript : MonoBehaviour {

    //添加到bloodyScreen对象的引用
    public GameObject bloodyScreen;

    //2.添加到healthText对象的引用
    public Text healthText;

    //3.玩家的生命值
    public int health;

    // Use this for initialization
    void Start () {

        //4.设置玩家的初始生命值为100
        health = 100;
    }

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

        //8.当玩家生命值为0时，切换到游戏结束的场景
        if (health <= 0) {

            SceneManager.LoadScene ("GameOver");

        }
    }
}
```

```

    }

    public void zombieAttack(bool zombieIsThere){

        //激活bloodyScreen对象
        bloodyScreen.gameObject.SetActive (true);

        //添加协程，在两秒后禁用bloodyScreen
        StartCoroutine(WaitTwoSeconds());

        //5.玩家的生命值减少5点
        health -= 5;

        //6.将玩家生命值转换为字符串
        string stringHealth = (health).ToString();

        //7.将生命值赋值给healthText文本对象
        healthText.text ="" + stringHealth;

    }

    IEnumerator WaitTwoSeconds(){

        yield return new WaitForSeconds (2f);
        bloodyScreen.gameObject.SetActive (false);

    }

}

```

接下来回到Unity编辑器，保存当前场景。

在菜单栏上选择File-New Scene，创建一个新的场景，保存并将其命名为GameOver。

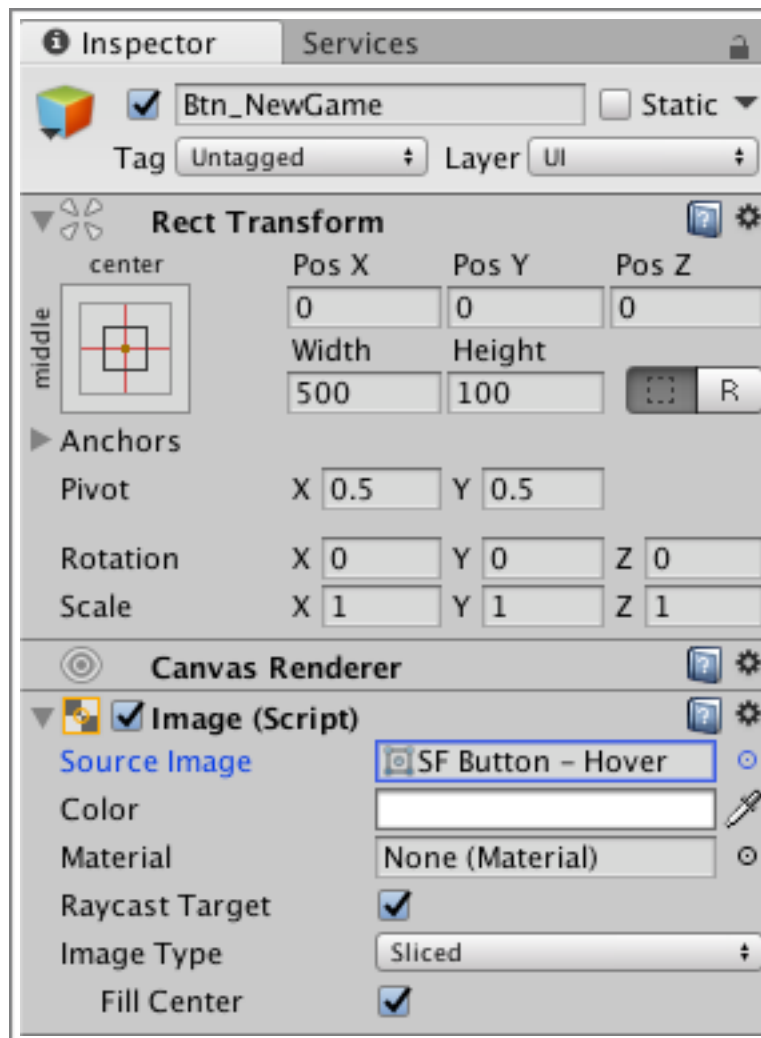
这个游戏场景非常简单，我们只需要创建一个UI元素，显示“Game Over”即可。

在Hierarchy视图中右键单击，选择UI-Text，在Inspector视图中将Text的内容更改为“游戏结束”，并作出以下设置：

- (1) 设置Rect Transform中的Width和Height为800和400，
- (2) 设置Paragraph处的Alignment为居中对齐
- (3) 设置Font Size 为100
- (4) 设置Color为纯白色
- (5) 设置Font为Jupiter

接下来在Text元素之下添加一个按钮，用于返回游戏主场景。
在Hierarchy视图中右键单击Canvas，选择UI-Button，将Button的名称更改为Btn_NewGame，将button的文本内容更改为New Game。

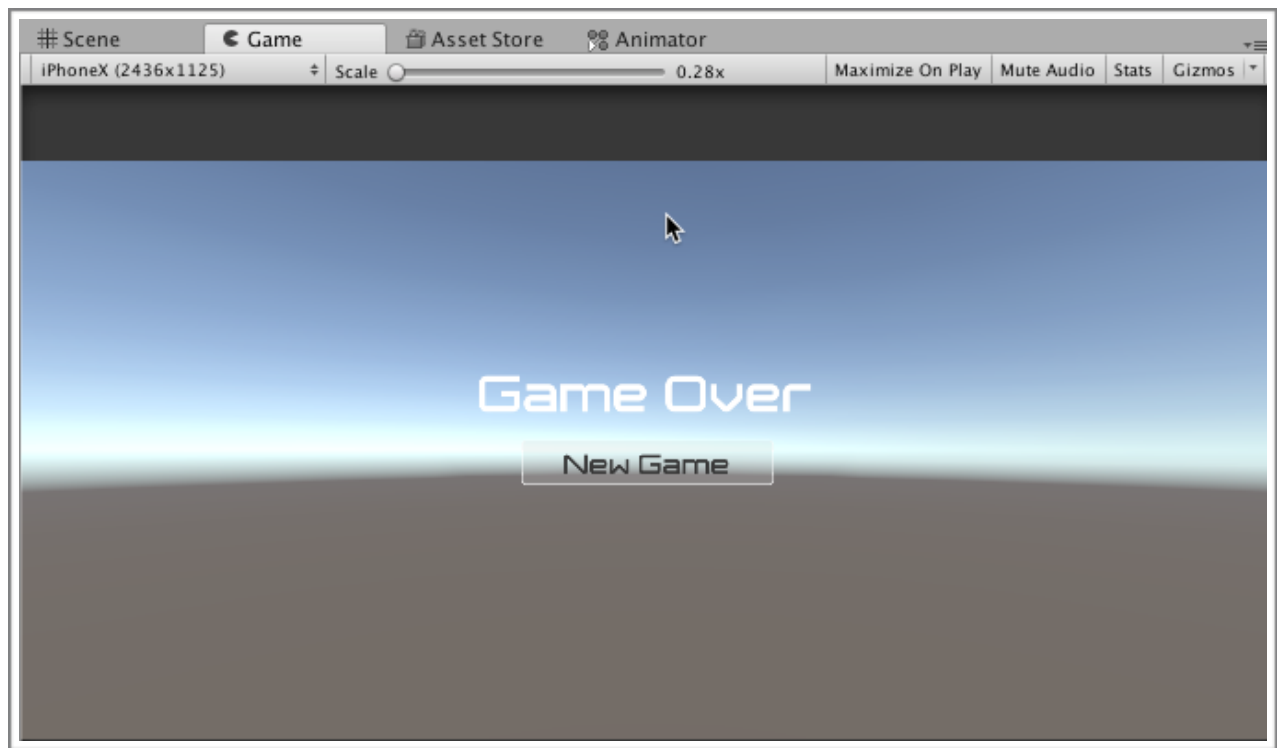
接下来要作出以下设置：



- (1) 设置Btn_NewGame的Rect Transform的Width和Height为500，100
- (2) 设置Btn_NewGame的Image组件的Source image为SF Button- Hover

此外还要设置button的文本的Font Size,Color和Font等属性，具体就不再赘述了。

设置完成后的 Game视图如下图所示：



接下来我们要添加一个脚本，当用户触碰按钮时，返回游戏主场景。

在Hierarchy视图中选择Canvas，在Inspector视图中点击Add Component，添加一个新的脚本，并将其命名为LoadNewGame，双击将其打开。

更改其中的代码如下：

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//1.导入UI命名空间
using UnityEngine.UI;
//2.导入场景管理命名空间
using UnityEngine.SceneManagement;

public class LoadNewGame : MonoBehaviour {

    public Button btnNewGame;

    // Use this for initialization
    void Start () {
```

```

        //3.监听按钮点击事件
        btnNewGame.onClick.AddListener (LoadGame);
    }

    void LoadGame(){

        //4.加载新的场景
        SceneManager.LoadScene ("UnityARKitScene");

    }

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

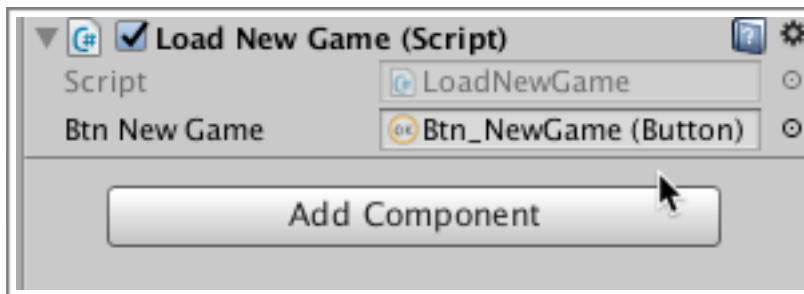
    }

}

```

这里还是按照注释行的数字顺序来简单解释一下。

其中1和2只是导入两个命名空间，第3行代码的作用是添加对按钮点击事件的监听。第4行代码的作用是返回游戏主场景。



完成后回到Unity编辑器，在Canvas的Load New Game组件处，设置Btn New Game为刚刚所添加的Btn_NewGame控件。

保存当前场景。

然后从菜单栏中选择File-Build Settings，点击Add Open Scenes，以添加GameOver这个场景。

最后从Project视图找到UnityARKitScene，点击工具栏上的Play按钮来预览游戏效果。

我们会看到玩家的生命值持续降低，当变成0的时候就会切换到GameOver场景。当点击New Game按钮时又会重新回到游戏主场景。

如果一切顺利，那么我们的游戏逻辑就成功实现了。

当然，初学者很容易漏掉一个小环节，那就是在GameOver场景设置完毕之后，需要在Build Setting中点击Add Open Scenes，从而将场景添加到游戏之中。否则是看不到所期待的画面的~

恭喜你，今天的学习到此结束~