以下内容为雨松 MOMO 原创文章如转载，请注明：转载自雨松 MOMO 的博客

原文地址列表: http://blog.csdn.net/column/details/unity3d.html

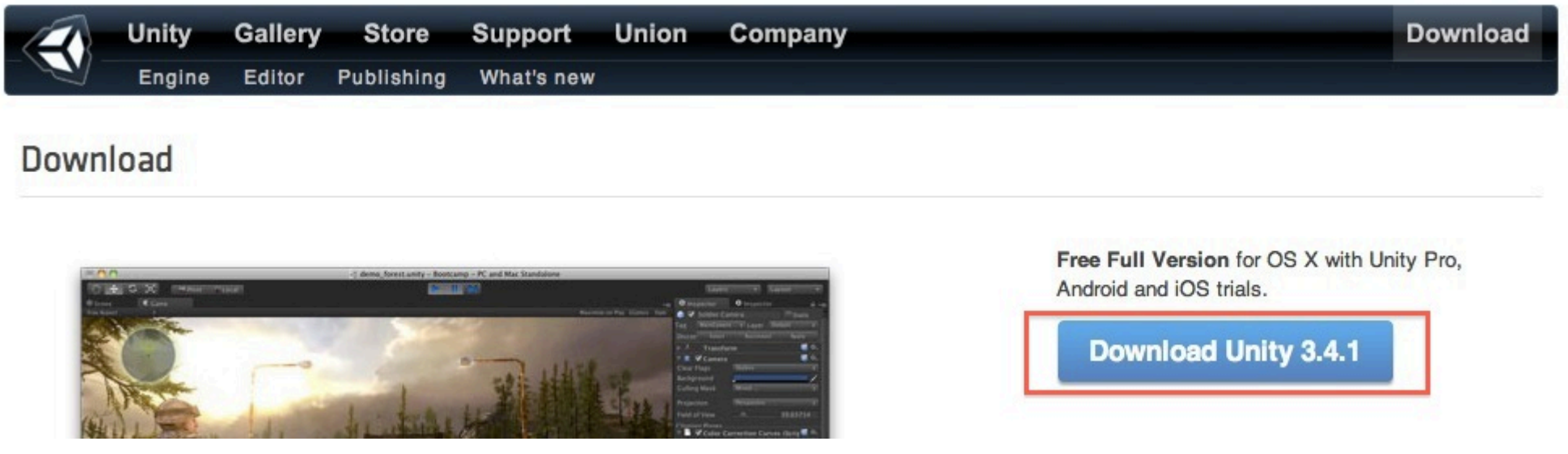
资料整理：六翼天使

## 目录



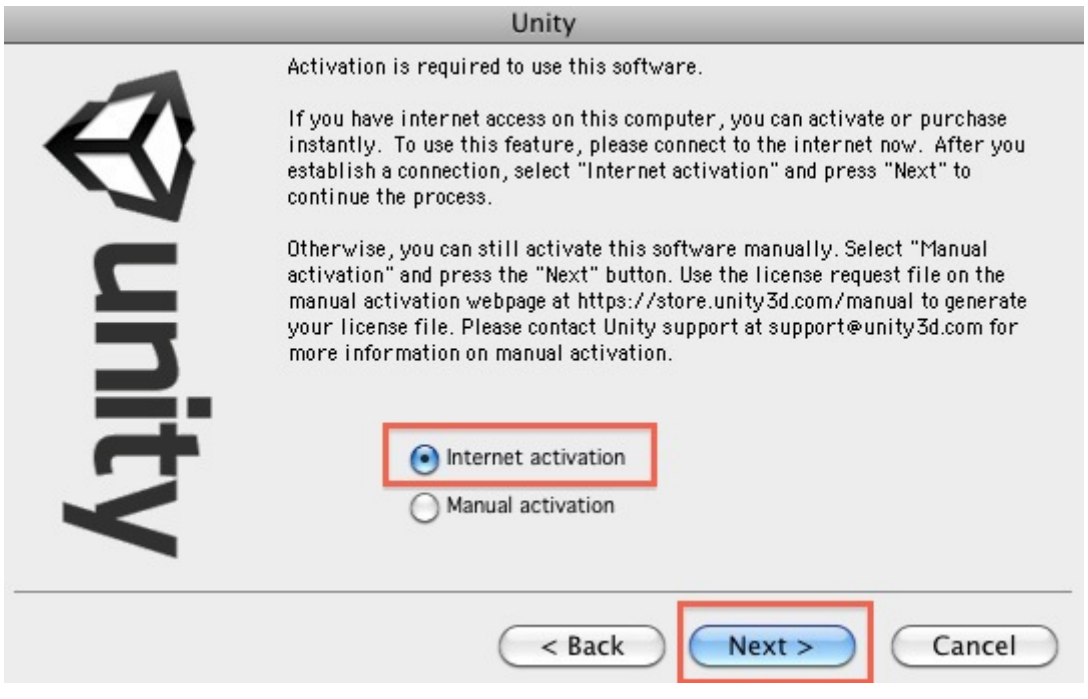
# 第一章 Unity3D 游戏引擎之构建游戏框架与导出 IOS 项目

首先先去 Unity3D 官网 http://unity3d.com/，我们可以看到很多关于 Unity3D 的消息。点击 DownLoad 开始下载 Unity，下载的时候建议不要使用 Safari 自带的下载工具，因为不支持断点续传，国外网站不稳定。建议使用迅雷去下载。目前最新的版本是 Unity 3.4.1 ，完美支持 Xcode 4 。

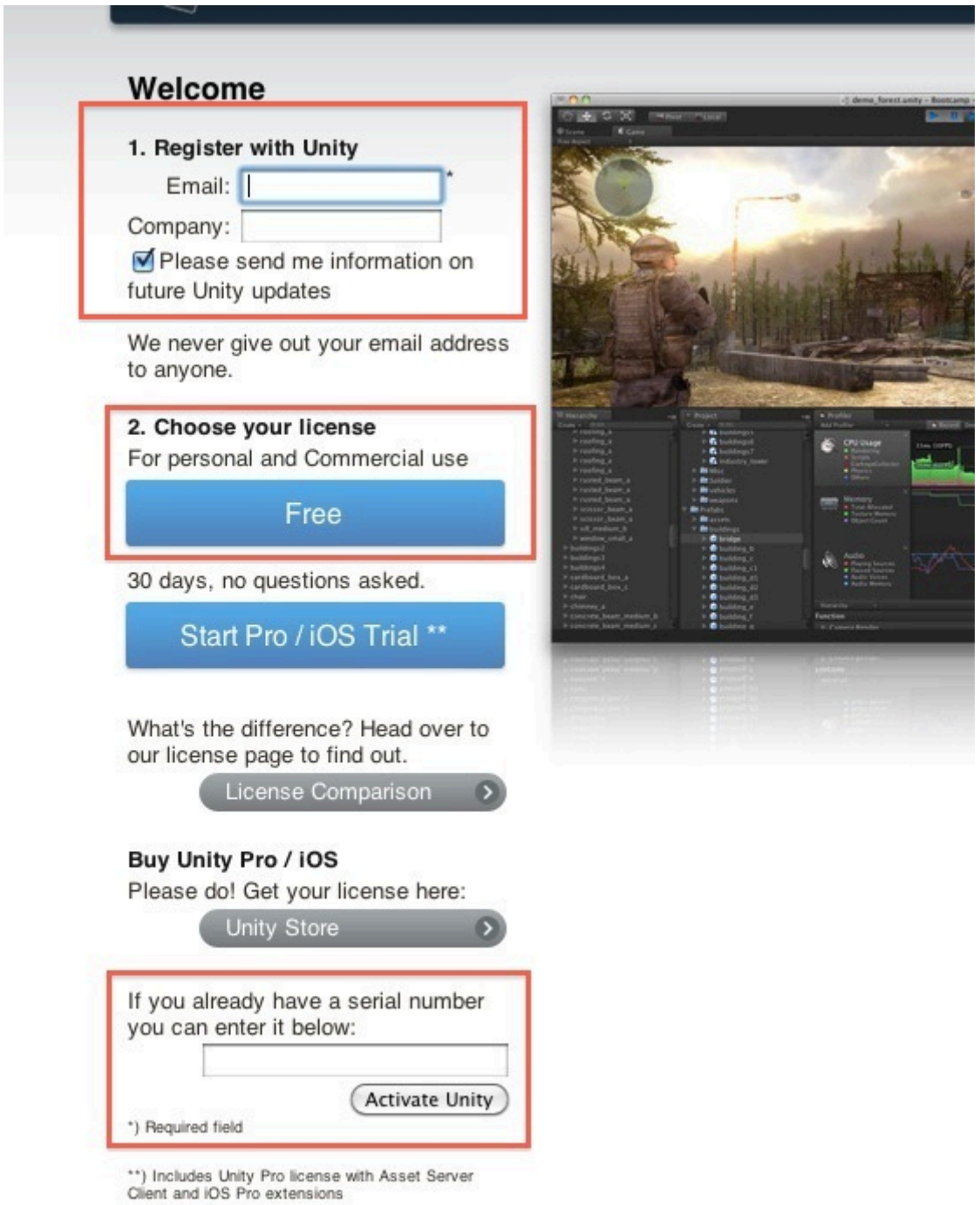



下载完成后，进入 Unity。首次须要走注册流程，这里说一下 Unity 3D 的购买许可为 1500 美刀，如果要部署在 IOS 设备上要在加 1500 美刀。确实有点贵，购买串号后装在我的 pro 上，我突然觉得我的电脑更加贵重了，哇咔咔～～当然便宜没好货，好货不便宜贵自然有贵的道理。废话不多说了我们继续～～ 如下图所示 点击 Register 开始注册。 网上有破解版本，也可以编译到机器上，大家可以去下载看看。但是破解版本不能上传 APP store，但是可以用来学习，哈哈～～

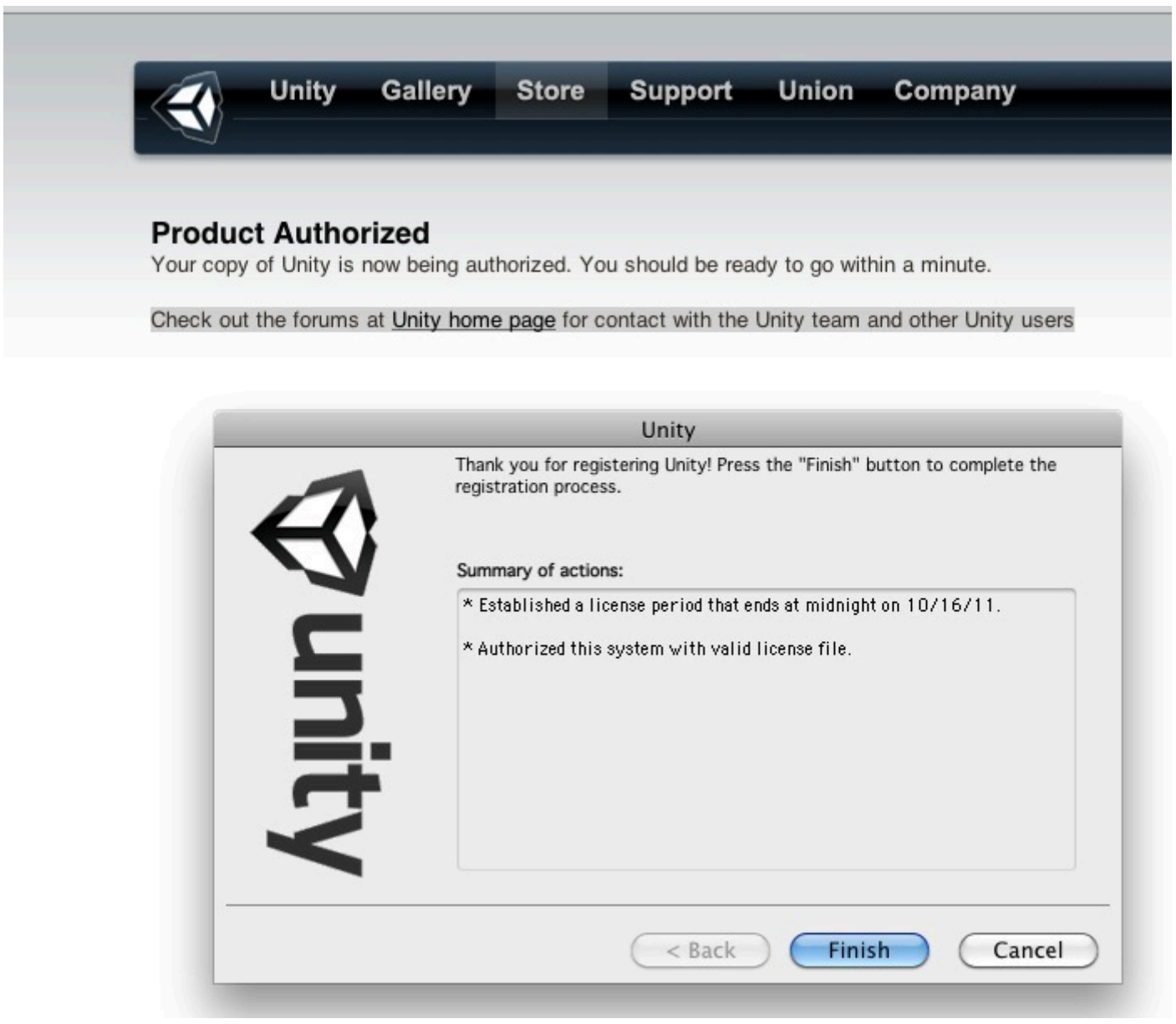

选中 Internet activation 点击 Next 继续。



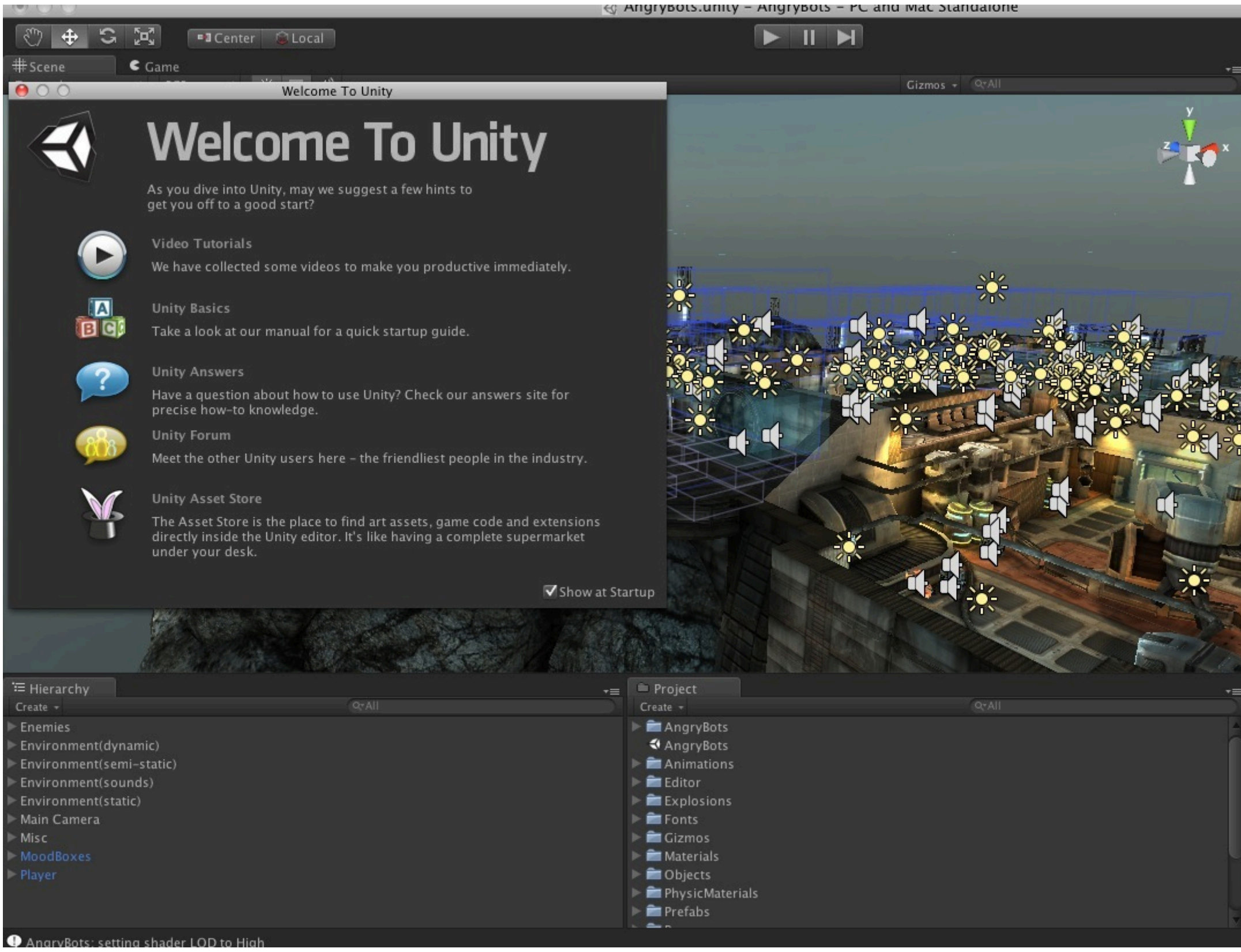

填写正确的 邮箱 与公司名称。点击 Free 的话可以试用 30 天，可以体验一下，如果已经购买过串号的话请在下方提示框中输入正确的串号，点击 Activate Unity 完成注册。


Welcome
1. Register with Unity
Email:
Company:
Please send me information on future Unity updates
We never give out your email address to anyone.
2. Choose your license
For personal and Commercial use
Free
30 days, no questions asked.
Start Pro / iOS Trial **
What's the difference? Head over to our license page to find out.
License Comparison
Buy Unity Pro / iOS
Please do! Get your license here:
Unity Store
If you already have a serial number you can enter it below:
Activate Unity
*) Required field
**) Includes Unity Pro license with Asset Server Client and iOS Pro extensions

这样子就注册就完成了



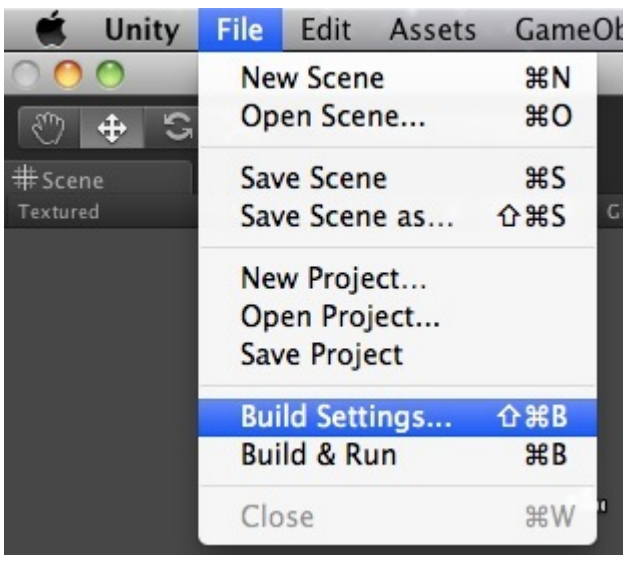

下面我们开始走进 Unity3D 的开发世界中 ,关闭欢迎界面 ,映入眼帘的是 Unity3D 自带的一个游戏 DEMO ,在 3D 的世界中存在着很多 3D 模型，以后我会慢慢带领各位盆友们学习 Untiy3D 引擎的 模型，贴图，动画，等等的使用.看起来这个非常有意思哦 嘻嘻~ ~



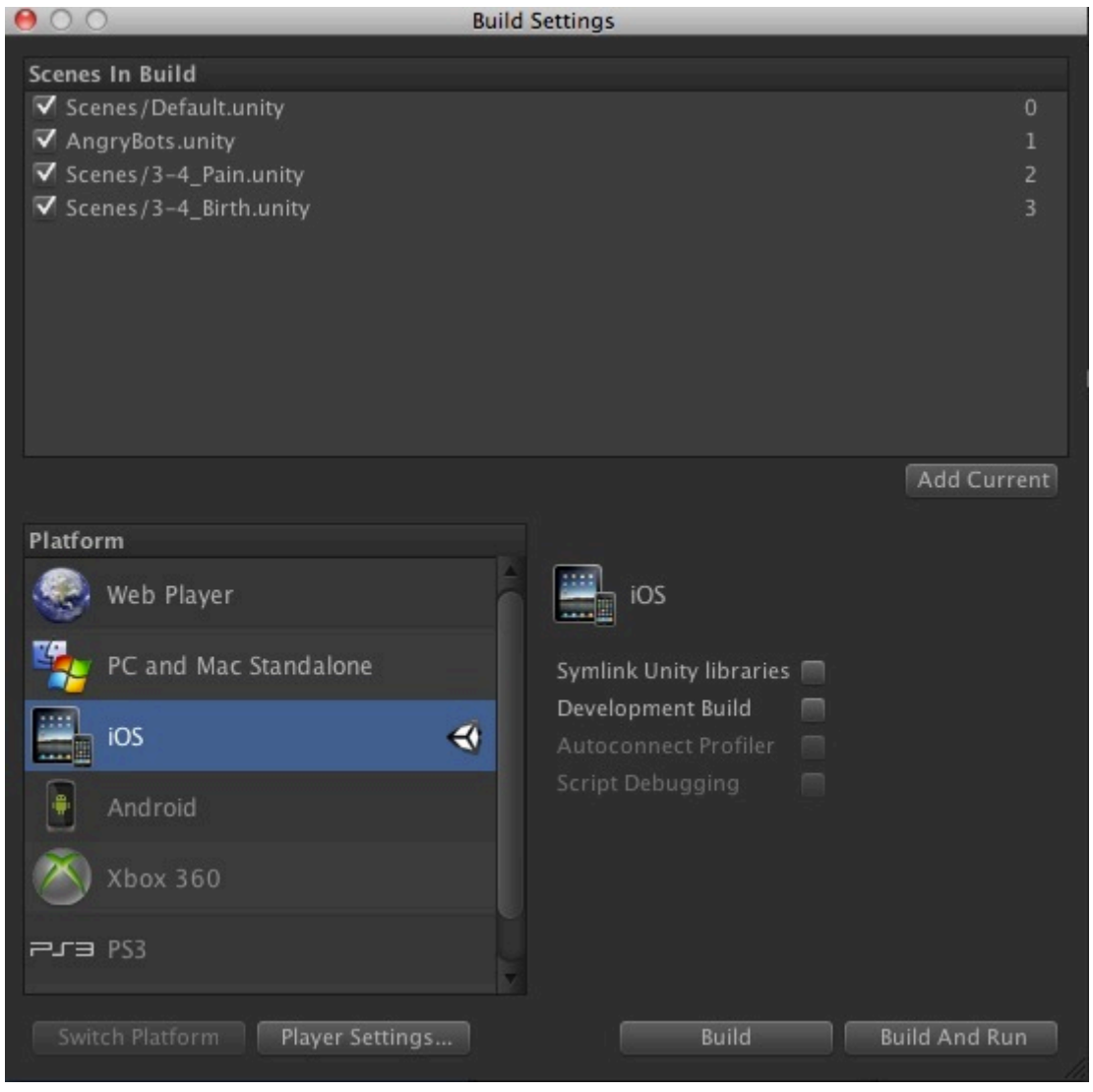

单机运行按钮，我们可以在 PC 上来玩这个游戏 DEMO，Untiy3D 真的是太强大了。本篇文章的目标是在 IOS 设备上部署 Unity 3D 引擎，那么下面我们将把这个游戏 DEMO 导入到 IOS 设备上，在 iPhone 上去玩这个游戏 DEMO。

下面将这个游戏 DEMO 导出为 IOS 程序。

点击 File->Build Settings

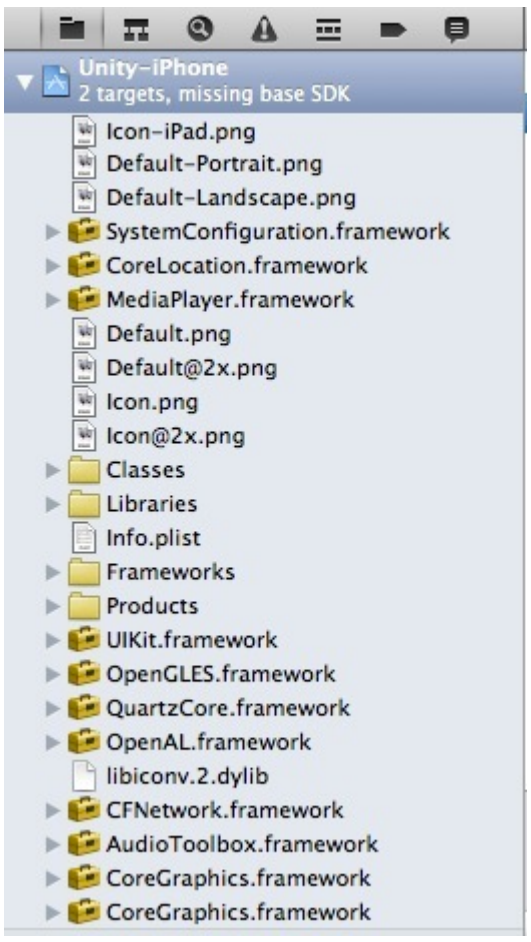



首先确保你的机器中装有 Xcode 4 ，IOS 设备 连接在电脑中，然后选中 IOS 设备，单机 Build and Run 这时候 Unity3D 就开始导出这个游戏项目了，请大家稍等片刻。导出成功后自动打开 Xcode 并且运行我们导出的这个项目。




因为模拟器是无法运行 Unity3D 导出的项目，所以设备一定要链接上 mac 否则无法运行。如果暂时没有设备可以在电脑上调试运行，等有设备的话将调试好的项目直接导入 IOS 设备方可，。

下图为 Unity3D 导出的项目结构，这时候看看我们连接在 mac 上的 IOS 设备。




哈哈，游戏已经顺利的安装成功，快快打开游戏，玩玩我们导出的游戏吧 嘿嘿～～

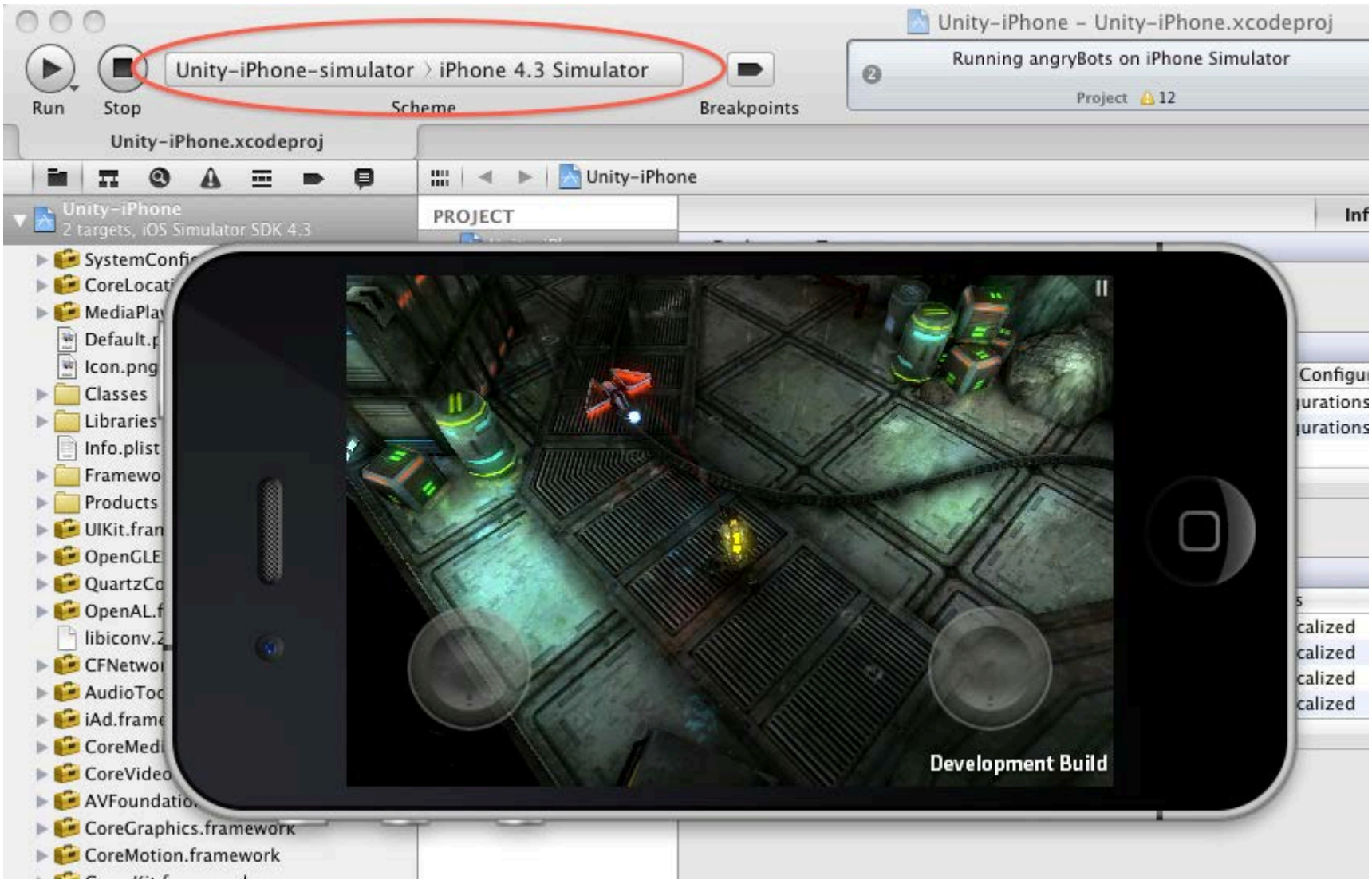
图片中左下面 和右下方分别有两个为 Unity3D 自动生成出来的按钮 一个是控制人物行走，一个是控制子弹发射方向。后面我会继续写一些这套游戏引擎方面的文章，哇咔咔～～ 欢迎广大盆友们可以和我一起交流 大家一起学习嘛～嘻嘻～～

## 补充：使用 Unity 游戏引擎在 IOS 模拟器中运行的方法：

之前 MOMO 一直有个误区，一直都是使用真机来调试程序，以为模拟器中不能运行 Unity 编译的 程序。但是不是的，模拟器同样可以运行 Unity 编译出来的 IOS 程序。

在 Unity 编译 IOS 程序时，在 Unity 导航栏菜单中选择 Edit->ProjectSettings ->Player（菜单项）选择 IOS 平台在下方 SDK Version 处选择运行设备为 IOS 模拟器。 选择完毕后 Build and Run 即可。

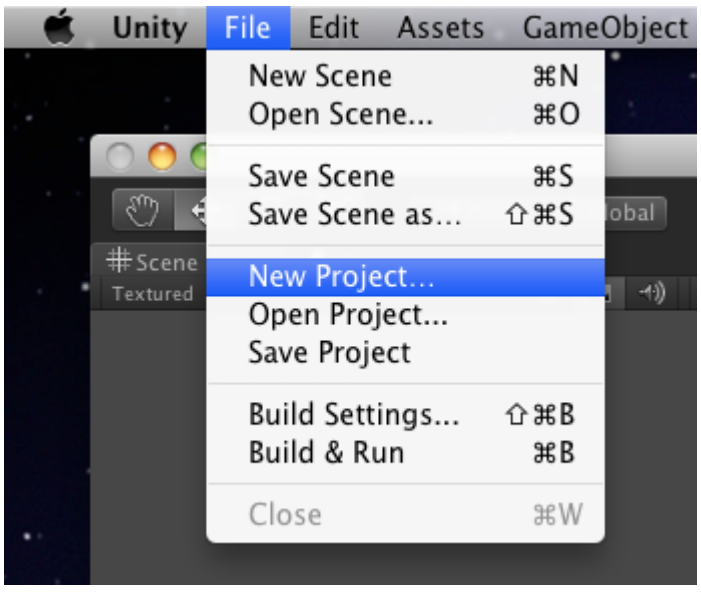

在 Xcode 中运行游戏即可将游戏模拟器打开。




# 第二章 Unity 游戏引擎之实现平面多点触摸

在上一章中已经介绍了 Unity for 3D 游戏引擎的构建，从本章以后我将带领大小盆友们一起更进一步的学习 Unity 游戏引擎。先从 Unity 平面开始，本章介绍 Unity 平面上的多点触摸。

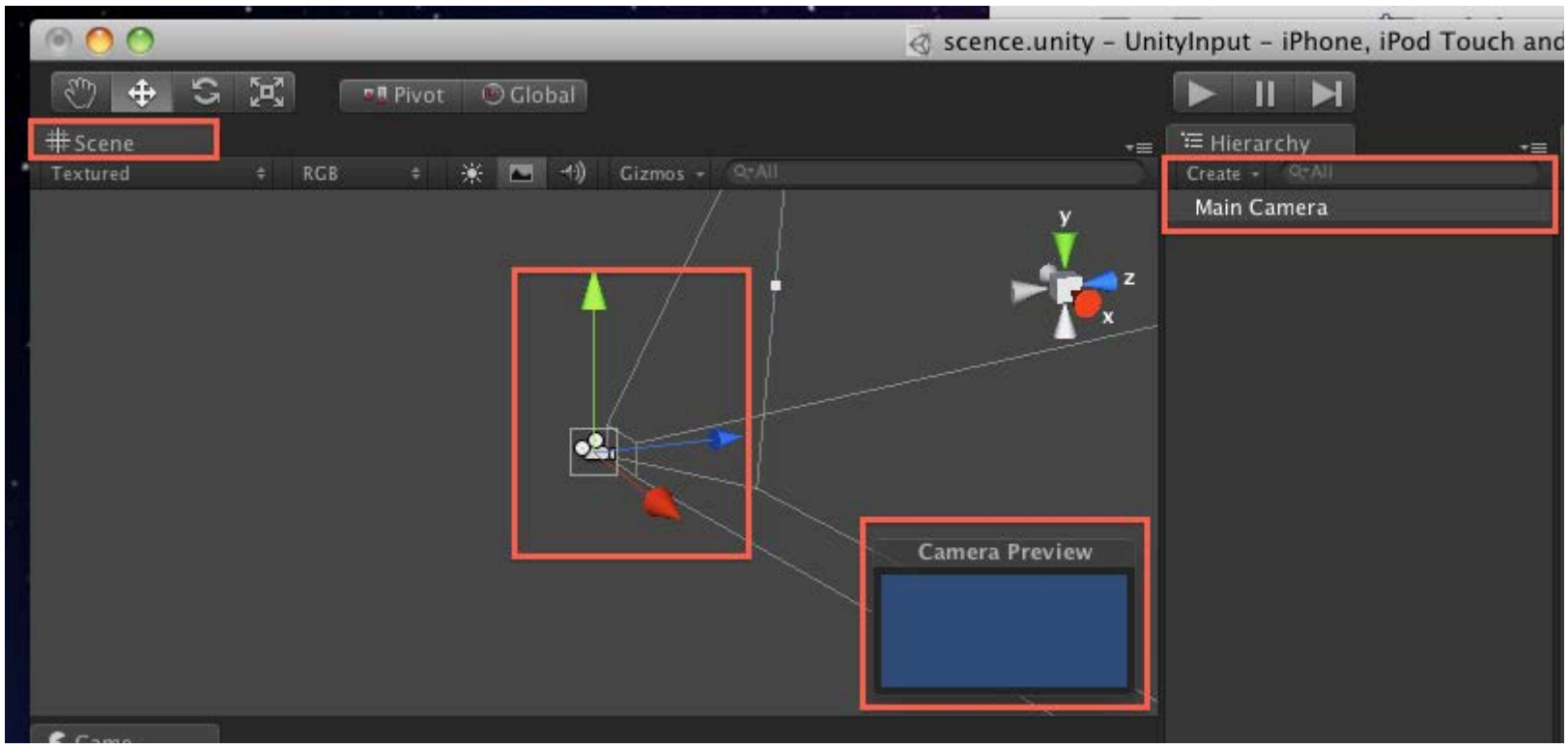
点击 File->New Project 创建一个工程，可以使用默认 或者 名称随便起一起即可。点击继续完成创建工程。



Scene:游戏场景视图，这里面可以摆放任意场景模型。

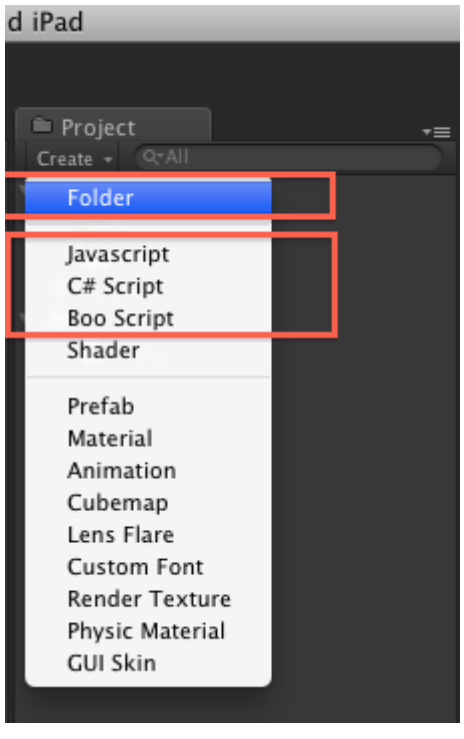
Camera Preview:摄像头正对方向屏幕显示内容

MainCamera:创建工程后默认会添加游戏的主摄像头，在场景视图中我们可以看见 MainCamera 在三维坐标系中的位置。




我们今天的目标实际上就是让摄像头正对一个平面，在这个平面中实现多点触摸。

下面介绍一下脚本的使用，为了让摄像头显示我们须要给主摄像头绑定脚本，如下图所示，点击红框内的 Create 出现下拉列表，先创建两个文件夹 ,Image 用来存放图片 ,Scripts 用来存放脚本。因为 iPhone 4 的分辨率是 960x640 所以找到一张 960x640 的图片做为屏幕背景图，然后在找一张小一点的图片做为触摸后在屏幕中显示的图片。




Unity 支持三种的脚本分别是 javascript,C# Script,Boo Script，官方推荐使用 javascript 来编写，所以我们也就用 javascript。

创建一个脚本名称为 menu.js，声明了两个变量 imageMenu 与 imageItem 来储存游戏背景显示与游戏触摸显示的纹理图片。

function OnGUI ()：这个方法用来通知屏幕绘制。

DrawTexture: 绘制纹理。

Label：绘制一个文本。

iPhoneInput.touchCount :得到多点触摸的数量。

iPhoneInput.GetTouch(i).position: 得到循环中每一个多点触摸的位置。

iPos.x： 触摸的 x 坐标

iPos.y： 触摸的 y 坐标（左上角为 00 点的 Y 坐标）

GUI.DrawTexture(Rect(x,960 - y ,120,120),imageItem);

960 - y： 因为取得的 y 坐标是左上角 00 点的坐标，而 Unity 绘制是以左下角为 00 点的坐标，不处理直接用 y 的话坐标就是一个反的，所以这里用 iPhone 4 的 高度 960 减去 当前触摸的 Y 就算出触摸正确的显示坐标。

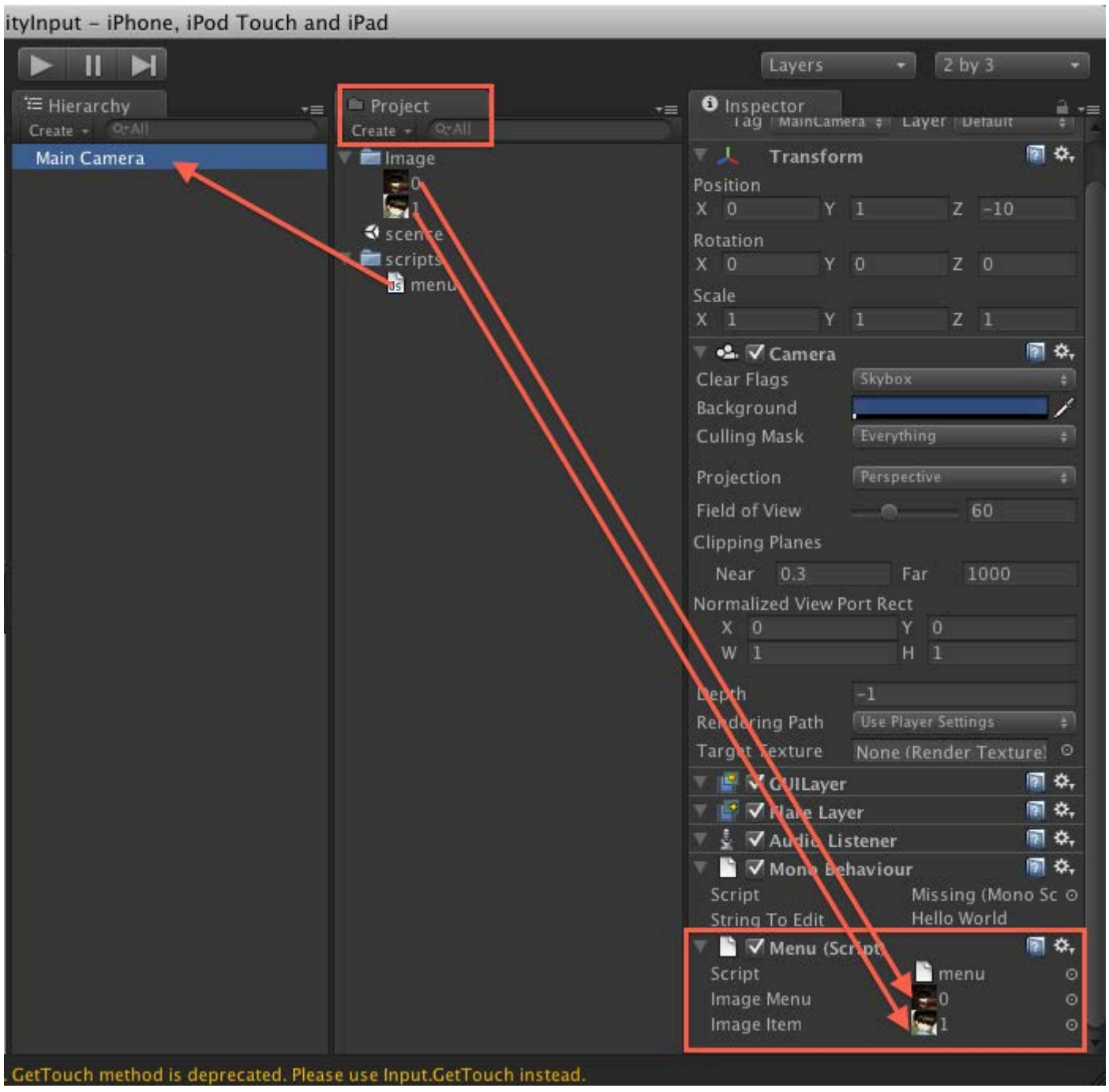
[javascript] view plaincopyprint?

```
1.  var imageMenu :Texture2D;
2.
3.  var imageItem :Texture2D;
4.
5.  function OnGUI () {
6.
7.      GUI.DrawTexture(Rect(0,0,640,960),imageMenu);
8.
9.       var touchCount = iPhoneInput.touchCount;
10.      for(var i = 0; i < touchCount; i++)
11.      {
12.         var iPos = iPhoneInput.GetTouch(i).position;
13.         var x = iPos.x;
14.         var y = iPos.y;
15.
16.         GUI.DrawTexture(Rect(x,960 - y ,120,120),imageItem);
17.
18.         GUI.Label(Rect(x, 960 - y,120,120),"Touch position is  " + iPos);
19.      }
20.
21.
22. }
```

```
var imageMenu :Texture2D;

var imageItem :Texture2D;

function OnGUI () {
```

脚本已经添加完毕，接下来是绑定变量。

先将 menu.js 拖拽到 Maincamera 中，可以看到右下角红框中出现两个没有赋值的变量 ImageMenu 与 ImageItem ，因为上面脚本中声明了这两个变量，这里就会出现。在这里须要对这两个变量赋值。然后拖拽图片放入为其赋值。




OK 接下来就是 build and run ，具体方法见上一章，这里就不罗嗦了。 我们看看运行在 iPhone 上的真机效果图。

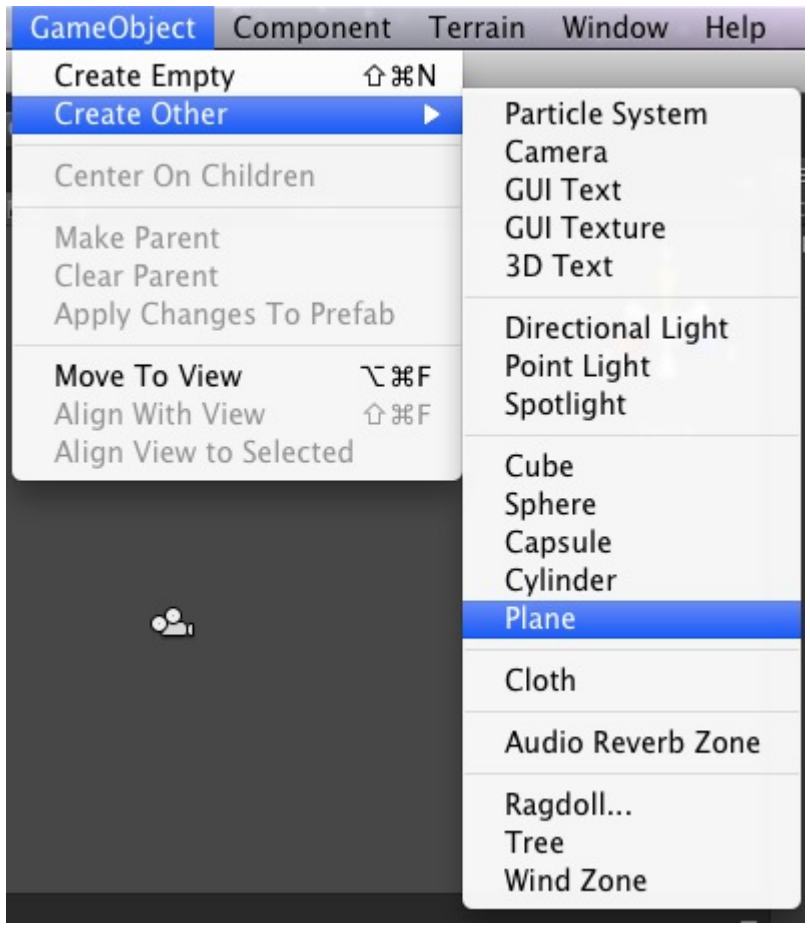
因为 Unity3D for iPhone 只支持 5 点触摸，所以我将 5 根手指头放入 iPhone 4 中 图中清楚的记录我的 5 根手指头所在屏幕中的位置，以及正确的贴上了须要显示的图片。


哇咔咔～～同样也希望各位技术达人可以和 MOMO 一起进行交流～～ 一起进步喔～～

# 第三章 Unity3D 游戏引擎之构建简单的游戏世界

## 创建游戏地面

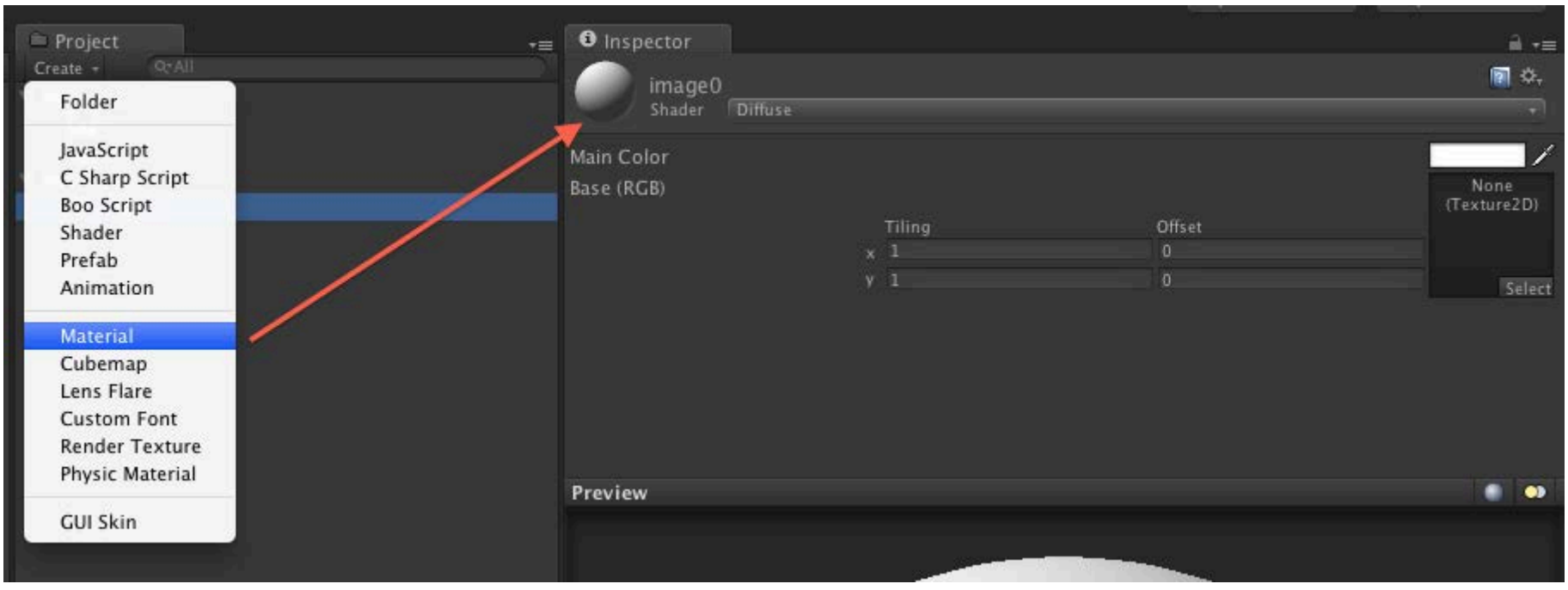
创建一个新的游戏工程名称命名为 FirstGame，场景视图中出现了默认的主摄像头，那么开始添加一个游戏地面，如下图所示添加一个游戏地面，其实 Unity3D 有一个地形的概念，Terrain 可以创建一个游戏地形，以后在介绍这方面的知识。今天我们主要的目的是构建简单的游戏世界。




## 创建纹理贴图

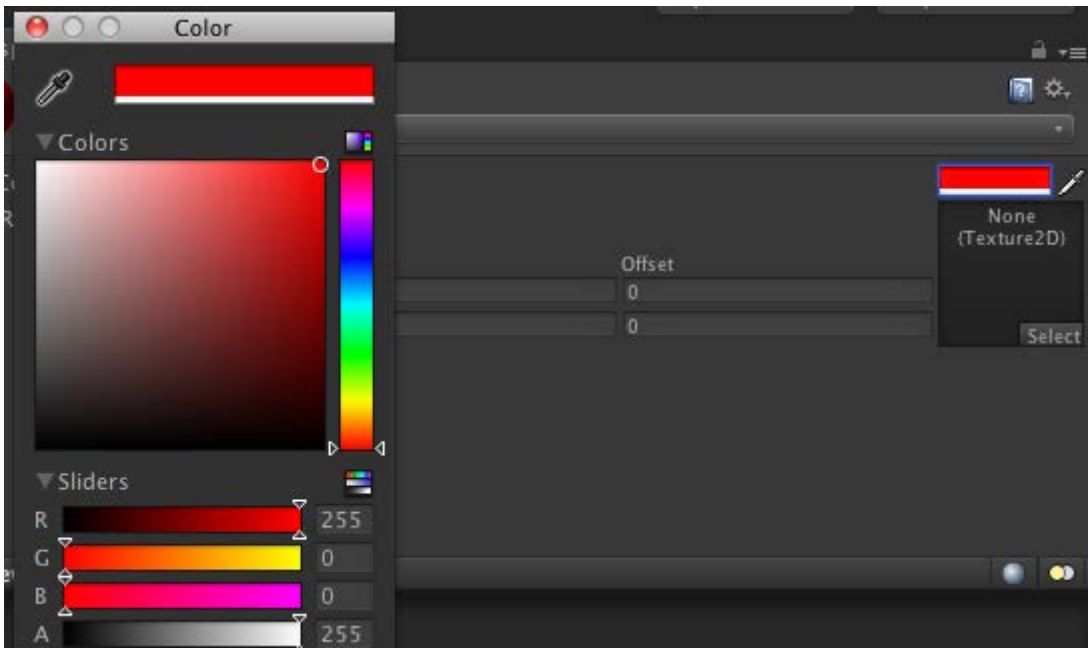
将须要显示的图片托放入工程中，拖放方法不知道的盆友请看上一章。

点击 Material 创建一个纹理贴图，命名为 image0 -> 出现箭头右侧 信息框，渲染模式与位置偏移量这里都使用默认，点击右侧颜色框与 Texture2D 纹理选择框 编辑这个纹理的显示颜色与贴图内容。




这里出现之前拖入工程的图片，选择一张图片做 image0 的贴图，默认有一张渐变的 Default-Parti 可以进行选择。


点击颜色编辑框，不仅可以添加图片，也可以修改颜色，如下图所示。




点击 GameObject - > CreateOther 创建简单的游戏世界 3D 系统自带模型。

Capsule :胶囊体

Cube:正立方体

Cyinder:圆柱体

Main Camera:主摄像头

Plane:一个平面

Point Light:点光源

Sphere:圆形

Spotlight:摄像光源

如下图所示：鼠标连线那样将贴图拖放至需要显示的物体上，左上角四个按钮从左到右的功能分别是

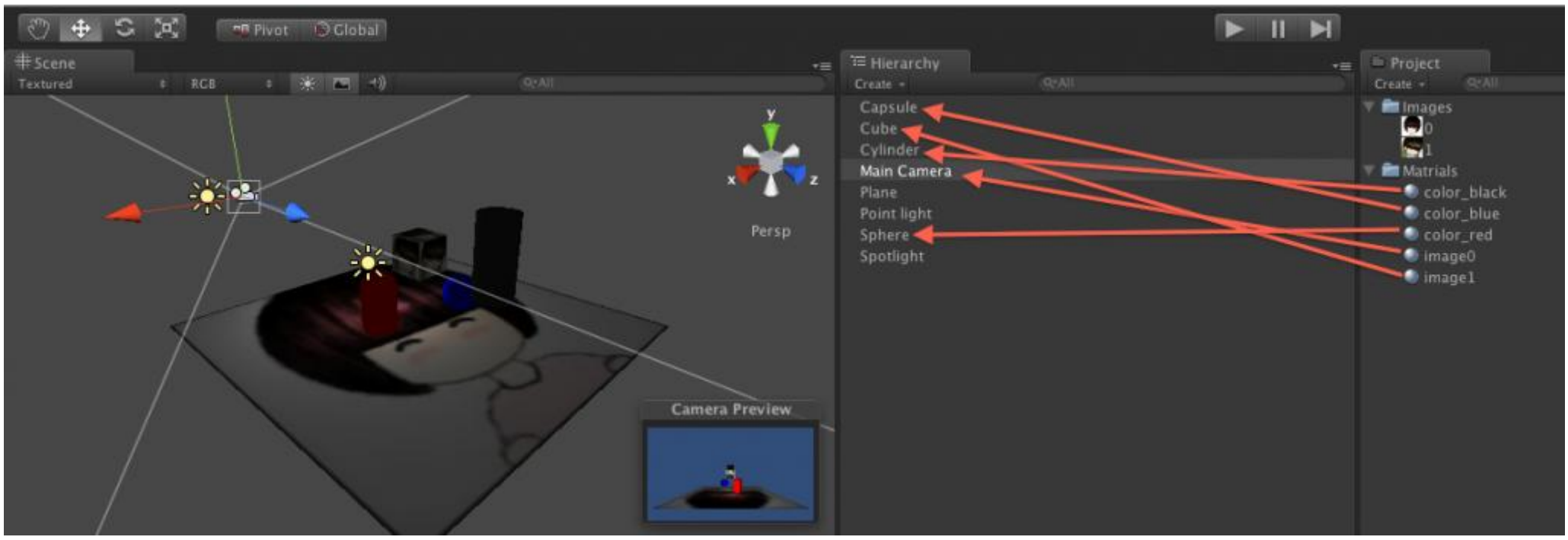
1 点住鼠标移动整体视图

2 移动某个物体在三维坐标系的坐标

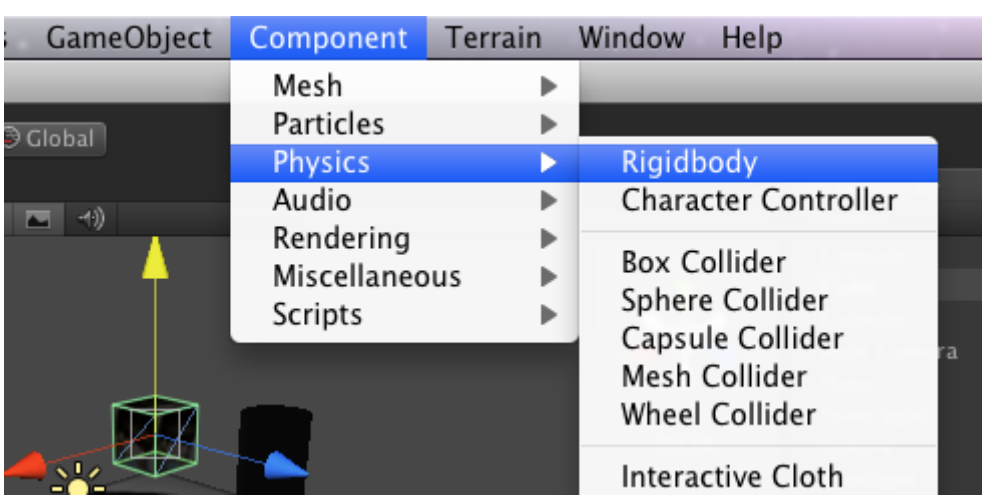
3 物体的旋转

4 物体的放大

三维坐标系默认是 45 度角在平面中拖动实在是没有感觉也没有概念，总不能及时的确定拖放的位置，这里值得一提的是 Scene 视图中右上角的 Persp ，可以看到 X Y Z 方向都有 3 个锥形，点击后视图只显示选择的那个方向的平面视图，可以方面我们拖动物体，点击中间的小方块后又重新回到默认的 45 度角。

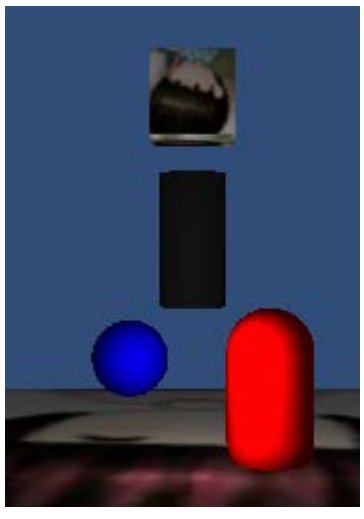



点击 Component -> Physics - > Rigidbody 可以给多个物体绑定一个重力感应碰撞，我将这个箱子的 Y 坐标抬高到地面 看看这个箱子是怎么自由落体的。

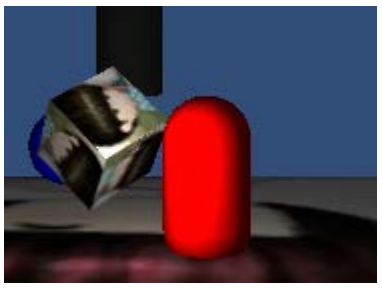



说了着么多了我们快快点击运行按钮快快看看游戏运行结果。具体往 IOS 设备上编译的方法 ,第一章有详细介绍 ,不懂的盆友请阅读 Unity 游戏引擎第一章 ,哇咔咔～～～

箱子开始下落



箱子下落中



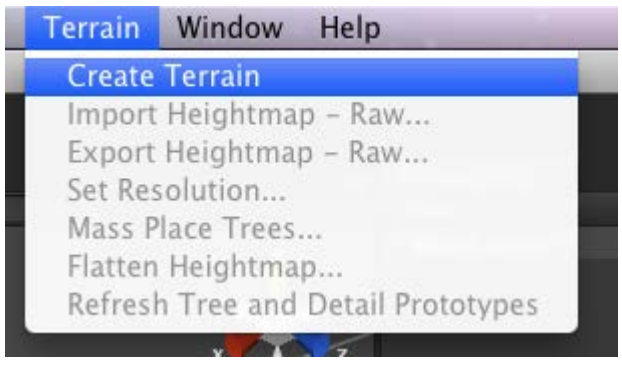
怎么样？不错吧，物理引擎这一块也非常棒把怪不得仙剑 Online 也选择用 Unity3D 这套游戏引擎。还是那句老话 欢迎各位大小盆友和 MOMO 一起交流游戏开发 哇咔咔～～～

# 第四章 Unity3D 游戏引擎之构建 3D 游戏的基本地形

## 创建一个 3D 地形

在上一章中介绍了简单的游戏平面，当然 Unity3D 中提供了非常强大的地形编辑器，凹凸，贴图，碰撞，你能想到的功能它都可以做，给力吧。快快构建我们的 3D 游戏地形～哇咔咔～～～

点击 Terrain - > Create Terrain 创建一个游戏地形，我们命名为 MyTerrain。



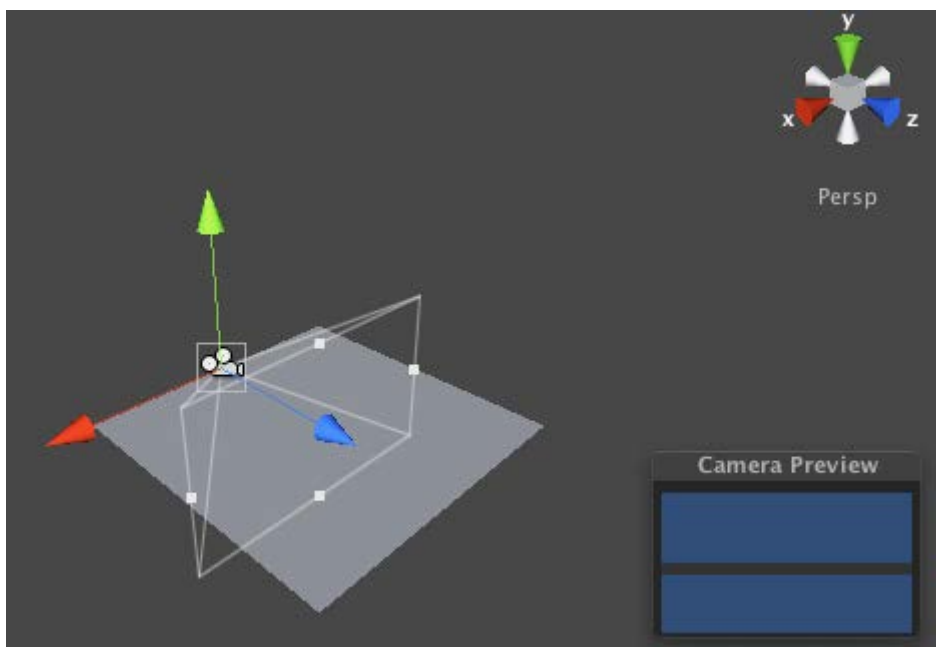

调整一下摄像头的角度，正对着我们创建的游戏地形，补充一下 Unity3D 场景编辑器的移动的控制。

模型的旋转： Option + 鼠标左建

模型的平移： Option + 鼠标中键

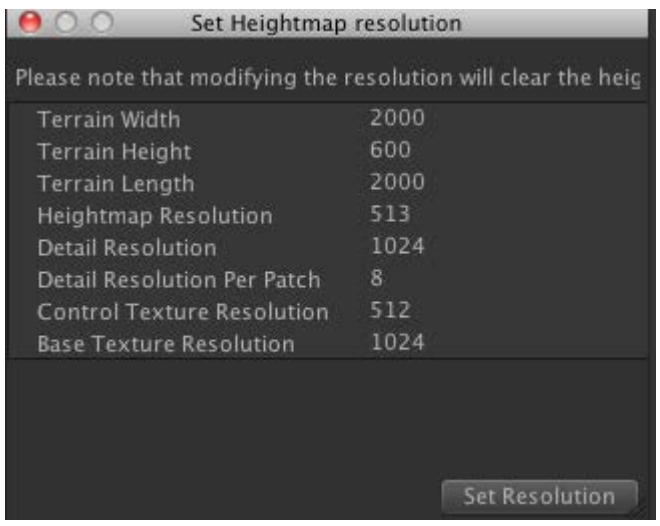
模型的缩放： Option + 鼠标右键

在下图中可以清晰的看见创建的游戏地形，与摄像头映射的方向与显示的内容。




地形一旦创建完毕后，Unity3D 会默认地形的大小，宽度，厚度，图像分辨率，纹理分辨率，等等，这些数值是可以任意修改的。

点击 Terrain - > set Resolution 打开设置地形参数菜单，如下图所示。



如上图所示从上到下分别代表的含义是

Terrain Width:  地形的宽度

Terrain Height:  地形的高度

Terrain Length:地形的长度

HeightMap Resolution:地形高度图的分辨率.

Detail Resolution:细节分辨率，控制草和细节网格地图的分辨率。数值越高标示效果越好，想对也越消耗机器性能，可能会卡。根据情况适当的调节。

Control Texture Resolution:控制不同纹理的分辨率。

Base Texture Resolution:控制相对纹理分辨率，这里指一定范围内的。

设置完毕点击 set Resolution 按钮，有兴趣的朋友可以动态的修改一下参数，看看你的地形发生了什么样的改变？

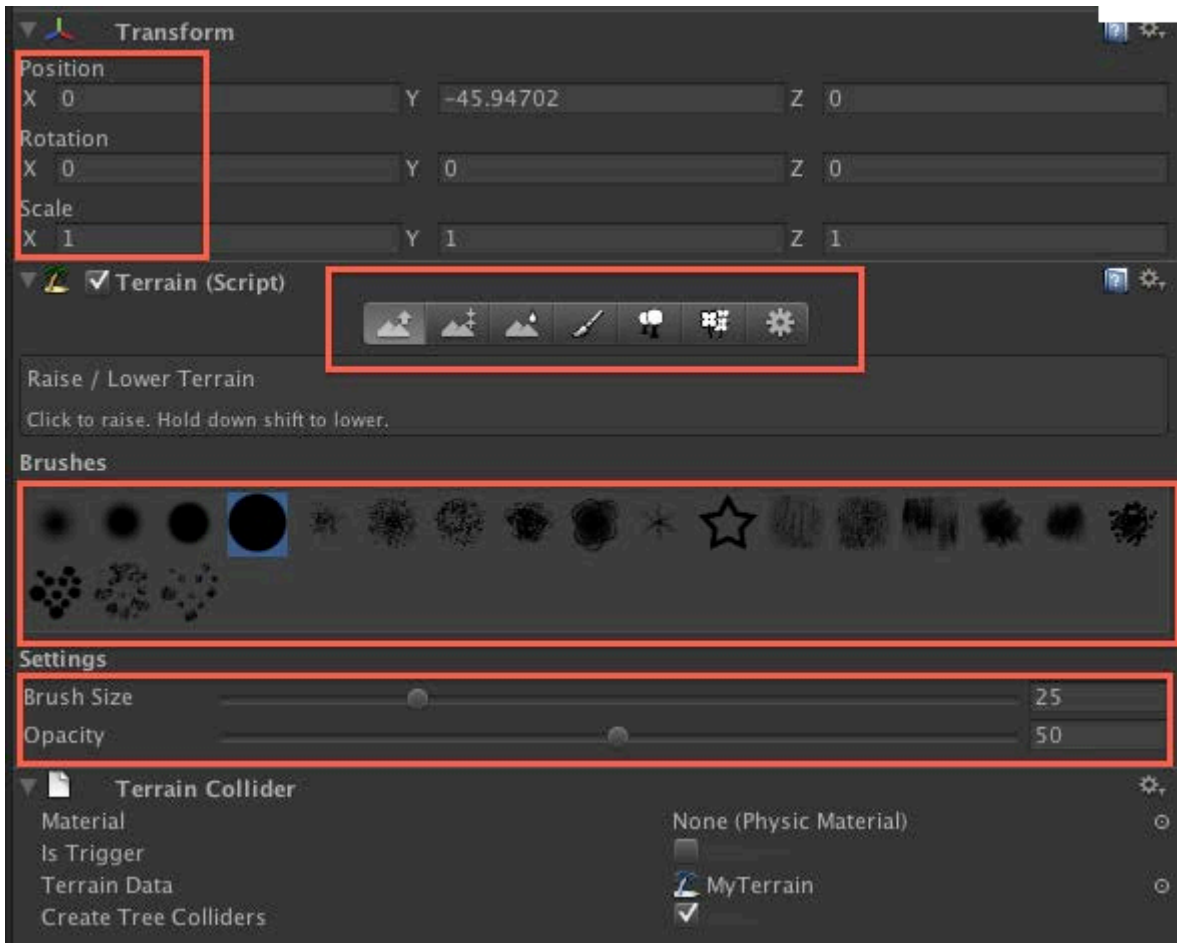
这样子我们的地形表面就创建完毕，下面给地形上添加一些五彩缤纷的元素吧。。。

五彩缤纷的元素

Hierarchy 中点击 Terrain，Unity 编辑器右侧栏中出现地形编辑窗口。下图中红框内是比较重要的一些信息，从上倒下分别为。

1 .Position Rotation Scale 地形的位置旋转 缩放，任何模型物体点击后右侧都会出现这个窗口，标志着当前模型在 3D 空间中的位置，修改参数模型也会跟着在三维空间中改变。

2.这里包含 7 个按钮，从左到右依次为编辑高度、编辑特定高度、设置平滑、纹理贴图、画树模型、画草模型、其他设置.善用着七个工具可以编辑一个好看实用的地形，稍后我会介绍这七个按钮的具体操作流程。




## 编辑高度

默认光标是指在编辑高度这个按钮上。

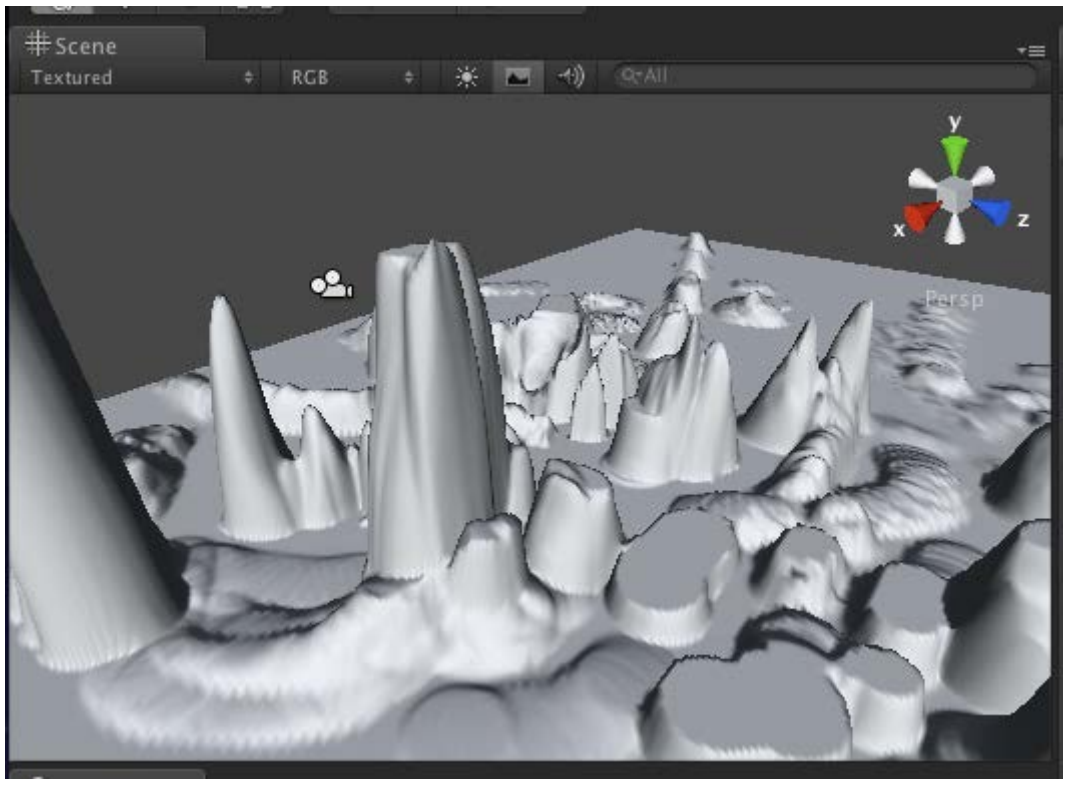
Brushes：地形绘制画笔，这里有很多种画笔的图案可供我们选择。

Brush Size :画笔宽度取值范围

Opacity :画笔高度取值范围

各位盆友们具体设置一下拖动鼠标点地形中点击一下就可以充分的感受这些参数的意义，鼠标左右移动是画笔绘制地形宽度，上下移动则是绘制地形高度。按住 Shift 键拖动鼠标是凹陷地形。

如下图所示简单的地形就映入我们的眼帘，看起来有点粗糙，别担心好戏在后面。




## 编辑特定高度

和编辑高度页面中的信息差不多多了一个 Height 用来设置最大的高度，编辑高度中 Opacity 是最大高度，但是在这里 Height 才是最大高度但是 Opacity 必需有数值，举个例子 Opacity 设置为 100　Height 设置 80 最高的高度就是 80　.Opacity 设置为 20　Height 设置 100 最高的高度还是 20.

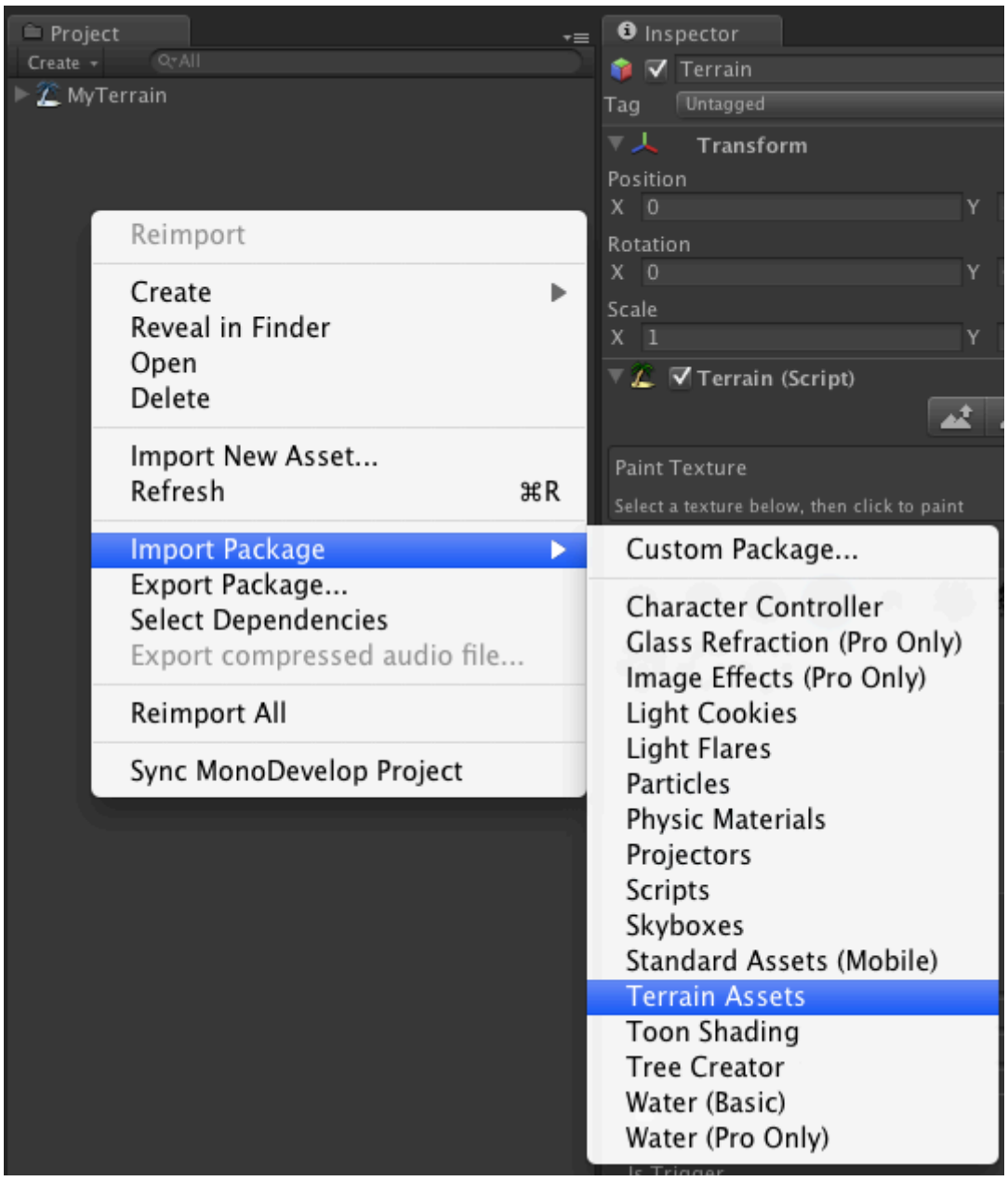
## 设置平滑

用上面的方法创建的地形，感觉边角有点粗糙，在这里可以设置边角平滑过渡。

## 纹理贴图

给地形添加图片，制作好看的游戏地形，资源方面我们可以导入系统标准的资源库,里面有很多好看的地形资源，当然也可以自己添加喜欢的图片做地形资源。

Project 标签中 ，右键 - > Import Package - > Terrain Assets。我们可以看见里面有很多资源包，暂时我们先导入地形的资源，之后再去考虑其他资源包。



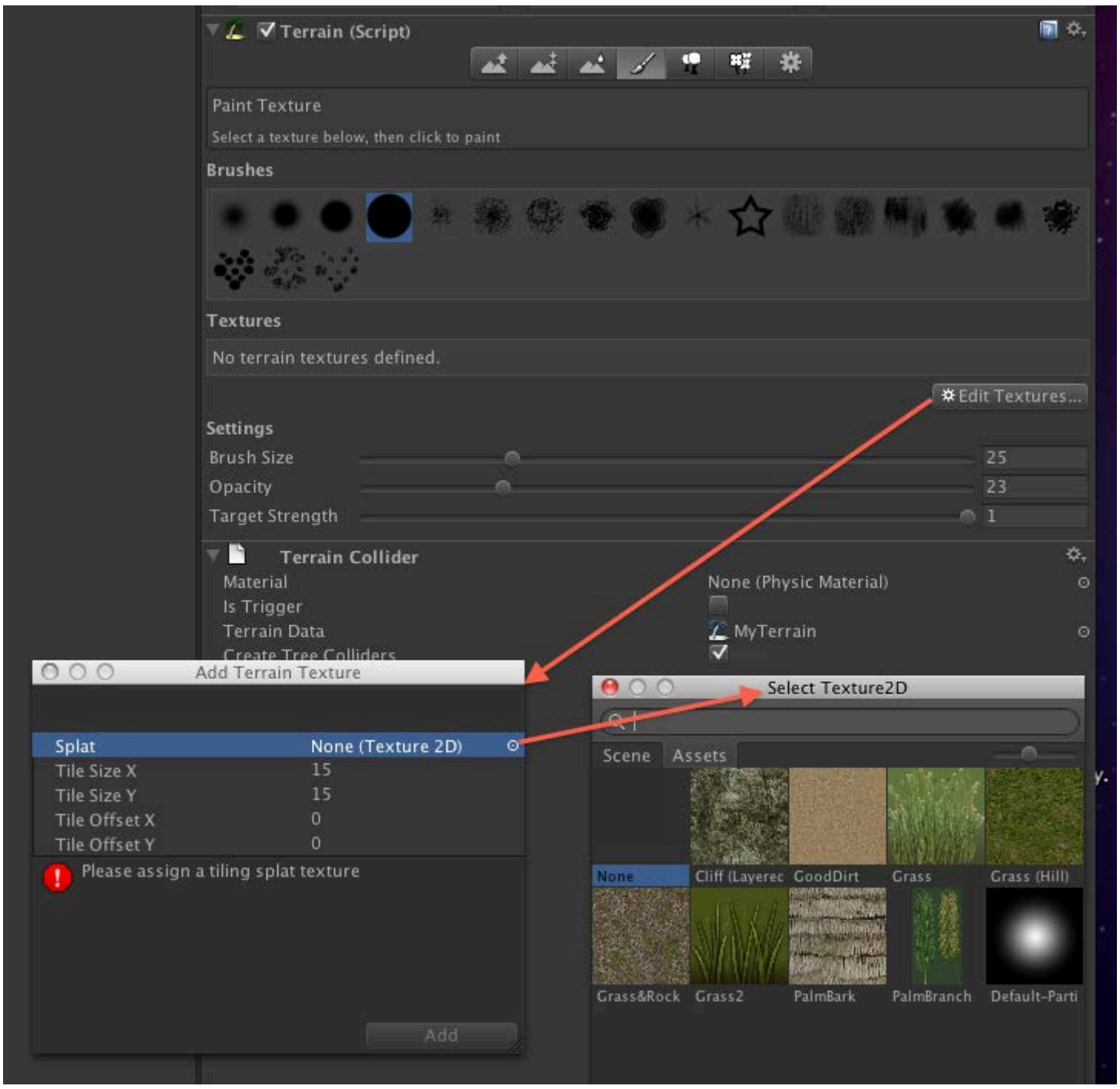

点击 Edit Textures　出现一个下拉列表

add Textures 添加一个贴图

Edit Textures　编辑贴图

Remove Textures 删除贴图

一个场景可以添加多个贴图，比如山丘用绿色的，平原用黄色的等等。




## Add Terrain Texture (添加贴图)

splat :选择一个贴图，上面导入了系统自带的地形资源，在这里随便添加两个贴图用于区分资源。也可以添加自己喜欢的图片，拖放在工程中就可以在这里看到图片资源。

Tile size x:  设置贴图 X 轴宽度

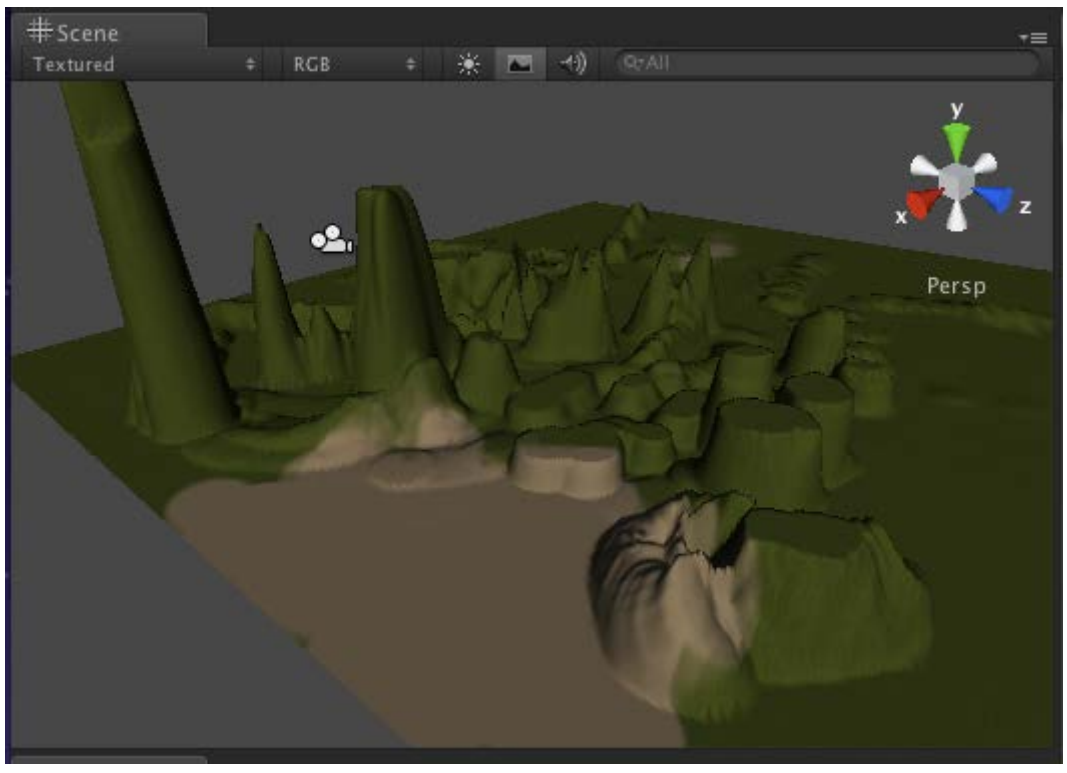
Tile size x:设置贴图 Y 轴宽度

Tile offset x:设置贴图 X 轴偏移量

Tile offset y:设置贴图 Y 轴偏移量

有兴趣的盆友修改一下参数方可看到效果，这里暂时使用默认数值。。

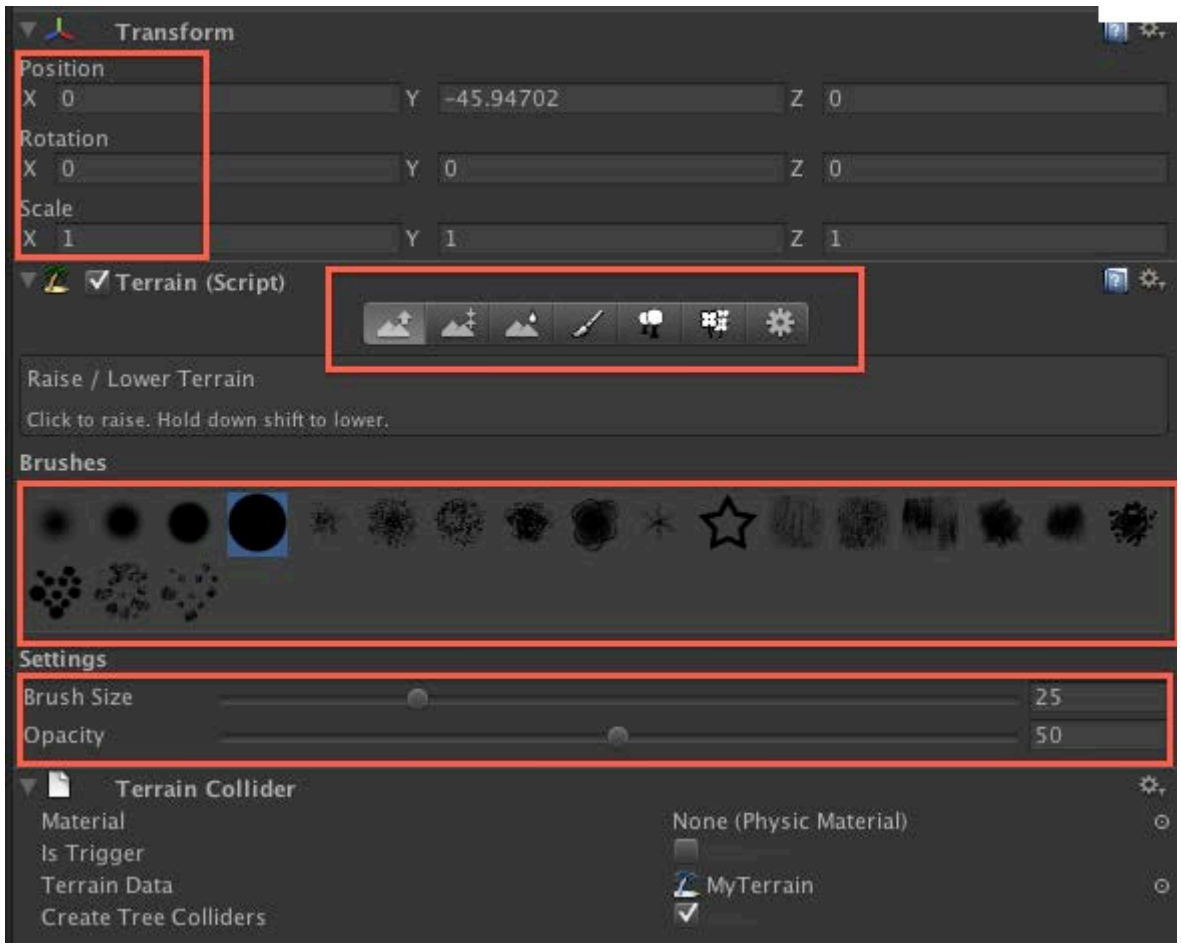
如下图所示，这里我添加了两种颜色的贴图，用于区分山丘和平地。




本章就先到这里，有关地图的特效画树模型、画草模型、其他设置这三组模型的建立，可以丰富我们的游戏场景，我将在下一章中重点介绍，届时欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，哇咔咔～～～

# 第五章 Unity3D 游戏引擎之构建 3D 游戏的基本元素

我们继续创建一个完美的 3D 游戏地形，将树木，花草，加入我们的游戏世界中，丰富 3D 世界的游戏元素。如下图所示 MOMO 继续向大家讲解 画树模型、画草模型、其他设置 ，这三个标签栏的意义。




## 画树模型

Hierarchy 标签栏中，点击 Create - > Tree 可以创建一个树的模型，设置自己的树木模型，这里我们先导入系统自带的树木模型，以后在讨论自定义模型的制作。

因为新建的工程中是没有树木和草地的贴图元素，可以在 Unity3D 的标准资源库中导入，导入的方法和上一章介绍的一样。打开 Unity3D 在 Project 标签栏中 鼠标右键 Import package - > Tree Creator 将标准树木资源模型导入工程。

点击 Edit Textures 出现一个下拉列表

add Tree 添加一个树的模型

Edit Tree 编辑树的模型

Remove Tree 删除树的模型

## 红框内是设置树木的一些关键重要参数

Brush Size： 画笔绘制一次树木添加数量，数值越大越多，越小则越少，取值范围 0 到 100。

Tree Density:树之间的百分比，在一片树中间在放入量一片树就得修改这个数值了，取值范围 0 到 100。

Color Variation：树之间颜色差的范围，取值范围 0 到 1。

TreeHeight：树的高度，它是与场景模型有一定比例，越大树越高，取值范围 0 到 200。

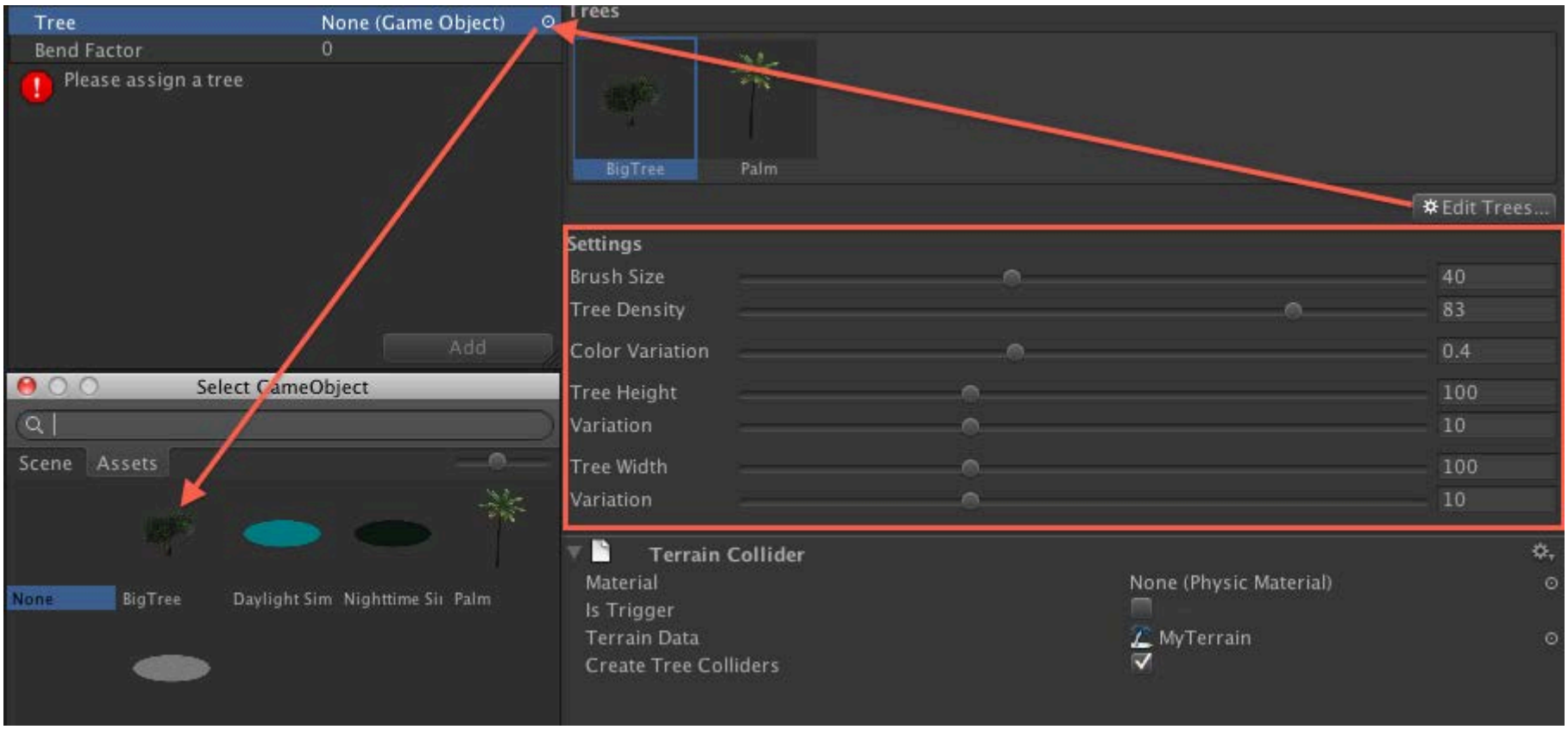
Variation：树与树之间的高度比例，取值范围 0 到 30。

Tree Width：树的宽度，它是与场景模型有一定比例，越大树越宽，取值范围 0 到 200。

Variation： 树与树之间的宽度比例，取值范围 0 到 30。

有了这些参数，我们就可以创建一些更贴切的 3D 树的模型在场景中啦。

Tree None(Game Object) 添加树木的模型.




设置树木的模型完毕后，用鼠标在场景中点击添加树木把， 按住 Shift 点击鼠标可以消除之前场景中已有的树木。


画草模型：

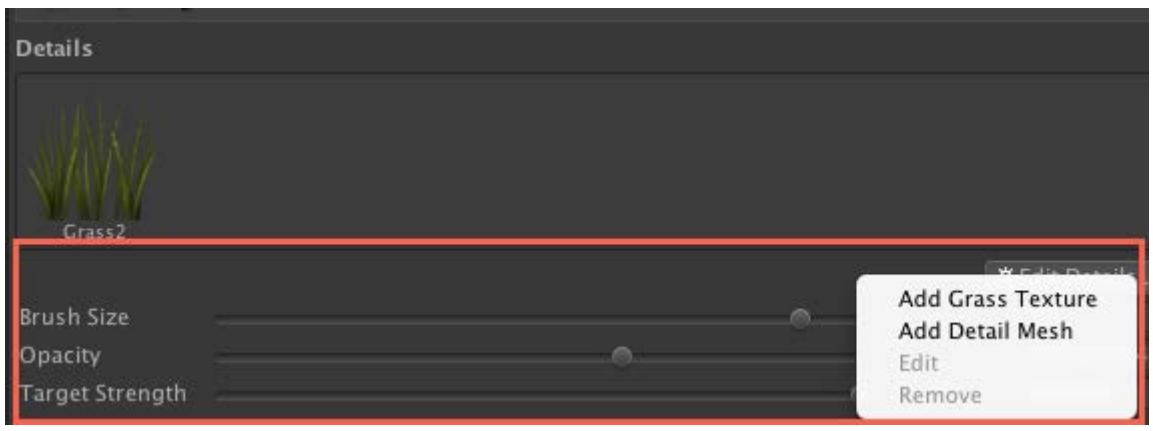
起始它不仅能草，还能画一些自定义模型,可以处理一些零碎的小东西丰富游戏场景，如下图所示，Add Grass Texture 为添加一个草的纹理图，Add Detail Mesh 添加一个自定义的模型。

Brush Size: 和上面一样，标志绘制的面积。

Opacity: 绘制的高度

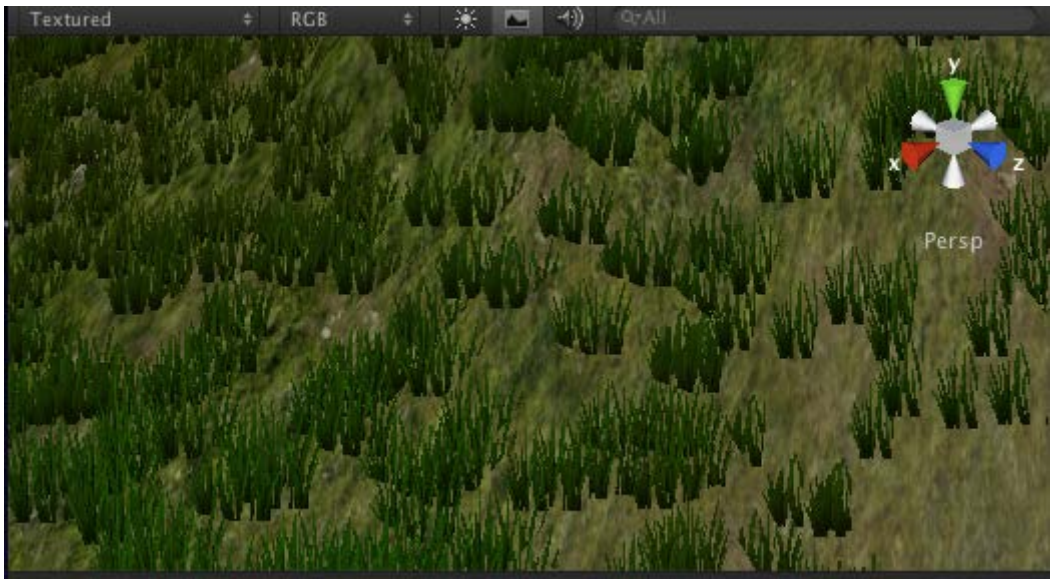
Target Strength:绘制的密度。

具体绘制的方法与添加模型的方法和画树类似，这里就不贴图概述了，盆友们打开编辑器试一试就 OK 啦。。




看一看草绘制出来的效果

具体编辑 ，移动草，拖动，以及模型的方法和上面类似，快快构建场景中的小玩意吧。嘎嘎～～




## 其他设置

主要设置一些 3D 游戏地形的一些参数

一下面列出一些主要参数的介绍，盆友们可以自己拖动鼠标修改一下其中的具体数值就可以在游戏视图中清晰的看到效果。

Pixel Error：控制地形密度容差，数值越大越圆滑，越小地形角度越明显。

Base Map Dist：控制地形贴图的距离.

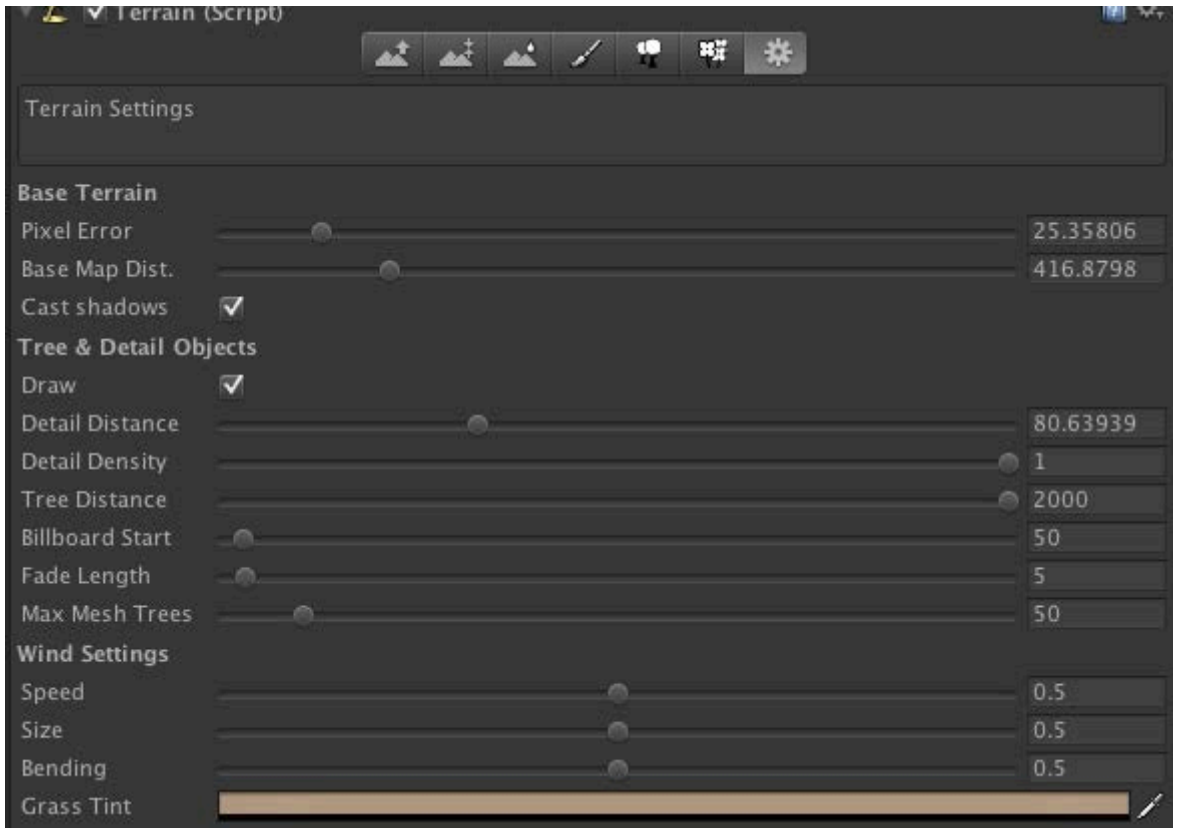
Case shadows: 是否显示地形阴影

Draw:是否绘制绘制场景模型，比如树，草等等。

Wind Settings 地形中风的设置

Speed：风速

Size：风的范围



通过本章的学习，有了这些地形知识我们快快创建属于我们的 3D 游戏地形吧，欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，哇咔咔～～～

# 第六章 Unity3D 游戏引擎之脚本实现模型的平移与旋转

这一章 MOMO 带大家讨论一下 Unity3D 中使用的脚本，脚本的最大特点就是用少量的代码实现繁多的功能，避免大量的代码。Untiy3D 这一块可以使用脚本做很多东西，那么我们开始学习脚本吧。

有关 Unity3D 脚本的 API 所有文档盆友们都可以去这里查阅。

官方 API 文档：http://unity3d.com/support/documentation/ScriptReference/

脚本描述

Scripting inside Unity consists of attaching custom script objects called behaviours to game objects. Different functions inside the script objects are called on certain events. The most used ones being the following:

Update:

This function is called before rendering a frame. This is where most game behaviour code goes, except physics code.

FixedUpdate:

This function is called once every physics time step. This is the place to do physics-based game behaviour.

Code outside any function:

Code outside functions is run when the object is loaded. This can be used to initialise the state of the script.

Note: Sections of this document assume you are using Javascript, but see Writing scripts in C# & Boo for information about how to use C# or Boo scripts.

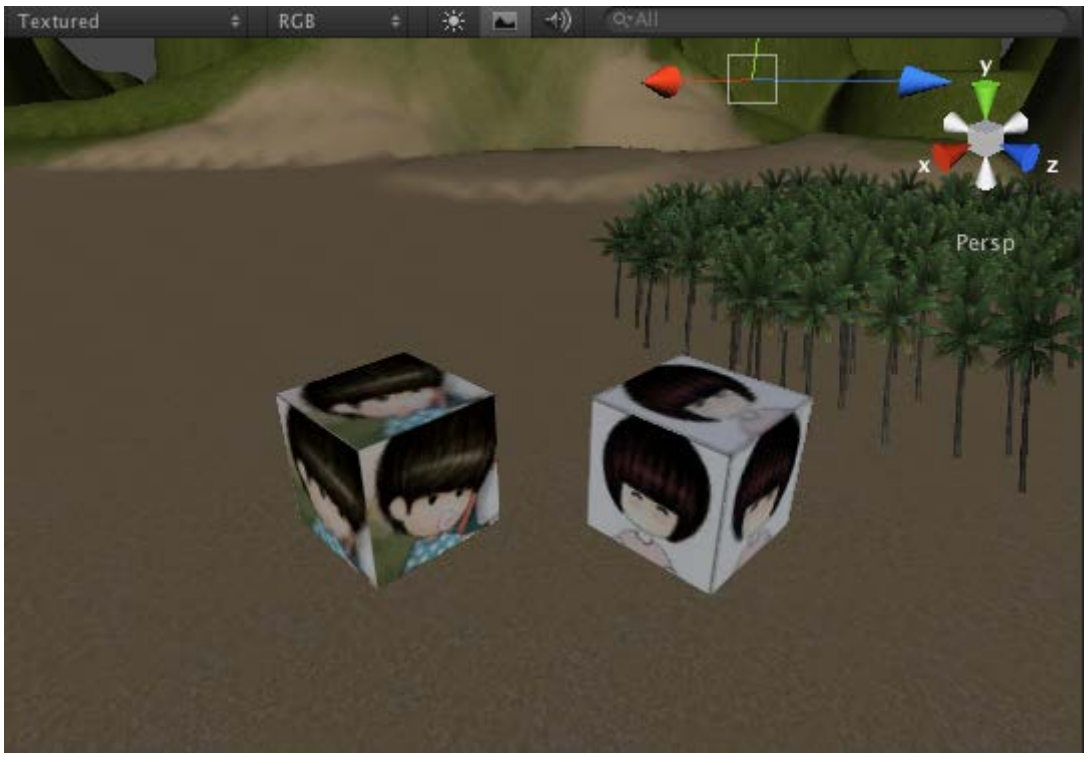
大概意思是介绍三个重要的脚本函数

Update:这个函数在渲染帧之前被调用,大部分的游戏行为代码都在这里执行,除 物理代码。

FixedUpdate:这个函数在每进行一次物理时间步调时被调用,它是基于物理的游戏行为。

Code outside any function:这类函数在对象加载时被调用,它可以用来脚本的初始化工作。

本章我们着重讨论 Update 这个函数，创建脚本与绑定脚本的方法在第二章中已经介绍过了不会的盆友请去那里阅读。虽然官方推荐脚本使用 JavaScript 编辑，但是其实 C＃更符合 Unity3D 的编程思想，推荐新人先使用 JavaScript，然后在学习 C＃，因为 JavaScript 更容易上手一些。




在三维世界中创建两个矩形，然后在添加两个脚本分别绑定在这两个箱子上，脚本的名称暂时命名为 js0 、js1。

在 Project 页面中打开刚刚创建的 js0,发现 Unity3D 已经将 Update 函数添加在脚本中了。

## 模型的移动

Translate 方法中的三个参数分别标示，模型在三维世界中 X 、Y、Z 轴移动的单位距离。

[javascript] view plaincopyprint?

```
1.  function Update () {
2.
3.  //模型 x 轴，移动一个单位
4.  transform.Translate(1,0,0);
5.
6.  //模型 y 轴，移动一个单位
7.  transform.Translate(0,1,0);
8.
9.  //模型 z 轴，移动一个单位
10. transform.Translate(0,0,1);
11.
12. }
```

```
function Update () {

//模型x轴，移动一个单位
transform.Translate(1,0,0);
```

执行代码发现参数为 1 速度居然移动的着么快，怎么能修改移动的速度呢？

Time.deltaTime：标示上一次调用 Update 一秒为标示每帧执行所消耗的时间。

有了这个参数，我们就可以根据它修改方向移动的速度了。

[javascript] view plaincopyprint?

```
1.  function Update () {
2.
3.      //设置移动的范围
4.      var translation : float = Time.deltaTime * 10;
5.
6.      //移动的方向
7.      transform.Translate (translation, 0, 0);
8.      transform.Translate (0, translation, 0);
9.      transform.Translate (0, 0, translation);
10.
11. }
```

```
function Update () {

    //设置移动的范围
    var translation : float = Time
```

模型的平移可以选择一个参照物，下面代码第二个参数设置模型移动参照物，这里设置成摄像机。那么模型将以相对与摄像机进行移动。

[javascript] view plaincopyprint?

```
1.  function Update () {
2.
3.      //设置移动范围
4.      var translation : float = Time.deltaTime * 10;
5.
6.      //相对于摄像机，模型向右移动。
7.      transform.Translate(Vector3.right * translation, Camera.main.transform);
8.
9.      // 相对于摄像机，模型向上移动。
10.     transform.Translate(Vector3.up * translation, Camera.main.transform);
11.
12.     // 相对于摄像机，模型向左移动。
13.     transform.Translate(Vector3.left * translation, Camera.main.transform);
14.
15.     }
```

```
function Update () {

    //设置移动范围
    var translation : float = Time
```

## 模型的旋转

Rotate 方法中的三个参数分别标示，模型在三维世界中 X 、Y、Z 轴旋转的单位距离。

[javascript] view plaincopyprint?

```
1.  function Update () {
2.
3.      //以模型 X 轴旋转，单位为 2.
4.      transform.Rotate(2, 0, 0);
5.
6.      //以模型 Y 轴旋转，单位为 2.
7.      transform.Rotate(0, 2, 0);
8.
9.      //以模型 Z 轴旋转，单位为 2.
10.     transform.Rotate(0, 0, 2);
11. }
```

```
function Update () {

    //以模型X轴旋转，单位为2.
    transform.Rotate(2, 0, 0);
```

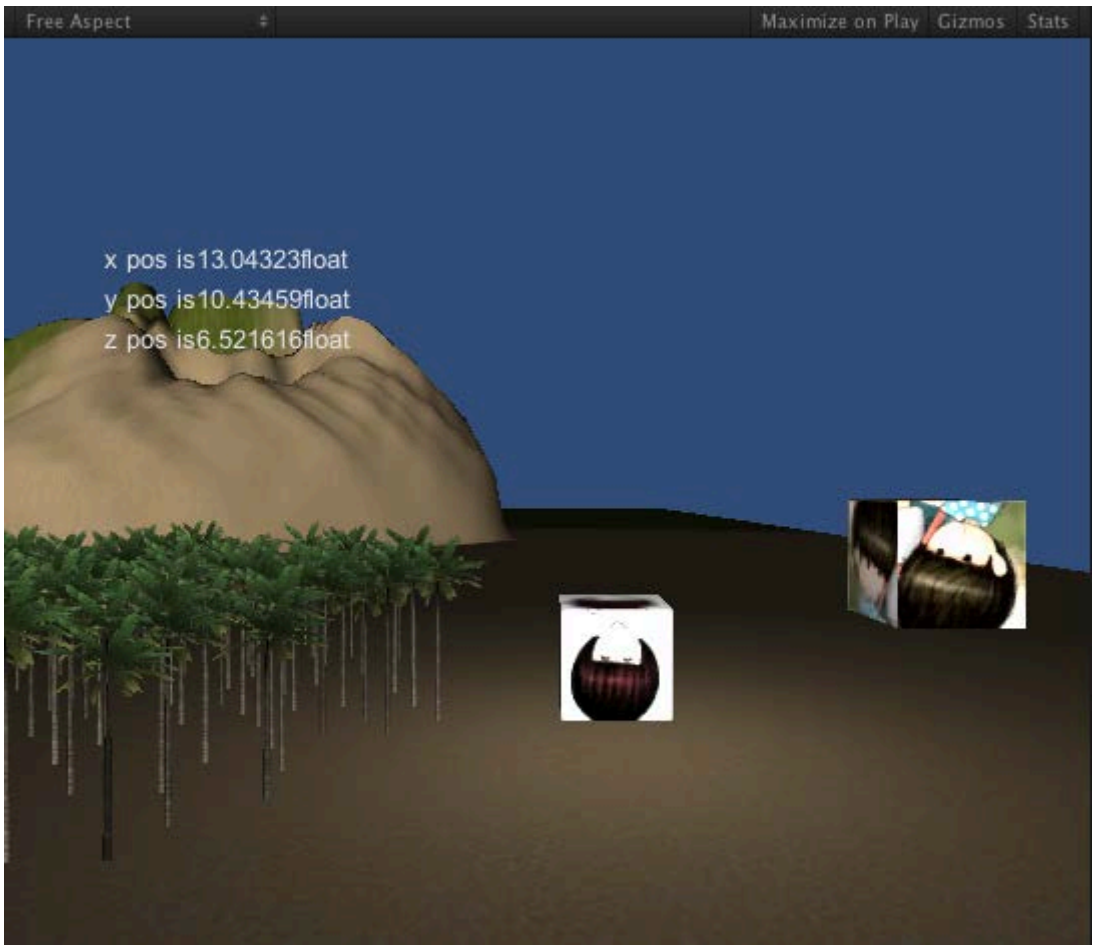
模型的旋转可以选择一个参照物，下面代码第二个参数设置模型移动参照物，这里设置成 3D 世界。那么模型将以相对与整个 3D 世界进行旋转。

[javascript] view plaincopyprint?

```
1.  function Update () {
2.
3.      //设置旋转的范围
4.       var rotate : float = Time.deltaTime * 100;
5.
6.       //旋转的方向
7.
8.       //相对于世界坐标中心向右旋转物体
9.       transform.Rotate(Vector3.right * rotate, Space.World);
10.
11.      //相对于世界坐标中心向上旋转物体
12.      transform.Rotate(Vector3.up * rotate, Space.World);
13.
14.      //相对于世界坐标中心向左旋转物体
15.      transform.Rotate(Vector3.left * rotate, Space.World);
16. }
```

```
function Update () {

   //设置旋转的范围
    var rotate : float = Time.delt
```

如下图所示，给出一个小例子，在脚本中移动箱子的坐标，在屏幕中记录模型移动的位置，并且显示在游戏视图中。效果很不错吧，嘻嘻～～




## 完整代码

[javascript] view plaincopyprint?

```
1.  //X 轴移动位置
2.  var posX : float;
3.  //Y 轴移动位置
4.  var posY : float;
5.  //Z 轴移动位置
6.  var posZ : float;
7.
8.
9.
10. function Update () {
11.
12.  //设置移动的范围
13.    var x : float = Time.deltaTime * 10;
14.    var y : float = Time.deltaTime * 8;
15.    var z : float = Time.deltaTime * 5;
16.
17.    //移动的方向 X 轴
18.    transform.Translate (x, 0, 0);
19.
20.    //移动的方向 Y 轴
21.    transform.Translate (0, y, 0);
22.    //移动的方向 Z 轴
23.    transform.Translate (0, 0, z);
24.
25.
26.    //赋值计算模型在三维坐标系中的位置
27.     posX += x;
```

```
28.       posY += y;
29.       posZ += z;
30.  }
31.
32.  function OnGUI () {
33.
34.    //将坐标信息显示在 3D 屏幕中
35.    GUI.Label(Rect(50, 100,200,20),"x pos is" + posX +"float");
36.    GUI.Label(Rect(50, 120,200,20),"y pos is" + posY +"float");
37.    GUI.Label(Rect(50, 140,200,20),"z pos is" + posZ +"float");
38.
39.  }
```

```
//X轴移动位置
var posX : float;
//Y轴移动位置
var posY : float;
//Z轴移动位置
```

Unity3D 的世界中脚本还可以做很多事情，以后我在慢慢向各位道来～ 欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，哇咔咔～～～

# 第七章 Unity3D 游戏引擎之控制模型移动旋转与碰撞

上一章介绍了模型的旋转与平移，本章介绍一些好玩的，通过按钮来控制模型的移动与镜头的跟随，脚本参数的传递。个人觉得 Unity3D 这套游戏引擎真的很棒，它可以为开发者节省时间。用更多少的脚本，可以实现更多的功能。先赞一下，哇咔咔～废话不多说了，入正题！

首先在 3D 世界中创建一个箱子模型，然后添加一个脚本，命名为 js0 绑定到这个箱子中。在添加一个脚本，命名为 js1 ，同样绑定到这个箱子上，用来实现对象的创建与调用方法参数的传递 ~

var Control;  这里声明一个控制类的对象，用来进行参数的传递,调用方法。

Getomponent(js1)：得到脚本名称为 js1 的对象，这里把值赋给 Control，就可以使用 js1 脚本中的方法了，下面的代码中通过这个对象调用模型的移动方法。

Input.GetKey(KeyCode)：这个方法返回一个 Bool 判断当前按键是否被按下，当然这里只能运行在 PC 上才可以感应按钮，因为 IOS 的设备上没有按键，以后我会介绍在 IOS 上添加游戏摇杆的功能，本章我们先学习一下 PC 上的基本按钮响应。

下面代码

实现按下“W”控制物体向前

实现按下“S”控制物体向后

实现按下“A”控制物体向左平移

实现按下“D”控制物体向右平移

实现按下“Q”控制物体向左旋转

实现按下“E”控制物体向右旋转

js0.js 代码

[javascript] view plaincopyprint?

```
1.  //旋转移动速度
2.  var Speed = 50;
3.
4.  //控制对象
5.  var Control;
6.
7.  function Update()
8.  {
9.    //得到控制对象
10.   Control = GetComponent(js1);
11.
12.   //判断按键
13.
14.   if(Input.GetKey(KeyCode.W))
15.   {
16.       //前进
17.       Control.ForWard();
18.
19.   }else if(Input.GetKey(KeyCode.S))
20.   {
21.       //后退
22.       Control.Back();
23.   }
```

```
24.
25.     if(Input.GetKey(KeyCode.A))
26.    {
27.         //前左
28.         Control.GLeft();
29.
30.    }else if(Input.GetKey(KeyCode.D))
31.    {
32.         //后右
33.         Control.GRight();
34.    }
35.
36.
37.
38.    if(Input.GetKey(KeyCode.Q))
39.    {
40.      //左旋转
41.
42.       Control.leftRotate(Vector3.up *Time.deltaTime * -Speed);
43.
44.    }else if(Input.GetKey(KeyCode.E))
45.    {
46.      //右旋转
47.       Control.RightRotate(Vector3.up *Time.deltaTime * Speed);
48.
49.    }
50.
51.  }
```

```
//旋转移动速度
var Speed = 50;

//控制对象
var Control;
```

js1.js 代码

参数的传递可以使用 obj 进行传递，代码中我将旋转的角度做为参数传递到了 js1.js 中控制模型旋转。

Vector3 标示一个 3D 的向量单位，它可以标示模型的三个方向，这里根据旋转的角度与移动的向量设置它。

[javascript] view plaincopyprint?

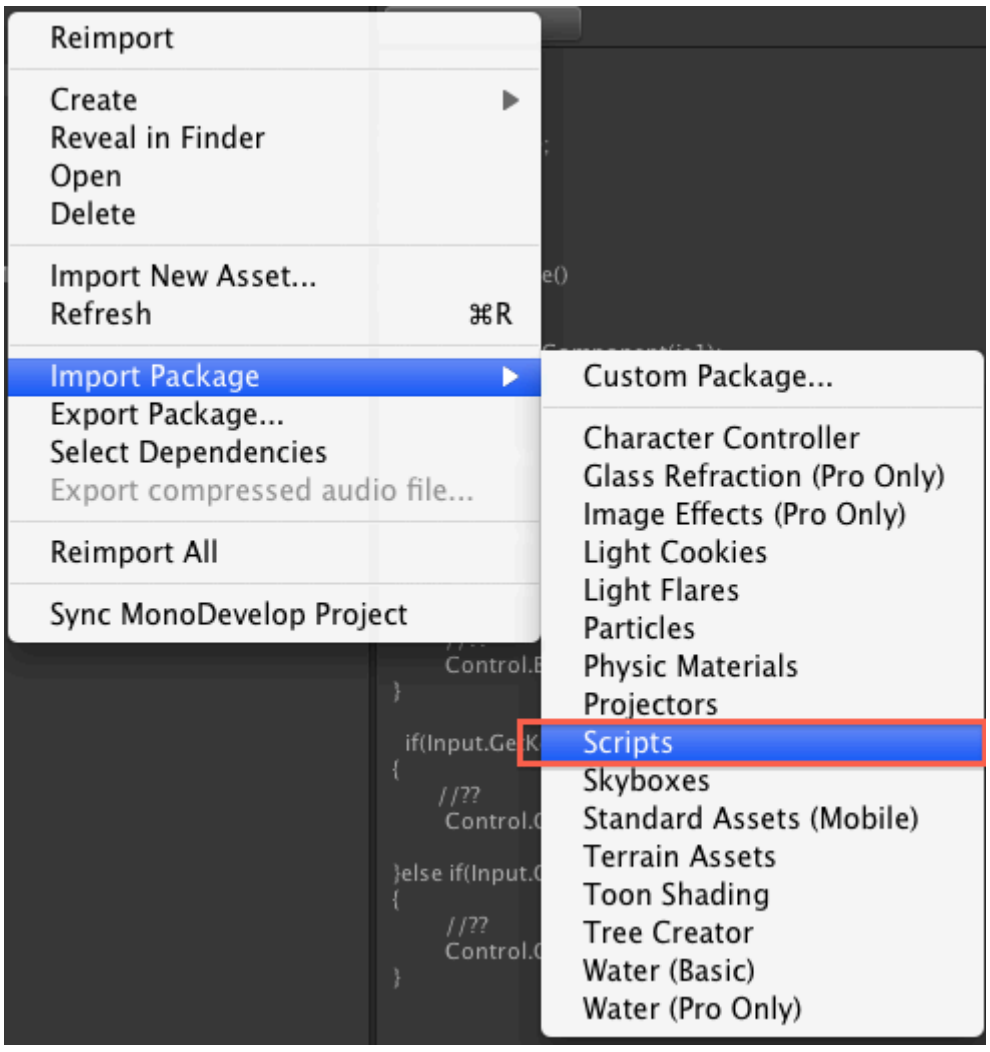
```
1.  //前进速度
2.  var Speed = 50;
3.
4.  //前进
5.  function ForWard()
6.  {
7.      transform.Translate(Vector3.forward * Time.deltaTime *Speed);
```

```
8.    }
9.
10.  //后退
11.  function Back()
12.  {
13.      transform.Translate(Vector3.forward * Time.deltaTime * -Speed);
14.  }
15.
16.
17.  //向左
18.  function GLeft()
19.  {
20.      transform.Translate(Vector3.right * Time.deltaTime *-Speed);
21.  }
22.
23.  //向右
24.  function GRight()
25.  {
26.      transform.Translate(Vector3.right * Time.deltaTime * Speed);
27.  }
28.
29.
30.
31.
32.  //传递参数
33.
34.  //左旋转
35.  function leftRotate(obj)
36.  {
37.      transform.Rotate(obj);
38.  }
39.
40.  //右旋转
41.  function RightRotate(obj)
42.  {
43.      transform.Rotate(obj);
44.  }
```

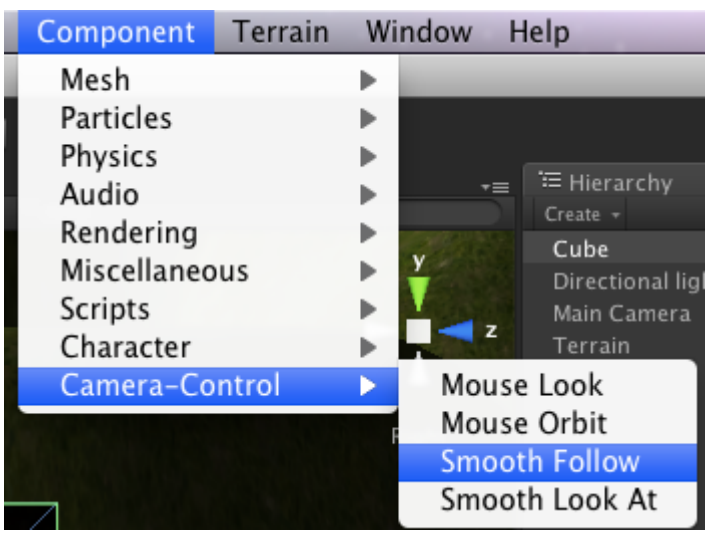
```
//前进速度
var Speed = 50;

//前进
function ForWard()
```

OK 点击运行，通过按键我们可以正常的控制模型平移与旋转，但是现在有两个问题 1， 主摄像机不能跟随控制模型， 2 、模型可以横穿过山丘，感觉很假。下面我们解决这两个问题。

在 Unity3D 标准资源的脚本中系统帮我们写好了跟随脚本.如下图所示，点击 Import Package - > Scripts 将系统脚本导入。

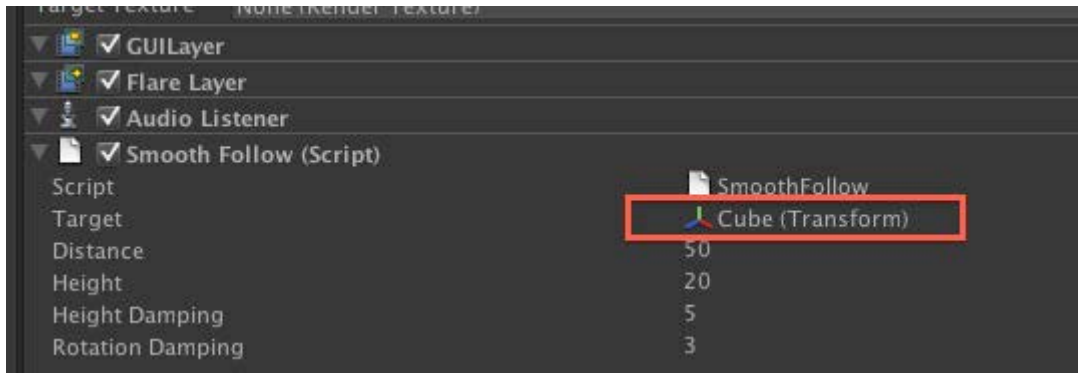



选中摄像机 点击 Component - > Camera-Control - > Smoot Follow ，给主摄像机添加一个跟随脚本。

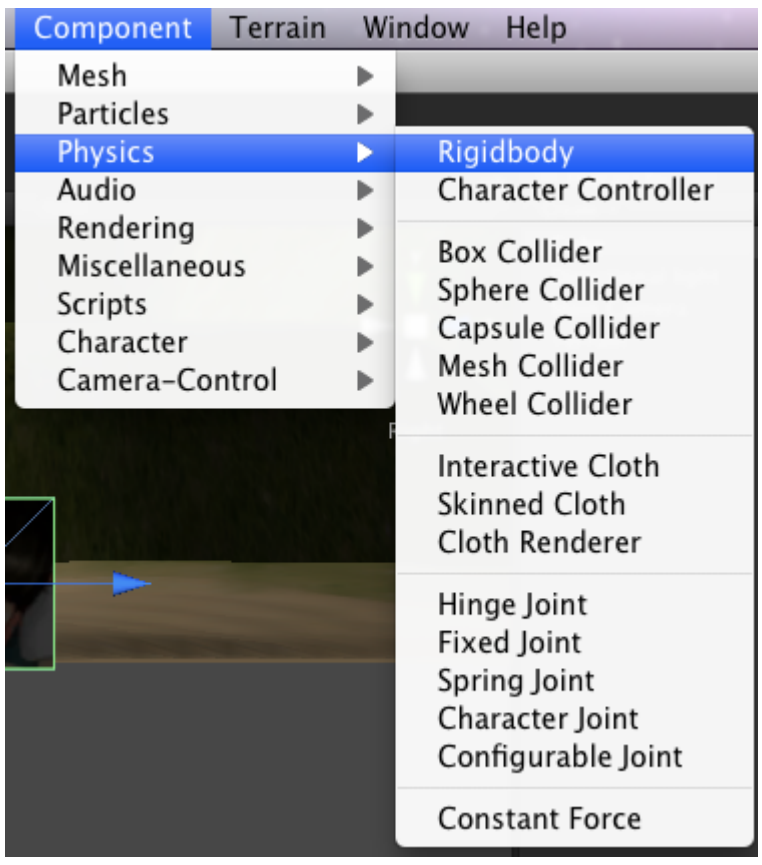



跟随脚本添加完以后，须要指定一个跟随的物体，下面红框内将跟随的物体 Cube 拖动进来，这里 Cube 就是我们上面控制的模型，那么这样，摄像机将会一直在身后

跟随我们控制的模型。下面是一些跟随的参数，设置 跟随的距离 高度等等，盆友们手动的修改一下运行游戏就可以看出来变化 。




控制物体的碰撞可以使用 选中模型， Component - > Physics - > Rigidbody 设置这个模型的重力碰撞。那么这样你控制的模型就不会穿过山丘，而是感应重力碰撞。



点击运行游戏，看看效果，控制箱子行走的同时，我们可以清晰的看出旁边的树木被物理碰撞所干扰，前后，左右，旋转 完全 OK .大家快试试吧。哇咔咔～～

明天开始 MOMO 又要开始处理一部分 iPhone 的界面图形化开发，Tomorrow is another day ,加油哇咔咔～ Unity3D 的世界中脚本还可以做很多事情，以后我在慢慢向各位道来～ 欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，哇咔咔～～～

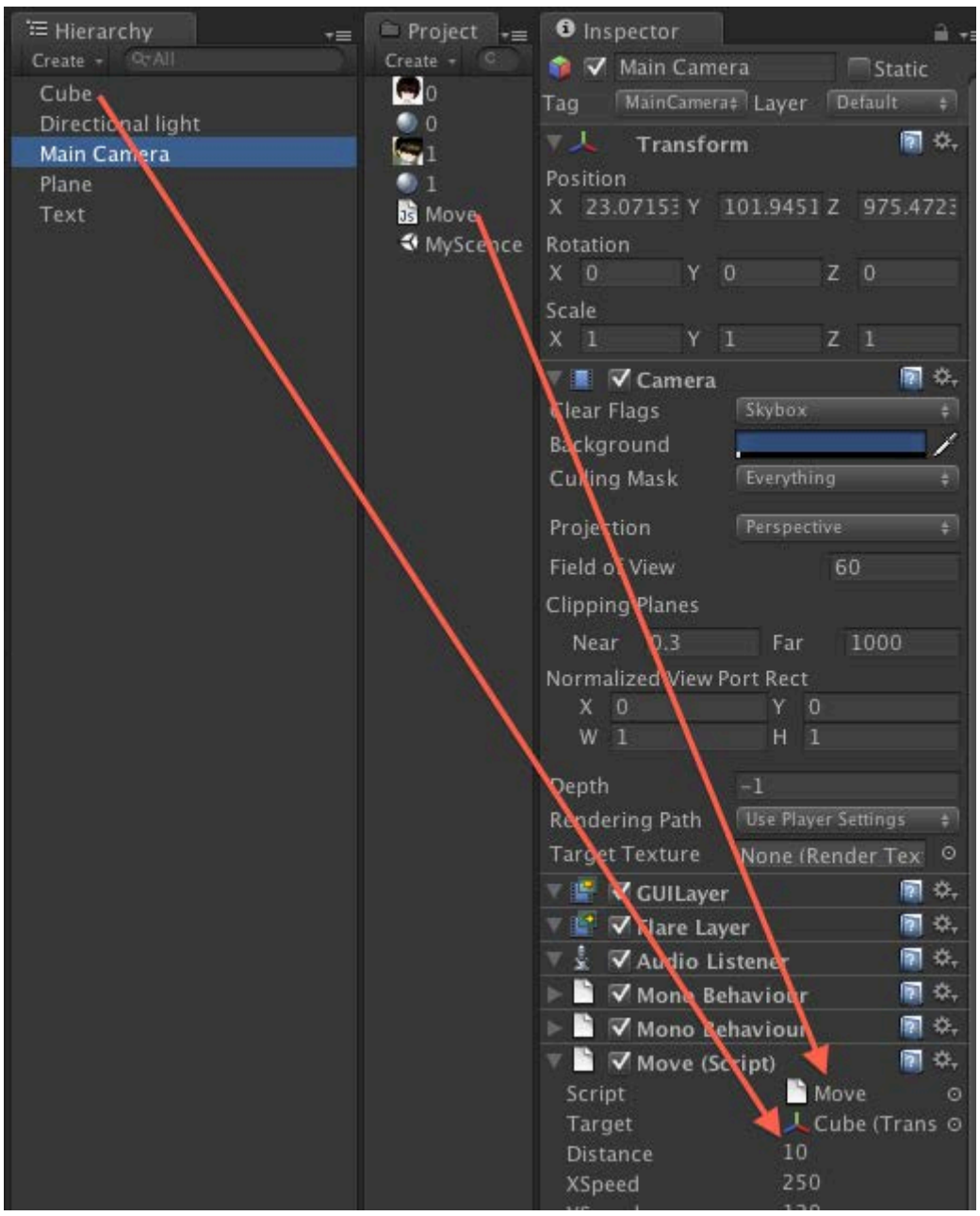
# 第八章 Unity3D 游戏引擎之 IOS 触摸屏手势控制镜头旋转与缩放

前几篇文章介绍了很多 Unity3D 引擎自身的一些问题， 今天我们在回到 IOS 设备上讨论一些触摸屏幕手势，本章的目标是通过触摸 iPhone 屏幕手势 实现模型左右的旋转，与模型的缩放。

大家想一想模型的旋转，实际上是镜头的旋转。模型的缩放实际上是镜头 Z 轴方向的坐标。那么实现本章的内容只需要控制镜头的位置方可实现。

我们在游戏场景中创建一些简单的模型做为参照物，插一句 “大家有谁知道 .FBX 的模型那里可以免费找到，我想在博文中加点游戏模型让场景更好看一些，可是苦于找不到有点郁闷，用公司的模型来写博文有觉得有点不合适”哇咔咔 ，如果有知道的哥们 不妨告诉我喔。啦啦啦。

我们创建一个简单的游戏平面， 然后平面中放一个箱子做为旋转缩放的参照物。如下图所示，选中摄像机，给摄像机添加一个脚本名称为 Move. 脚本中有一个参数 Target，它的作用是设置摄像头旋转移动参照物，这里把一个箱子赋值给了 Target，那么左右滑动屏幕会发现箱子在旋转，两手缩放屏幕会发现箱子在放大与缩小。




我们看看 Move 这条脚本，说明一下几个重要的 ：

这些方法都是系统自己调用的方法

function Start () ： 游戏启动以后只调用一次，可用于脚本的初始化操作，

function Update ()：Start()方法调用结束以后每一帧都会调用，可以在这里更新游戏逻辑。

function LateUpdate (): Start()方法调用结束以后每一帧都会调用，但是它是在 Update()调用完后调用。

Move.js 完整代码

[javascript] view plaincopyprint?

```
1.  //用于绑定参照物对象
2.  var target : Transform;
3.  //缩放系数
4.  var distance = 10.0;
5.  //左右滑动移动速度
6.  var xSpeed = 250.0;
7.  var ySpeed = 120.0;
8.  //缩放限制系数
9.  var yMinLimit = -20;
10. var yMaxLimit = 80;
11. //摄像头的位置
12. var x = 0.0;
13. var y = 0.0;
14. //记录上一次手机触摸位置判断用户是在左放大还是缩小手势
15. private var oldPosition1 : Vector2;
16. private var oldPosition2 : Vector2;
17.
18.
19. //初始化游戏信息设置
20. function Start () {
21.     var angles = transform.eulerAngles;
22.     x = angles.y;
23.     y = angles.x;
24.
25.     // Make the rigid body not change rotation
26.     if (rigidbody)
27.         rigidbody.freezeRotation = true;
28. }
29.
30.
31. function Update ()
32. {
33.     //判断触摸数量为单点触摸
34.     if(Input.touchCount == 1)
35.     {
36.         //触摸类型为移动触摸
37.         if(Input.GetTouch(0).phase==TouchPhase.Moved)
38.         {
39.             //根据触摸点计算 X 与 Y 位置
40.             x += Input.GetAxis("Mouse X") * xSpeed * 0.02;
41.             y -= Input.GetAxis("Mouse Y") * ySpeed * 0.02;
42.
43.         }
44.     }
45.
46.     //判断触摸数量为多点触摸
47.     if(Input.touchCount >1 )
48.     {
49.         //前两只手指触摸类型都为移动触摸
50.         if(Input.GetTouch(0).phase==TouchPhase.Moved||Input.GetTouch(1).phase==TouchPhase.Moved)
```

```
51.     {
52.         //计算出当前两点触摸点的位置
53.         var tempPosition1 = Input.GetTouch(0).position;
54.         var tempPosition2 = Input.GetTouch(1).position;
55.         //函数返回真为放大，返回假为缩小
56.         if(isEnlarge(oldPosition1,oldPosition2,tempPosition1,tempPosition2))
57.         {
58.             //放大系数超过 3 以后不允许继续放大
59.             //这里的数据是根据我项目中的模型而调节的，大家可以自己任意修改
60.             if(distance > 3)
61.             {
62.                 distance -= 0.5;
63.             }
64.         }else
65.         {
66.             //缩小洗漱返回 18.5 后不允许继续缩小
67.             //这里的数据是根据我项目中的模型而调节的，大家可以自己任意修改
68.             if(distance < 18.5)
69.             {
70.                 distance += 0.5;
71.             }
72.           }
73.         //备份上一次触摸点的位置，用于对比
74.         oldPosition1=tempPosition1;
75.         oldPosition2=tempPosition2;
76.       }
77.   }
78. }
79.
80.
81.
82. //函数返回真为放大，返回假为缩小
83. function isEnlarge(oP1 : Vector2,oP2 : Vector2,nP1 : Vector2,nP2 : Vector2) : boolean
84. {
85.   //函数传入上一次触摸两点的位置与本次触摸两点的位置计算出用户的手势
86.   var leng1 =Mathf.Sqrt((oP1.x-oP2.x)*(oP1.x-oP2.x)+(oP1.y-oP2.y)*(oP1.y-oP2.y));
87.   var leng2 =Mathf.Sqrt((nP1.x-nP2.x)*(nP1.x-nP2.x)+(nP1.y-nP2.y)*(nP1.y-nP2.y));
88.   if(leng1<leng2)
89.   {
90.       //放大手势
91.       return true;
92.   }else
93.   {
94.       //缩小手势
95.       return false;
96.   }
97. }
98.
99. //Update 方法一旦调用结束以后进入这里算出重置摄像机的位置
100.function LateUpdate () {
101.
102.   //target 为我们绑定的箱子变量，缩放旋转的参照物
103.   if (target) {
```

```
104.
105.        //重置摄像机的位置
106.        y = ClampAngle(y, yMinLimit, yMaxLimit);
107.        var rotation = Quaternion.Euler(y, x, 0);
108.        var position = rotation * Vector3(0.0, 0.0, -distance) + target.position;
109.
110.        transform.rotation = rotation;
111.        transform.position = position;
112.    }
113.}
114.
115.
116.static function ClampAngle (angle : float, min : float, max : float) {
117.    if (angle < -360)
118.        angle += 360;
119.    if (angle > 360)
120.        angle -= 360;
121.    return Mathf.Clamp (angle, min, max);
122.}
```

```
//用于绑定参照物对象
var target : Transform;
//缩放系数
var distance = 10.0;
//左右滑动移动速度
```

在 Untiy3D 中运行用鼠标手势点击上看不到任何效果的，必需在 iPhone 真机上才可以触摸感应到效果喔 . 嘻嘻~

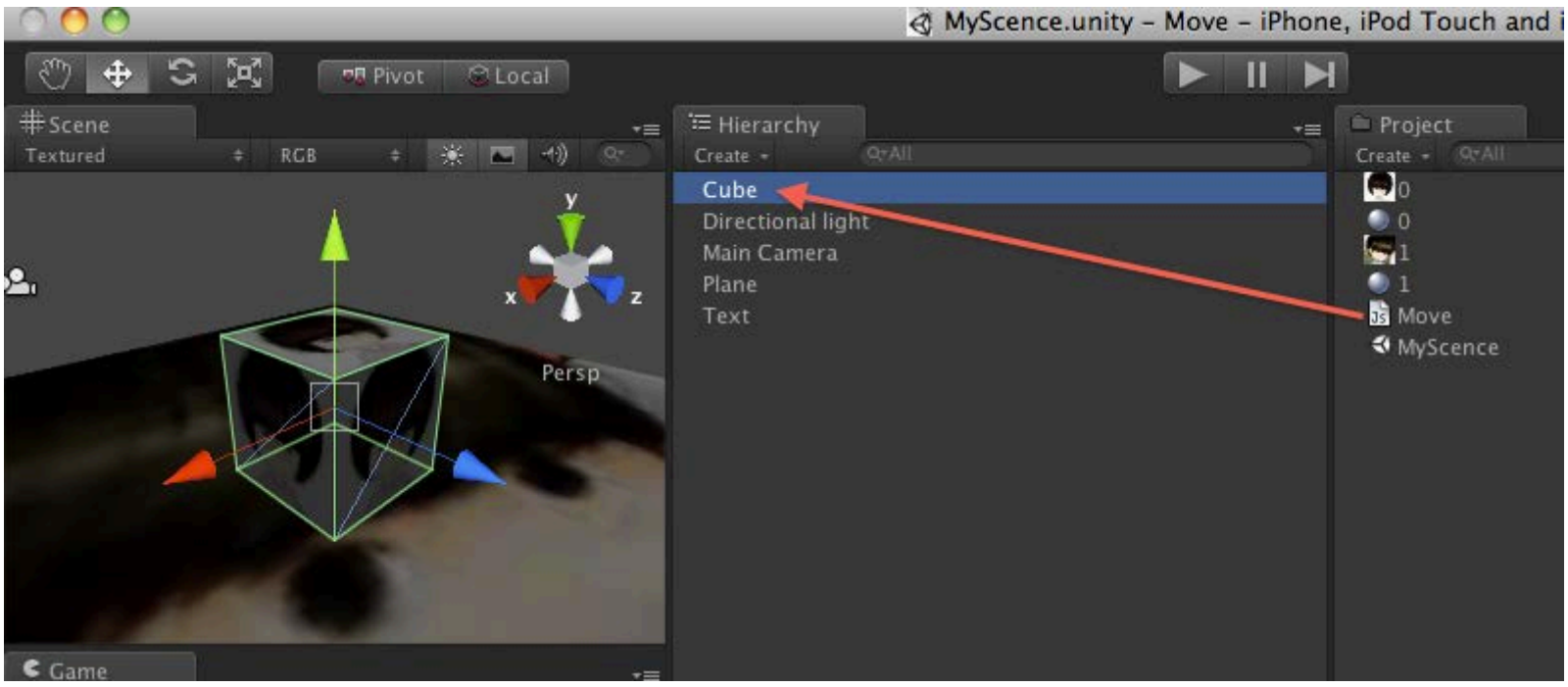
OK 接下来将 Unity3D 导出成 Xcode 项目，导出的方法有谁还不会？？？ 我的第一篇文章有说明噢。哇咔咔~ ~ 然后运行项目只能在真机上运行哦。。 看看我在 iPhone 上的截图。

# 第九章 Unity3D 游戏引擎之 IOS 高级界面发送消息与 Unity3D 消息的接收

刚搬家了新家超累，大家久等了。哇咔咔~ 今天和盆友们讨论 IOS 的高级界面与 Unity3D 游戏引擎的交互，这个在开发中是非常重要的，Unity3D 毕竟是一个面向多平台的一个游戏引擎，它不可能全部为 IOS 考虑的面面俱到，引擎中也不存在针对 IOS 的高级界面的控件的使用。

本例实现游戏背景是 Unity3D 的游戏世界，前面添加 4 个 IOS 的高级界面的按钮，并且点击这些按钮可以将消息传递给背景的 Unity3D ，让它做一些事情。

上一章介绍了触摸 IOS 屏幕 移动摄像机的位置，下面有盆友问我说他不想移动摄像机的位置，就想移动物体的位置，我在这里补充一下，可以把脚本绑定在箱子上，参照物选择为主摄像机，这样子更新箱子的脚本就 OK 啦。今天例子，我就将脚本绑定在箱子上，如下图所示，把 Move 脚本绑定在这个 Cube 中。




先把 Move 脚本的代码贴出来，这里面我写了 4 个方法分别处理这个箱子的旋转，这 4 个方法是由 IOS 上的代码向 Unity 发送消息后调用的，下面我会介绍具体操作的方法。

[javascript] view plaincopyprint?

```
1.  var vrotate : Vector3;
2.
3.  //向左旋转
4.  function MoveLeft()
5.  {
6.      var rotate : float = Time.deltaTime * 100;
7.      vrotate = Vector3.up * rotate;
8.      transform.Rotate(vrotate, Space.World);
9.  }
10.
11. //向右旋转
12. function MoveRight()
13. {
```

```
14.     var rotate : float = Time.deltaTime * 100;
15.     vrotate = Vector3.down* rotate;
16.     transform.Rotate(vrotate, Space.World);
17. }
18.
19. //向上旋转
20. function MoveUp(){
21.     var rotate : float = Time.deltaTime * 100;
22.     vrotate = Vector3.right* rotate;
23.     transform.Rotate(vrotate, Space.World);
24. }
25.
26. //向下旋转
27. function MoveDown(){
28.     var rotate : float = Time.deltaTime * 100;
29.     vrotate = Vector3.left* rotate;
30.     transform.Rotate(vrotate, Space.World);
31. }
```

```
var vrotate : Vector3;

//向左旋转
function MoveLeft()
{
```

到这里盆友们可以将这个 Unity 工程导出成 Xcode 项目，不会的盆友请看我之前的文章哈，Xcode 项目导出成功后，我们先添加 4 个高级界面的按钮用来点击响应上面脚本的这 4 个旋转箱子的方法。

创建一个类继承 UIViewController，用来添加我们的高级界面的视图，我暂且命名为 MyView.

打开 Unity3D 导出的 AppController.mm 这个类，头文件处先导入我们的这个类 #import "MyView"

找到下面这个方法，来添加 view

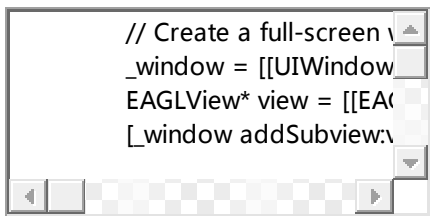
int OpenEAGL_UnityCallback(UIWindow** window, int* screenWidth, int* screenHeight,  int* openglesVersion)

EAGLView 是 Unity3D 背景的那个 View，下面我们添加一个我们自己写的 View 覆盖在它上面。

[cpp] view plaincopyprint?

```
1.  // Create a full-screen window
2.  _window = [[UIWindow alloc] initWithFrame:rect];
3.  EAGLView* view = [[EAGLView alloc] initWithFrame:rect];
4.  [_window addSubview:view];
5.
6.  MyView * myView =  [[MyView alloc] init];
```

```
7.  [_window addSubview:myView.view];
```




贴出 MyView 的代码，写完发现忘释放内存了，呵呵，懒得改了，本章主要的介绍的不是这个哦。

[cpp] view plaincopyprint?

```
1.  //
2.  //  MyView.m
3.  //  Unity-iPhone
4.  //
5.  //  Created by 雨松 MOMO on 11-11-1.
6.  //  Copyright 2011 __MyCompanyName__. All rights reserved.
7.  //
8.
9.  #import "MyView.h"
10.
11.
12. @implementation MyView
13.
14.
15. // Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
16. - (void)viewDidLoad {
17.     [super viewDidLoad];
18.     //创建 label 视图
19.     UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(0, 0, 320, 40)];
20.     //设置显示内容
21.     label.text = @"雨松 MOMO 的程序世界";
22.     //设置背景颜色
23.     label.backgroundColor = [UIColor blueColor];
24.     //设置文字颜色
25.     label.textColor = [UIColor whiteColor];
26.     //设置显示位置居中
27.     label.textAlignment = UITextAlignmentCenter;
28.     //设置字体大小
29.     label.font = [UIFont fontWithName:[[UIFont familyNames] objectAtIndex:10] size:20];
30.
31.     //创建按钮
32.     UIButton *button0 = [UIButton buttonWithType:1];
33.     //设置按钮范围
34.     button0.frame = CGRectMake(0, 40, 100, 30);
35.     //设置按钮显示内容
36.     [button0 setTitle:@"矩形左旋转" forState:UIControlStateNormal];
37.     //设置按钮改变后 绑定响应方法
38.     [button0 addTarget:self action:@selector(LeftButtonPressed) forControlEvents:UIControlEventTouchUpInside];
39.
```

```
40.     //创建按钮
41.     UIButton *button1 = [UIButton buttonWithType:1];
42.     //设置按钮范围
43.     button1.frame = CGRectMake(0, 100, 100, 30);
44.     //设置按钮显示内容
45.     [button1 setTitle:@"矩形右旋转" forState:UIControlStateNormal];
46.     //设置按钮改变后 绑定响应方法
47.     [button1 addTarget:self action:@selector(RightButtonPressed) forControlEvents:UIControlEventTouchUpInside];
48.
49.     //创建按钮
50.     UIButton *button2 = [UIButton buttonWithType:1];
51.     //设置按钮范围
52.     button2.frame = CGRectMake(0, 160, 100, 30);
53.     //设置按钮显示内容
54.     [button2 setTitle:@"矩形上旋转" forState:UIControlStateNormal];
55.     //设置按钮改变后 绑定响应方法
56.     [button2 addTarget:self action:@selector(UpButtonPressed) forControlEvents:UIControlEventTouchUpInside];
57.
58.     //创建按钮
59.     UIButton *button3 = [UIButton buttonWithType:1];
60.     //设置按钮范围
61.     button3.frame = CGRectMake(0, 220, 100, 30);
62.     //设置按钮显示内容
63.     [button3 setTitle:@"矩形下旋转" forState:UIControlStateNormal];
64.     //设置按钮改变后 绑定响应方法
65.     [button3 addTarget:self action:@selector(DownButtonPressed) forControlEvents:UIControlEventTouchUpInside];
66.
67.
68.     //向 view 添加
69.     [self.view addSubview:label];
70.     [self.view addSubview:button0];
71.     [self.view addSubview:button1];
72.     [self.view addSubview:button2];
73.     [self.view addSubview:button3];
74. }
75.
76. //向左按钮
77. -(void)LeftButtonPressed{
78.     UnitySendMessage("Cube","MoveLeft","");
79. }
80.
81. //向右按钮
82. -(void)RightButtonPressed{
83.     UnitySendMessage("Cube","MoveRight","");
84. }
85. //向上按钮
86. -(void)UpButtonPressed{
87.     UnitySendMessage("Cube","MoveUp","");
88. }
89.
90. //向下按钮
91. -(void)DownButtonPressed{
92.     UnitySendMessage("Cube","MoveDown","");
```

```
93.  }
94.
95.
96.
97.  - (void)didReceiveMemoryWarning {
98.      // Releases the view if it doesn't have a superview.
99.      [super didReceiveMemoryWarning];
100.
101.    // Release any cached data, images, etc. that aren't in use.
102.}
103.
104.- (void)viewDidUnload {
105.    [super viewDidUnload];
106.}
107.
108.
109.- (void)dealloc {
110.    [super dealloc];
111.}
112.
113.
114.@end
```

```
//
//  MyView.m
//  Unity-iPhone
//
//  Created by 雨松MOMO on 1
```

这里我主要说一下下面这个方法，它是 Unity 底层帮我们写好的一个方法，意思 iPhone 向向 Unity 发送消息，

参数 1：场景中的模型名称，Cube 就是我们定义的一个箱子。

参数 2：脚本方法名称 MoveLeft 就是上面脚本中的方法，

参数 3：为一个 char *类型的  可以向 Unity 中传递数据。

```
UnitySendMessage("Cube","MoveLeft","");
```

我们可以向 Unity3D 中任意模型发送消息调用它绑定的脚本中的方法，当前前提是模型名称、方法名称、  参数都填写正确。

这里 4 个按钮都是以这种方式传递消息,下面是 iPhone  真机的效果图，我们触摸点击 4 个高级界面的按钮可以实现 Unity3D 世界中的模型旋转，  所以大家一定要切记这个方法，很重要噢，哇咔咔～


最后欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，哇咔咔～～～ 附上 Unity3D 工程的下载地址，Xcode 项目我就不上传了，不早了，大家晚安，哇咔咔～～

下载地址：http://download.csdn.net/detail/xys289187120/3744724

补充： 由于 Unity3.5 在渲染 3D 的时候添加了 sGLViewController，所以按照以前的方法添加的视图是无法接收旋转事件的。对应 3.5 的版本大家需要修改一下代码。

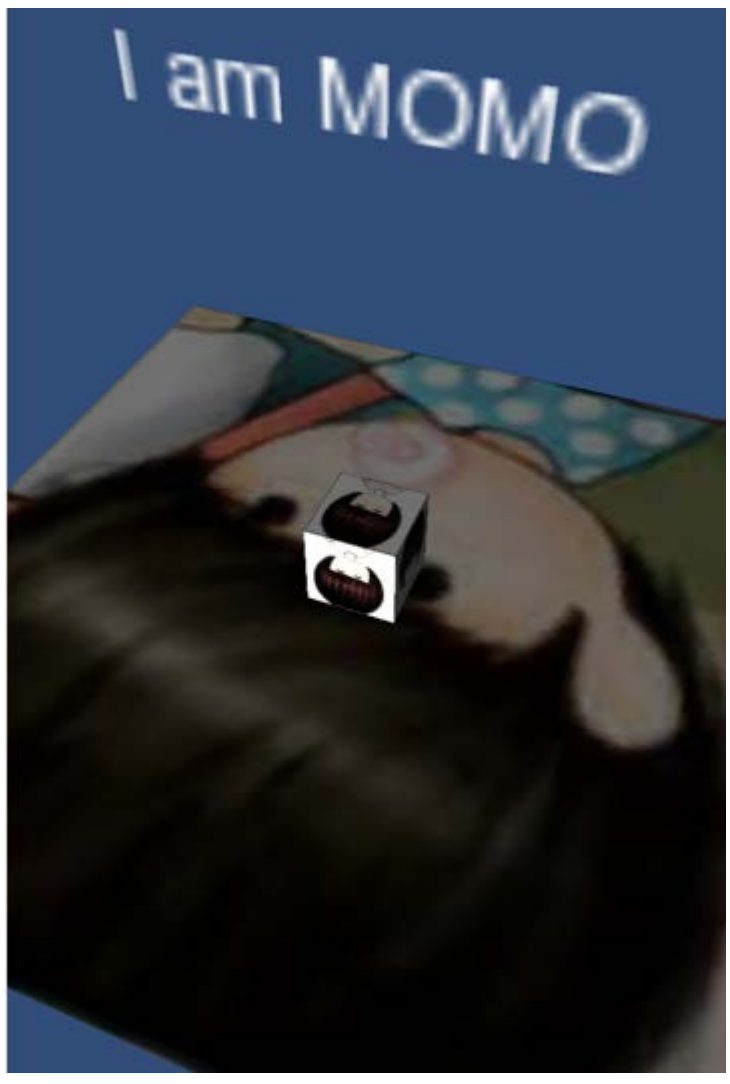
还是在 OpenEAGL_UnityCallback 方法中，在此方法的末尾添加代码：

```
MyViewController * myView =  [[MyViewController alloc] init];
[sGLViewController.view addSubview:myView.view];
```

MyViewController 是我们新定义的，

也就是说把我们的写的视图添加至

sGLViewController 当中，这样就完事 OK 啦。 Unity 每次升级都会小改动一下，所以我们也需要小改动一下，哈哈！

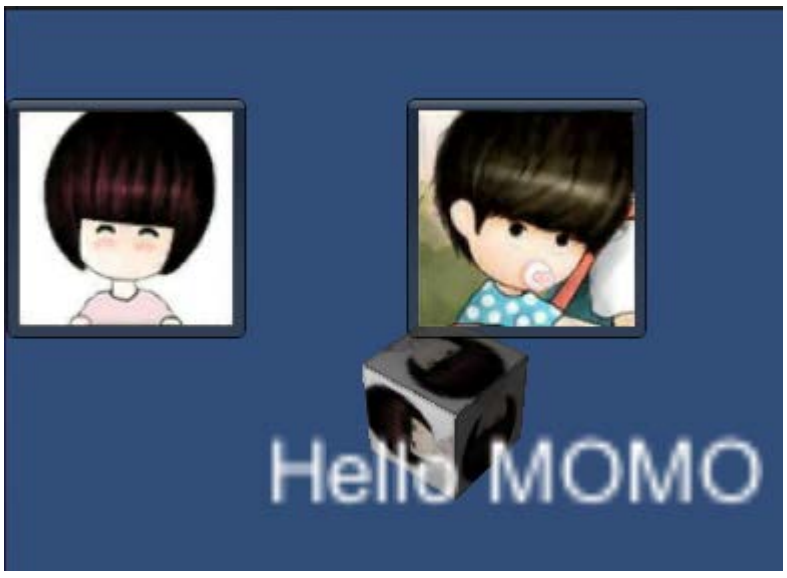



触摸 iPhone 屏幕镜头旋转与缩放以后的效果，这个图确实有点不太好截，我一会把项目的源代码工程下载地址贴上来，方便大家阅读与学习，哇咔咔～～

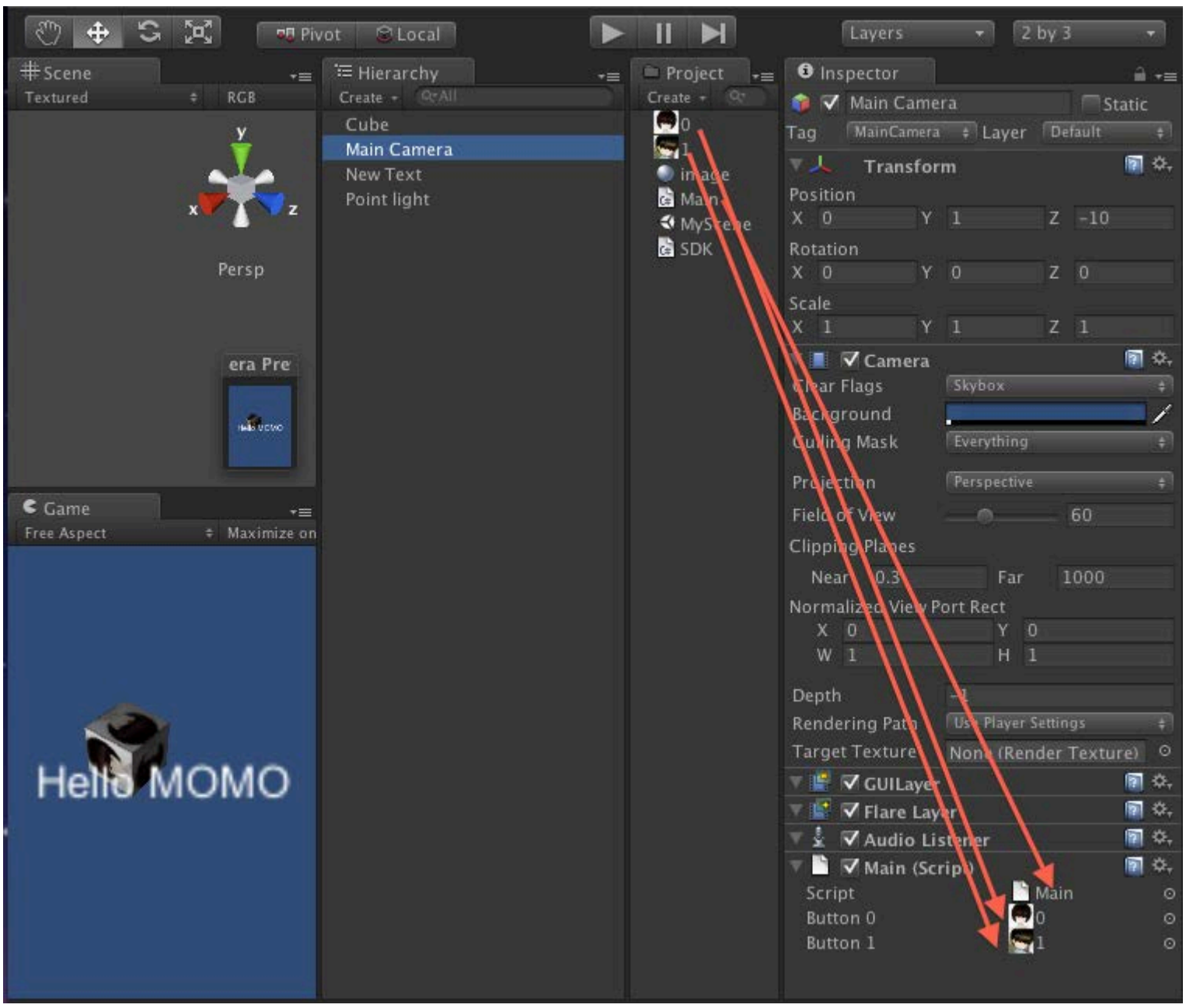
# 第十章 Unity3D 游戏引擎之 Unity3D 回馈 IOS 高级界面消息

上一章介绍了 IOS 高级界面向 Unity3D 发送消息与 Unity3D 接收消息的过程，有去的消息当然要有回的消息这样的过程才算完美，本章 MOMO 向大家介绍 Unity3D 消息的回馈。

如下图所示，本章我们的目标是在 Unity3D 界面中添加两个 GUI 按钮，并且在 iPhone 上点击这两个按钮后分别弹出两个 IOS 高级界面的对话框。相信盆友们对 GUI 应该不会太陌生，在这里我在前调一下 GUI 就是 Unity3D 提供的一套 UI 系统。图中的两个 UI 按钮我就是用 GUI 做出来的。



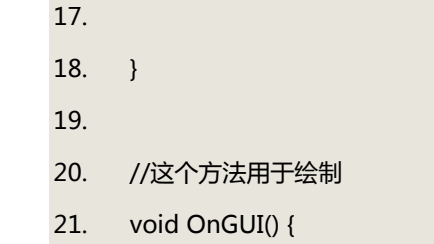

Project 栏目中创建一个 c#脚本，命名为 Main.cs ，之前没有使用过 C＃写脚本，今天我用 C#来写这个脚本，哇咔咔～～～ 如下图所示将脚本拖动在摄像机上，脚本中声明两个 Texture 类型变量用来保存按钮绘制的图片资源。




Main.cs 代码

[csharp] view plaincopyprint?

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class Main : MonoBehaviour {
5.
6.  //声明两个 Texture 变量，图片资源在外面连线赋值
7.  public Texture Button0;
8.  public Texture Button1;
9.
10.     // Use this for initialization
11.     void Start () {
12.
13.     }
14.
15.     // Update is called once per frame
16.     void Update () {
17.
18.     }
19.
20.     //这个方法用于绘制
21.     void OnGUI() {
22.         //绘制两个按钮
23.         if(GUI.Button(new Rect(0,44,120,120),Button0))
24.         {
25.             //返回值为 ture 说明这个按钮被点击
26.             SDK.ActivateButton0();
27.         }
28.
29.         //绘制两个按钮
30.         if(GUI.Button(new Rect(200,44,120,120),Button1))
31.         {
32.             //返回值为 ture 说明这个按钮被点击
33.             SDK.ActivateButton1();
34.         }
35.     }
36. }
```

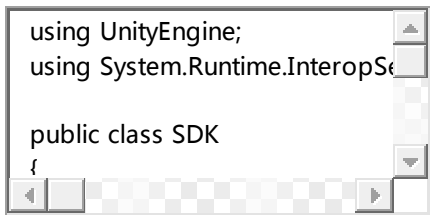



这里详细说一下 SDK 这个类，这个类我们看作它是一个管理类，它不赋值在任意对象身上，只接受调用管理，点击两个按钮后将分别调用下面方法中的_ActivateButton0() 与 _ActivateButton1()，而这两个方法则是去调用 xcode 我们自己实现的方法_pressButton0() 与 _pressButton1()，前提上须下面代码中的注册 ,这样子导出项目的时候 xcode 会帮我们 生成注册信息 ,我们只须要实现这两个方法就可以了。

## SDK.cs 代码

[csharp] view plaincopyprint?

```
1.  using UnityEngine;
2.  using System.Runtime.InteropServices;
3.
4.  public class SDK
5.  {
6.
7.      //导出按钮以后将在 xcode 项目中生成这个按钮的注册，
8.      //这样就可以在 xocde 代码中实现这个按钮点击后的事件。
9.      [DllImport("__Internal")]
10.     private static extern void _PressButton0 ();
11.
12.     public static void ActivateButton0 ()
13.     {
14.
15.       if (Application.platform != RuntimePlatform.OSXEditor)
16.       {
17.           //点击按钮后调用 xcode 中的 _PressButton0 ()方法，
18.           //方法中的内容须要我们自己来添加
19.           _PressButton0 ();
20.       }
21.     }
22.
23.     //和上面一样
24.     [DllImport("__Internal")]
25.     private static extern void _PressButton1 ();
26.
27.     public static void ActivateButton1 ()
28.     {
29.       if (Application.platform != RuntimePlatform.OSXEditor)
30.       {
31.           _PressButton1 ();
32.       }
33.     }
34.
35. }
```



这样子 Unity3D 部分已经完成，将 Untiy3D 项目导出成 Xcode 项目，我们用 Xcode 打开它。添加 Unit3D 中 GUI 按钮点击后的响应事件。创建一个类命名为 MyView.h 、MyView.m ,用它来接收 Unity3D 回馈回来的消息，_PressButton0 与 _PressButton1 这两个方法在 Unity3D 中已经注册过，所以在这个类中我们须要对它进行 Xcode 中的实现。

## MyView.m

[csharp] view plaincopyprint?

```
1.  #import "MyView.h"
2.
3.
4.  @implementation MyView
5.
6.  //接收 Unity3D 传递过来的信息
7.
8.  void _PressButton0()
9.  {
10.     UIAlertView *alert = [[UIAlertView alloc] init];
11.     [alert setTitle:@"雨松 MOMO 程序世界"];
12.     [alert setMessage:@"点击了第一个按钮"];
13.     [alert addButtonWithTitle:@"确定"];
14.     [alert  show];
15.     [alert release];
16. }
17.
18. void _PressButton1()
19. {
20.
21.     UIAlertView *alert = [[UIAlertView alloc] init];
22.     [alert setTitle:@"雨松 MOMO 程序世界"];
23.     [alert setMessage:@"点击了第二个按钮"];
24.     [alert addButtonWithTitle:@"确定"];
25.     [alert  show];
26.     [alert release];
27. }
28. @end
```

```
#import "MyView.h"


@implementation MyView
```

OK 大功告成，连上真机运行我们的项目，我们在 iPhone 中点击了 Unity3D 中 GUI 这两个按钮后，通过消息的回馈顺利的弹出 IOS 高级界面 的对话框，哇咔咔～


最后欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，冬天就要来啦大家注意添加衣服，注意身体健康噢。哇咔咔～～～

附上 Unity3D 工程的下载地址，Xcode 项目我就不上传了，须要的自己导出。不早了，大家晚安，哇咔咔～～

下载地址：http://download.csdn.net/detail/xys289187120/3752608

镜头任意的旋转与缩放，还不错噢，大家快点来学习 Unity3D 游戏开发吧，哇咔咔～～

最后欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，哇咔咔～～～ 附上工程的下载地址，文件名称是 zoom.unitypackage，下载完毕的盆友们双击就可以自动在 Unity3D 下打开它啦。然后导出成 iPhone 项目就可以运行了。

今天回家装个 windows 7 最近对 wp7 的游戏开发有点兴趣哦～ 哦也～回家钻研一下～哇咔咔

下载地址：http://download.csdn.net/detail/xys289187120/3725915

# 第十一章 Unity3D 游戏引擎之 IOS 自定义游戏摇杆与飞机平滑的移动

移动开发游戏中使用到的触摸游戏摇杆在 iPhone 上是非常普遍的，毕竟是全触摸屏手机，今天 MOMO 通过一个小例子和大家讨论 Unity3D 中如何自定义一个漂亮的全触摸游戏摇杆。

值得高兴的是，Unity3D 游戏引擎的标准资源中已经帮助我们封装了一个游戏摇杆脚本，所以实现部分的代码可以完全借助它的，具体调用需要我们自己来。

Joystick.js 是官方提供的脚本，具体代码如下，有兴趣的朋友可以仔细研究研究，MOMO 就不多说啦。哇咔咔～

[javascript] view plaincopyprint?

```
1.  ////////////////////////////////////////////////////////////////
2.  // Joystick.js
3.  // Penelope iPhone Tutorial
4.  //
5.  // Joystick creates a movable joystick (via GUITexture) that
6.  // handles touch input, taps, and phases. Dead zones can control
7.  // where the joystick input gets picked up and can be normalized.
8.  //
9.  // Optionally, you can enable the touchPad property from the editor
10. // to treat this Joystick as a TouchPad. A TouchPad allows the finger
11. // to touch down at any point and it tracks the movement relatively
12. // without moving the graphic
13. ////////////////////////////////////////////////////////////////
14.
15. @script RequireComponent( GUITexture )
16.
17. // A simple class for bounding how far the GUITexture will move
18. class Boundary
19. {
20.     var min : Vector2 = Vector2.zero;
21.     var max : Vector2 = Vector2.zero;
22. }
23.
24. static private var joysticks : Joystick[];                // A static collection of all joysticks
25. static private var enumeratedJoysticks : boolean = false;
26. static private var tapTimeDelta : float = 0.3;            // Time allowed between taps
27.
28. var touchPad : boolean;                                   // Is this a TouchPad?
29. var touchZone : Rect;
30. var deadZone : Vector2 = Vector2.zero;                    // Control when position is output
31. var normalize : boolean = false;                         // Normalize output after the dead-zone?
32. var position : Vector2;                                  // [-1, 1] in x,y
33. var tapCount : int;                                      // Current tap count
34.
35. private var lastFingerId = -1;                           // Finger last used for this joystick
36. private var tapTimeWindow : float;                       // How much time there is left for a tap to occur
37. private var fingerDownPos : Vector2;
38. private var fingerDownTime : float;
39. private var firstDeltaTime : float = 0.5;
40.
41. private var gui : GUITexture;                            // Joystick graphic
42. private var defaultRect : Rect;                          // Default position / extents of the joystick graphic
43. private var guiBoundary : Boundary = Boundary();         // Boundary for joystick graphic
44. private var guiTouchOffset : Vector2;                    // Offset to apply to touch input
45. private var guiCenter : Vector2;                         // Center of joystick
46.
47. function Start()
48. {
```

```
49.     // Cache this component at startup instead of looking up every frame
50.     gui = GetComponent( GUITexture );
51.
52.     // Store the default rect for the gui, so we can snap back to it
53.     defaultRect = gui.pixelInset;
54.
55.     defaultRect.x += transform.position.x * Screen.width;// + gui.pixelInset.x; // -  Screen.width * 0.5;
56.     defaultRect.y += transform.position.y * Screen.height;// - Screen.height * 0.5;
57.
58.     transform.position.x = 0.0;
59.     transform.position.y = 0.0;
60.
61.     if ( touchPad )
62.     {
63.         // If a texture has been assigned, then use the rect ferom the gui as our touchZone
64.         if ( gui.texture )
65.             touchZone = defaultRect;
66.     }
67.     else
68.     {
69.         // This is an offset for touch input to match with the top left
70.         // corner of the GUI
71.         guiTouchOffset.x = defaultRect.width * 0.5;
72.         guiTouchOffset.y = defaultRect.height * 0.5;
73.
74.         // Cache the center of the GUI, since it doesn't change
75.         guiCenter.x = defaultRect.x + guiTouchOffset.x;
76.         guiCenter.y = defaultRect.y + guiTouchOffset.y;
77.
78.         // Let's build the GUI boundary, so we can clamp joystick movement
79.         guiBoundary.min.x = defaultRect.x - guiTouchOffset.x;
80.         guiBoundary.max.x = defaultRect.x + guiTouchOffset.x;
81.         guiBoundary.min.y = defaultRect.y - guiTouchOffset.y;
82.         guiBoundary.max.y = defaultRect.y + guiTouchOffset.y;
83.     }
84. }
85.
86. function Disable()
87. {
88.     gameObject.active = false;
89.     enumeratedJoysticks = false;
90. }
91.
92. function ResetJoystick()
93. {
94.     // Release the finger control and set the joystick back to the default position
95.     gui.pixelInset = defaultRect;
96.     lastFingerId = -1;
97.     position = Vector2.zero;
98.     fingerDownPosition = Vector2.zero;
99.
100.    if ( touchPad )
101.        gui.color.a = 0.025;
```

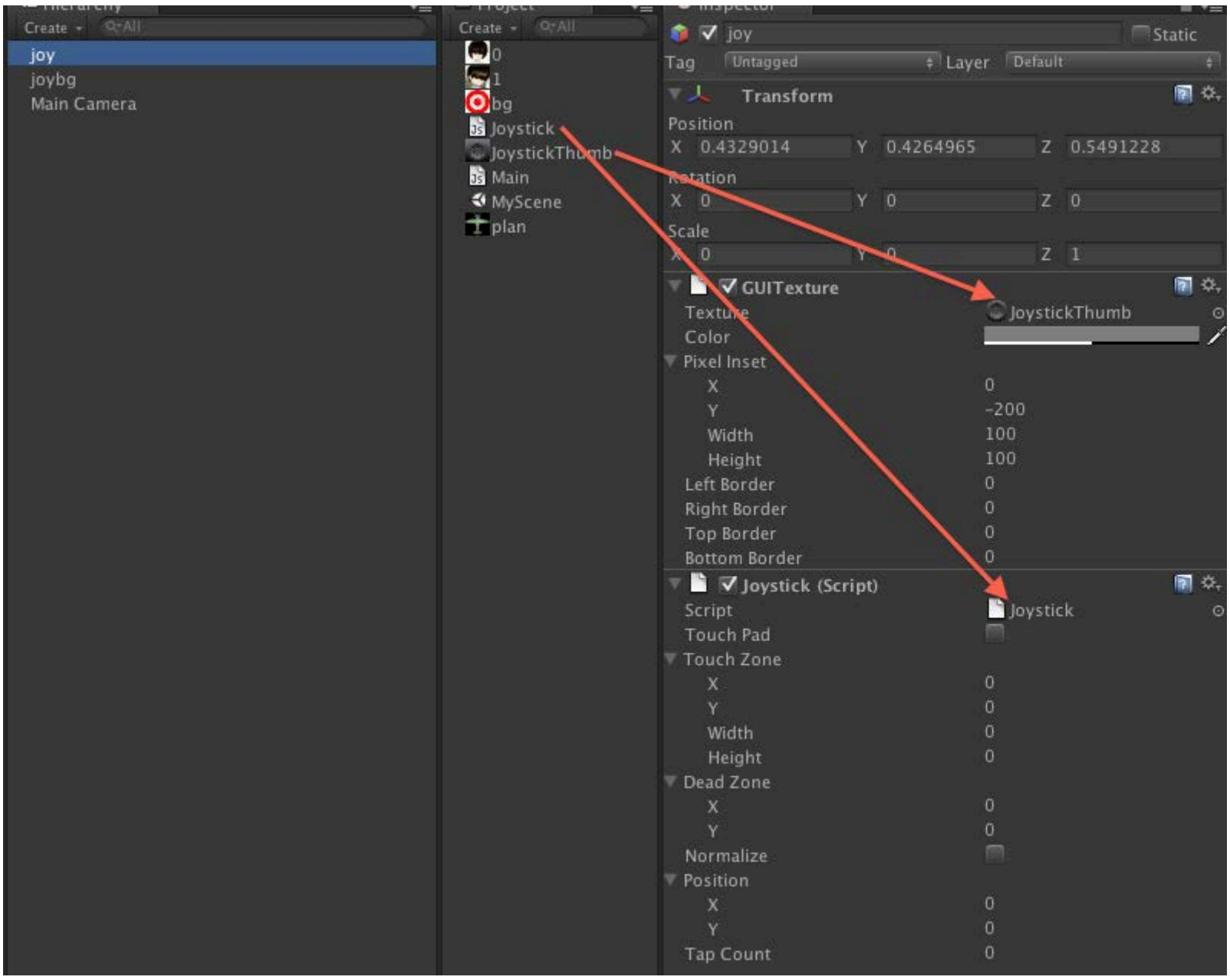
```
102.}
103.
104.function IsFingerDown() : boolean
105.{
106.    return (lastFingerId != -1);
107.}
108.
109.function LatchedFinger( fingerId : int )
110.{
111.    // If another joystick has latched this finger, then we must release it
112.    if ( lastFingerId == fingerId )
113.        ResetJoystick();
114.}
115.
116.function Update()
117.{
118.    if ( !enumeratedJoysticks )
119.    {
120.        // Collect all joysticks in the game, so we can relay finger latching messages
121.        joysticks = FindObjectsOfType( Joystick );
122.        enumeratedJoysticks = true;
123.    }
124.
125.    var count = Input.touchCount;
126.
127.    // Adjust the tap time window while it still available
128.    if ( tapTimeWindow > 0 )
129.        tapTimeWindow -= Time.deltaTime;
130.    else
131.        tapCount = 0;
132.
133.    if ( count == 0 )
134.        ResetJoystick();
135.    else
136.    {
137.        for(var i : int = 0;i < count; i++)
138.        {
139.            var touch : Touch = Input.GetTouch(i);
140.            var guiTouchPos : Vector2 = touch.position - guiTouchOffset;
141.
142.            var shouldLatchFinger = false;
143.            if ( touchPad )
144.            {
145.                if ( touchZone.Contains( touch.position ) )
146.                    shouldLatchFinger = true;
147.            }
148.            else if ( gui.HitTest( touch.position ) )
149.            {
150.                shouldLatchFinger = true;
151.            }
152.
153.            // Latch the finger if this is a new touch
154.            if ( shouldLatchFinger && ( lastFingerId == -1 || lastFingerId != touch.fingerId ) )
```

```
155.        {
156.
157.            if ( touchPad )
158.            {
159.                gui.color.a = 0.15;
160.
161.                lastFingerId = touch.fingerId;
162.                fingerDownPos = touch.position;
163.                fingerDownTime = Time.time;
164.            }
165.
166.            lastFingerId = touch.fingerId;
167.
168.            // Accumulate taps if it is within the time window
169.            if ( tapTimeWindow > 0 )
170.                tapCount++;
171.            else
172.            {
173.                tapCount = 1;
174.                tapTimeWindow = tapTimeDelta;
175.            }
176.
177.            // Tell other joysticks we've latched this finger
178.            for ( var j : Joystick in joysticks )
179.            {
180.                if ( j != this )
181.                    j.LatchedFinger( touch.fingerId );
182.            }
183.        }
184.
185.        if ( lastFingerId == touch.fingerId )
186.        {
187.            // Override the tap count with what the iPhone SDK reports if it is greater
188.            // This is a workaround, since the iPhone SDK does not currently track taps
189.            // for multiple touches
190.            if ( touch.tapCount > tapCount )
191.                tapCount = touch.tapCount;
192.
193.            if ( touchPad )
194.            {
195.                // For a touchpad, let's just set the position directly based on distance from initial touchdown
196.                position.x = Mathf.Clamp( ( touch.position.x - fingerDownPos.x ) / ( touchZone.width / 2 ), -1, 1 );
197.                position.y = Mathf.Clamp( ( touch.position.y - fingerDownPos.y ) / ( touchZone.height / 2 ), -1, 1 );
198.            }
199.            else
200.            {
201.                // Change the location of the joystick graphic to match where the touch is
202.                gui.pixelInset.x =  Mathf.Clamp( guiTouchPos.x, guiBoundary.min.x, guiBoundary.max.x );
203.                gui.pixelInset.y =  Mathf.Clamp( guiTouchPos.y, guiBoundary.min.y, guiBoundary.max.y );
204.            }
205.
206.            if ( touch.phase == TouchPhase.Ended || touch.phase == TouchPhase.Canceled )
207.                ResetJoystick();
```

```
208.            }
209.        }
210.    }
211.
212.    if ( !touchPad )
213.    {
214.        // Get a value between -1 and 1 based on the joystick graphic location
215.        position.x = ( gui.pixelInset.x + guiTouchOffset.x - guiCenter.x ) / guiTouchOffset.x;
216.        position.y = ( gui.pixelInset.y + guiTouchOffset.y - guiCenter.y ) / guiTouchOffset.y;
217.    }
218.
219.    // Adjust for dead zone
220.    var absoluteX = Mathf.Abs( position.x );
221.    var absoluteY = Mathf.Abs( position.y );
222.
223.    if ( absoluteX < deadZone.x )
224.    {
225.        // Report the joystick as being at the center if it is within the dead zone
226.        position.x = 0;
227.    }
228.    else if ( normalize )
229.    {
230.        // Rescale the output after taking the dead zone into account
231.        position.x = Mathf.Sign( position.x ) * ( absoluteX - deadZone.x ) / ( 1 - deadZone.x );
232.    }
233.
234.    if ( absoluteY < deadZone.y )
235.    {
236.        // Report the joystick as being at the center if it is within the dead zone
237.        position.y = 0;
238.    }
239.    else if ( normalize )
240.    {
241.        // Rescale the output after taking the dead zone into account
242.        position.y = Mathf.Sign( position.y ) * ( absoluteY - deadZone.y ) / ( 1 - deadZone.y );
243.    }
244.}
```

```
////////////////////////////////////
// Joystick.js
// Penelope iPhone Tutorial
//
// Joystick creates a movable jo
```

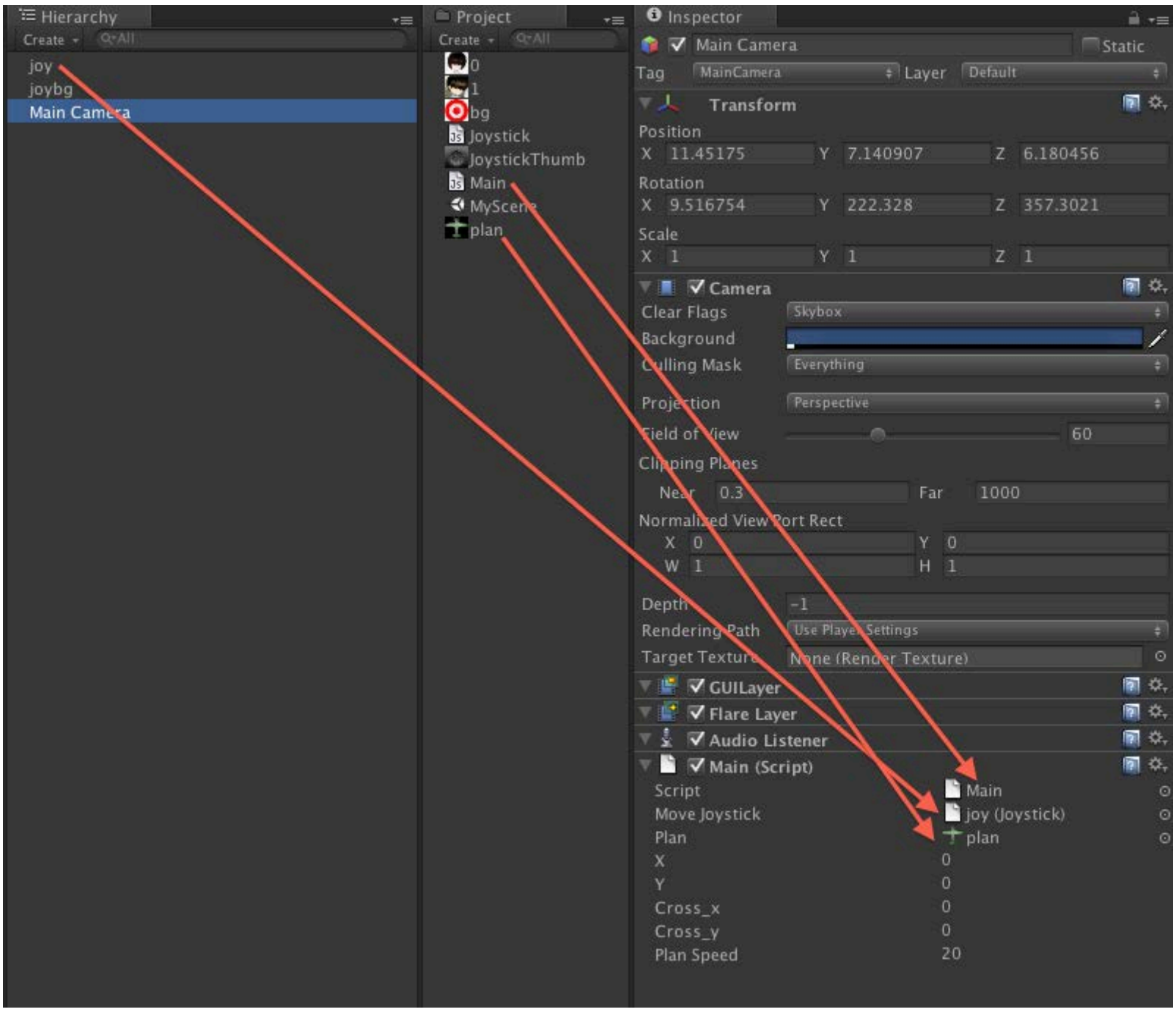
单击 Create 创建一个 GUI Texture，命名为 Joy ，它用来显示游戏摇杆，如下图所示将摇杆的图片资源，与摇杆的脚本连线赋值给 Joy. Pixel Inset 中可以设置摇杆的显示位置与显示宽高。




到这一步 build and run 就可以在 iPhone 上看到这个游戏摇杆，并且可以通过触摸它，360 度平滑过度。

在屏幕中绘制一个飞机，通过游戏摇杆去控制飞机的移动。

创建一个脚本，命名为 Main.js 如下图所示 将 Main.js 、joy、plan 分别 绑定在 Main Camera 上。




moveJoystick.position.x;

moveJoystick.position.y;

这两个值是非常重要的两个信息，它们的取值范围是 -1 到 +1 ，表示 用户触摸摇杆的位置， 上 下 左 右 的信息。

[javascript] view plaincopyprint?

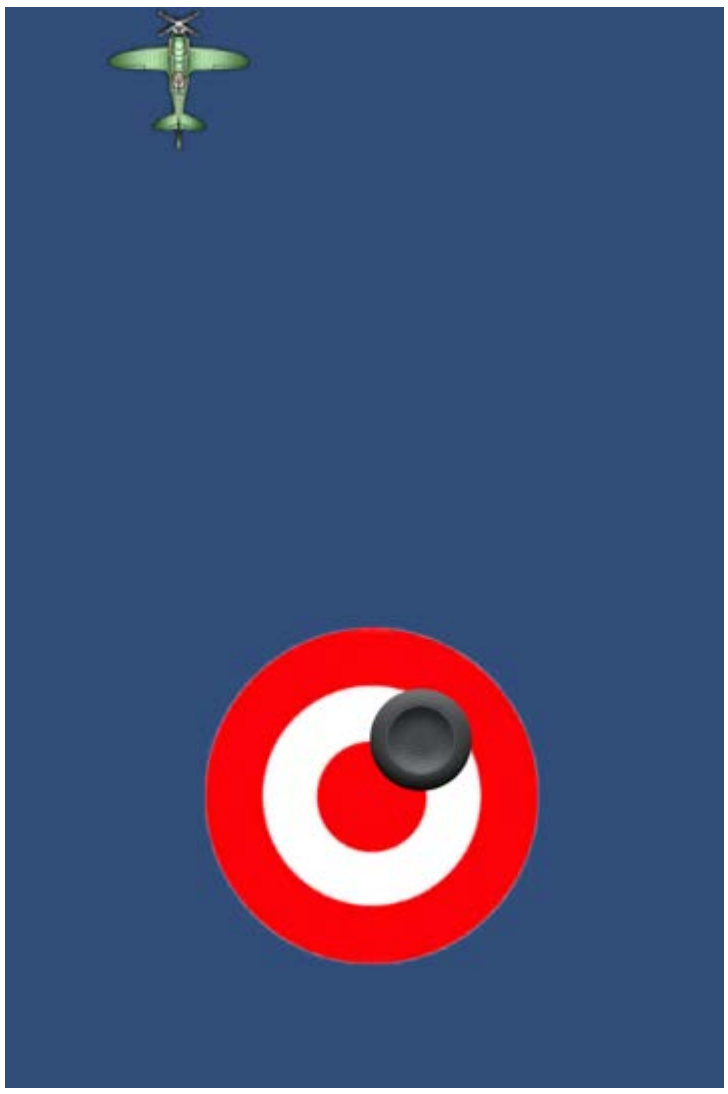
```
1.  //游戏摇杆对象
2.  var moveJoystick : Joystick;
3.
4.  //飞机的贴图
5.  var plan : Texture;
6.
7.  //飞机在屏幕中的坐标
8.  var x = 0;
9.  var y = 0;
10.
11. //避免飞机飞出屏幕，分别是 X、Y 最大坐标，最小坐标是 0、0
12. var cross_x = 0;
13. var cross_y = 0;
14.
```

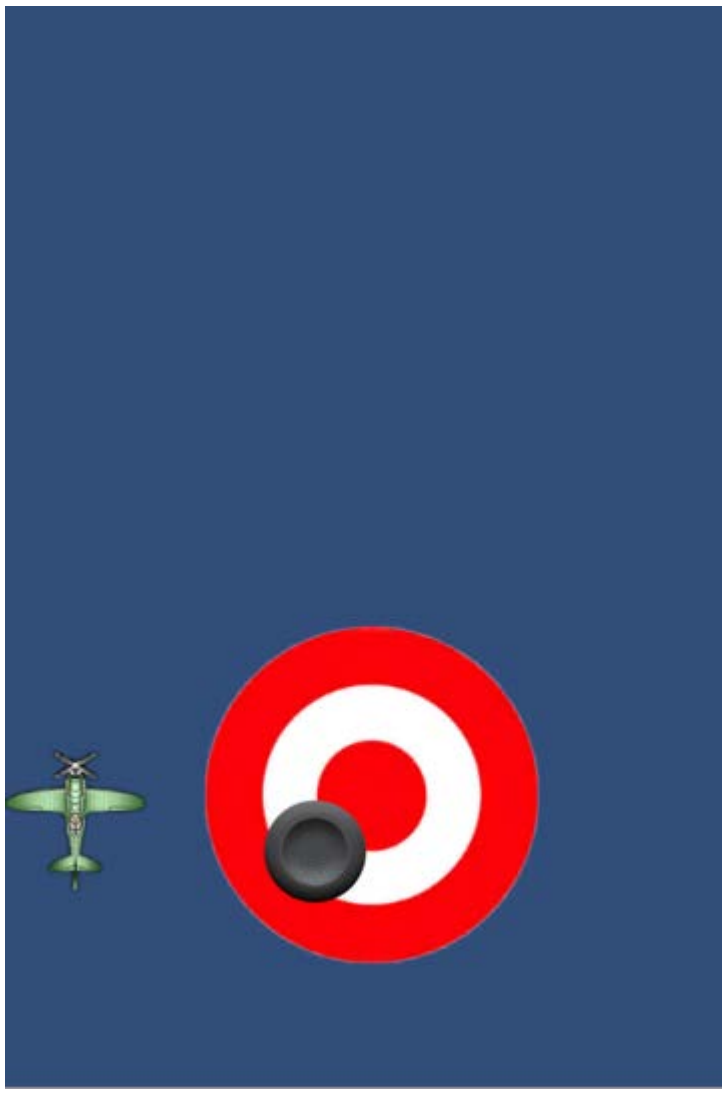
```
15. //飞机移动的速度
16. var planSpeed = 20;
17.
18. function Start() {
19.     //初始化赋值
20.     x = 100;
21.     y = 100;
22.     cross_x = Screen.width -  plan.width;
23.     cross_y = Screen.height -  plan.height;
24.
25. }
26.
27. function Update () {
28.     //得到游戏摇杆的反馈信息，得到的值是 -1 到 +1 之间
29.
30.     var touchKey_x =  moveJoystick.position.x;
31.     var touchKey_y =  moveJoystick.position.y;
32.
33.     //摇杆向左
34.     if(touchKey_x == -1){
35.         x -= planSpeed;
36.
37.     }
38.     //摇杆向右
39.     else if(touchKey_x == 1){
40.         x += planSpeed;
41.
42.     }
43.     //摇杆向上
44.     if(touchKey_y == -1){
45.         y += planSpeed;
46.
47.     }
48.     //摇杆向下
49.     else if(touchKey_y == 1){
50.         y -= planSpeed;
51.
52.     }
53.
54.     //防止飞机飞出屏幕，出界检测
55.     if(x < 0){
56.         x = 0;
57.     }else if(x > cross_x){
58.         x = cross_x;
59.     }
60.
61.     if(y < 0){
62.         y = 0;
63.     }else if(y > cross_y){
64.         y = cross_y;
65.     }
66. }
67.
```

```
68.
69.
70.
71.  function OnGUI () {
72.
73.
74.    //将飞机绘制在屏幕中
75.    GUI.DrawTexture(Rect(x,y,128,128),plan);
76.
77.  }
```

```
//游戏摇杆对象
var moveJoystick : Joystick;

//飞机的贴图
var plan : Texture;
```

导出 build and run 看看在 iPhone 上的效果，通过触摸游戏摇杆可以控制飞机的移动啦，不错吧，哇咔咔～～





最后欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，最近感冒的盆友越来越多，大家要多多注意身体健康噢。哇咔咔～～～ 附上 Unity3D 工程的下载地址，Xcode 项目我就不上传了，须要的自己导出。

下载地址：http://download.csdn.net/detail/xys289187120/3762265

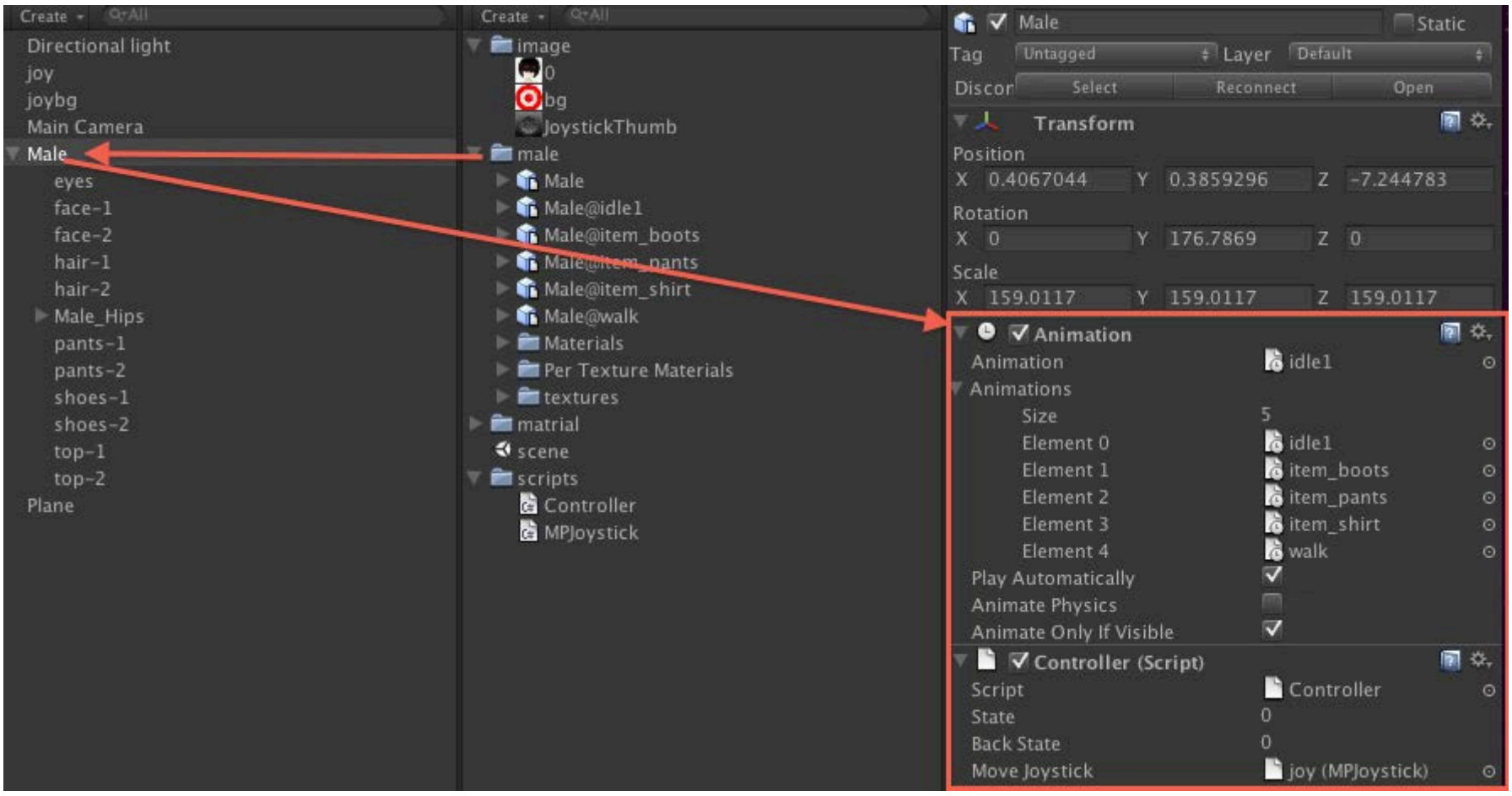
# 第十二章 Unity3D 游戏引擎之 FBX 模型的载入与人物行走动画的播放

3D 世界中自定义模型的使用恐怕是重中之重，因为系统自身提供的模型肯定是无法满足 GD 对游戏的策划，所以为了让游戏更加绚丽，我们须要调用美术制作的精品模型与动画，本章 MOMO 将带领盆友们学习 Unity3D 中模型的载入与动画的播放，哇咔咔～～

由于 MOMO 手头上没有现成的模型，所以我将在 Unity3D 官网中下载官方提供的游戏 DEMO 中的模型来使用。另外官方提供了很多 Unity3D 游戏 DEMO,与详细的文档。可以帮助我们学习 Unity.有兴趣的盆友可以去看看哈。

下载页面：http://unity3d.com/support/resources/

本章博文的目的是利用上一章介绍的游戏摇杆来控制人物模型的移动，与行走动画的播放。




如上图所示 Create 中的文件夹 male 中存放着模型动画与贴图等 ,这个应该是美术提供给我们的。然后将整个 male 用鼠标拖动到左侧 3D 世界中，通过移动，旋转，缩放将人物模型放置在一个理想的位置。右侧红框内设置模型动画的属性。

Animation

idle1　该模型默认动画名称为 idle1

Animations

size　该模型动画的数量

Element 该模型的动画名称

Play Automatically 是否自动播放

Animation Physics 是否设置该模型物理碰撞

Animation Only if Visable 是否设置该模型仅自己显示

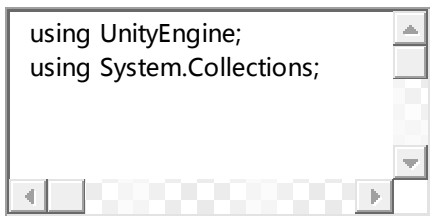
给该模型绑定一个脚本 Controller.cs 用来接收摇杆返回的信息更新模型动画。

Controller.cs

[csharp] view plaincopyprint?

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.
5.
6.  public class Controller : MonoBehaviour {
7.
8.      //人物的行走方向状态
9.      public const int HERO_UP= 0;
10.     public const int HERO_RIGHT= 1;
11.     public const int HERO_DOWN= 2;
12.     public const int HERO_LEFT= 3;
13.
14.     //人物当前行走方向状态
15.     public int state = 0;
16.
17.     //备份上一次人物当前行走方向状态
18.     //这里暂时没有用到
19.     public int backState = 0;
20.
21.     //游戏摇杆对象
22.     public MPJoystick moveJoystick;
23.
24.     //这个方法只调用一次，在 Start 方法之前调用
25.     public void Awake() {
26.
27.     }
28.
29.     //这个方法只调用一次，在 Awake 方法之后调用
30.     void Start () {
31.         state = HERO_DOWN;
32.     }
33.
34.
35.     void Update () {
36.
37.     //获取摇杆控制的方向数据 上一章有详细介绍
38.     float touchKey_x =  moveJoystick.position.x;
39.     float touchKey_y =  moveJoystick.position.y;
40.
41.
42.
43.     if(touchKey_x == -1){
44.         setHeroState(HERO_LEFT);
45.
46.     }else if(touchKey_x == 1){
47.         setHeroState(HERO_RIGHT);
48.
49.     }
50.
51.     if(touchKey_y == -1){
52.         setHeroState(HERO_DOWN);
53.
```

```
54.     }else if(touchKey_y == 1){
55.         setHeroState(HERO_UP);
56.     }
57.
58.     if(touchKey_x == 0 && touchKey_y ==0){
59.         //松开摇杆后播放默认动画，
60.         //不穿参数为播放默认动画。
61.         animation.Play();
62.     }
63.
64.
65.     }
66.
67.     public void setHeroState(int newState)
68.     {
69.
70.         //根据当前人物方向 与上一次备份方向计算出模型旋转的角度
71.         int rotateValue = (newState - state) * 90;
72.         Vector3 transformValue = new Vector3();
73.
74.         //播放行走动画
75.         animation.Play("walk");
76.
77.         //模型移动的位移的数值
78.         switch(newState){
79.             case HERO_UP:
80.                 transformValue = Vector3.forward * Time.deltaTime;
81.             break;
82.             case HERO_DOWN:
83.                 transformValue = -Vector3.forward * Time.deltaTime;
84.             break;
85.             case HERO_LEFT:
86.                 transformValue = Vector3.left * Time.deltaTime;
87.
88.             break;
89.             case HERO_RIGHT:
90.                 transformValue = -Vector3.left * Time.deltaTime;
91.                 break;
92.         }
93.
94.
95.         //模型旋转
96.         transform.Rotate(Vector3.up, rotateValue);
97.
98.         //模型移动
99.         transform.Translate(transformValue, Space.World);
100.
101.        backState = state;
102.        state = newState;
103.
104.    }
105.
106.}
```




上一章介绍了 javaScript 脚本使用游戏摇杆的方法，本章 MOMO 告诉大家使用 C＃脚本来使用游戏摇杆，上面我用 Controller.cs C＃脚本来接收系统提供的 Joystick.js 是肯定无法使用的，须要修改成.cs 文件，我在国外的一个网站上看到了一个老外帮我们已经修改了，那么我将他修改后的代码贴出来方便大家学习，有兴趣的朋友可以研究研究。哇咔咔～

## MPJoystick.cs

[csharp] view plaincopyprint?

```
1.  using UnityEngine;
2.
3.  /**
4.
5.   * File: MPJoystick.cs
6.
7.   * Author: Chris Danielson of (monkeyprism.com)
8.
9.   *
10.
11. // USED TO BE: Joystick.js taken from Penelope iPhone Tutorial
12.
13. //
14.
15. // Joystick creates a movable joystick (via GUITexture) that
16.
17. // handles touch input, taps, and phases. Dead zones can control
18.
19. // where the joystick input gets picked up and can be normalized.
20.
21. //
22.
23. // Optionally, you can enable the touchPad property from the editor
24.
25. // to treat this Joystick as a TouchPad. A TouchPad allows the finger
26.
27. // to touch down at any point and it tracks the movement relatively
28.
29. // without moving the graphic
30.
31. */
32. [RequireComponent(typeof(GUITexture))]
33.
34. public class MPJoystick : MonoBehaviour
35.
```

```
36.  {
37.
38.  class Boundary {
39.
40.  public Vector2 min = Vector2.zero;
41.
42.  public Vector2 max = Vector2.zero;
43.
44.  }
45.  private static MPJoystick[] joysticks;   // A static collection of all joysticks
46.
47.  private static bool enumeratedJoysticks = false;
48.
49.  private static float tapTimeDelta = 0.3f;    // Time allowed between taps
50.  public bool touchPad;
51.
52.  public Vector2 position = Vector2.zero;
53.
54.  public Rect touchZone;
55.
56.  public Vector2 deadZone = Vector2.zero;  // Control when position is output
57.
58.  public bool normalize = false; // Normalize output after the dead-zone?
59.
60.  public int tapCount;
61.
62.  private int lastFingerId = -1;   // Finger last used for this joystick
63.
64.  private float tapTimeWindow;     // How much time there is left for a tap to occur
65.
66.  private Vector2 fingerDownPos;
67.
68.  //private float fingerDownTime;
69.
70.  //private float firstDeltaTime = 0.5f;
71.  private GUITexture gui;
72.
73.  private Rect defaultRect;    // Default position / extents of the joystick graphic
74.
75.  private Boundary guiBoundary = new Boundary();   // Boundary for joystick graphic
76.
77.  private Vector2 guiTouchOffset;  // Offset to apply to touch input
78.
79.  private Vector2 guiCenter;   // Center of joystick
80.  void Start() {
81.
82.  gui = (GUITexture)GetComponent(typeof(GUITexture));
83.  defaultRect = gui.pixelInset;
84.
85.  defaultRect.x += transform.position.x * Screen.width;// + gui.pixelInset.x; // -  Screen.width * 0.5;
86.
87.          defaultRect.y += transform.position.y * Screen.height;// - Screen.height * 0.5;
88.  transform.position = Vector3.zero;
```

```
89.  if (touchPad) {
90.
91.  // If a texture has been assigned, then use the rect ferom the gui as our touchZone
92.
93.  if ( gui.texture )
94.
95.  touchZone = defaultRect;
96.
97.  } else {
98.
99.  guiTouchOffset.x = defaultRect.width * 0.5f;
100.
101.guiTouchOffset.y = defaultRect.height * 0.5f;
102.// Cache the center of the GUI, since it doesn't change
103.
104.guiCenter.x = defaultRect.x + guiTouchOffset.x;
105.
106.guiCenter.y = defaultRect.y + guiTouchOffset.y;
107.// Let's build the GUI boundary, so we can clamp joystick movement
108.
109.guiBoundary.min.x = defaultRect.x - guiTouchOffset.x;
110.
111.guiBoundary.max.x = defaultRect.x + guiTouchOffset.x;
112.
113.guiBoundary.min.y = defaultRect.y - guiTouchOffset.y;
114.
115.guiBoundary.max.y = defaultRect.y + guiTouchOffset.y;
116.
117.}
118.
119.}
120.public Vector2 getGUICenter() {
121.
122.return guiCenter;
123.
124.}
125.void Disable() {
126.
127.gameObject.active = false;
128.
129.//enumeratedJoysticks = false;
130.
131.}
132.private void ResetJoystick() {
133.
134.gui.pixelInset = defaultRect;
135.
136.lastFingerId = -1;
137.
138.position = Vector2.zero;
139.
140.fingerDownPos = Vector2.zero;
141.
```

```
142.}
143.private bool IsFingerDown() {
144.
145.return (lastFingerId != -1);
146.
147.}
148.public void LatchedFinger(int fingerId) {
149.
150.// If another joystick has latched this finger, then we must release it
151.
152.if ( lastFingerId == fingerId )
153.
154.ResetJoystick();
155.
156.}
157.void Update() {
158.
159.if (!enumeratedJoysticks) {
160.
161.// Collect all joysticks in the game, so we can relay finger latching messages
162.
163.joysticks = (MPJoystick[])FindObjectsOfType(typeof(MPJoystick));
164.
165.enumeratedJoysticks = true;
166.
167.}
168.int count = Input.touchCount;
169.if ( tapTimeWindow > 0 )
170.
171.tapTimeWindow -= Time.deltaTime;
172.
173.else
174.
175.tapCount = 0;
176.if ( count == 0 )
177.
178.ResetJoystick();
179.
180.else
181.
182.{
183.
184.for(int i = 0; i < count; i++) {
185.
186.Touch touch = Input.GetTouch(i);
187.
188.Vector2 guiTouchPos = touch.position - guiTouchOffset;
189.bool shouldLatchFinger = false;
190.
191.if (touchPad) {
192.
193.if (touchZone.Contains(touch.position))
194.
```

```
195.shouldLatchFinger = true;
196.
197.}
198.
199.else if (gui.HitTest(touch.position)) {
200.
201.shouldLatchFinger = true;
202.
203.}
204.// Latch the finger if this is a new touch
205.
206.if (shouldLatchFinger && (lastFingerId == -1 || lastFingerId != touch.fingerId )) {
207.if (touchPad) {
208.
209.//gui.color.a = 0.15;
210.
211.lastFingerId = touch.fingerId;
212.
213.//fingerDownPos = touch.position;
214.
215.//fingerDownTime = Time.time;
216.
217.}
218.lastFingerId = touch.fingerId;
219.
220.
221.
222.// Accumulate taps if it is within the time window
223.
224.if ( tapTimeWindow > 0 )
225.
226.tapCount++;
227.
228.else {
229.
230.tapCount = 1;
231.
232.tapTimeWindow = tapTimeDelta;
233.
234.}
235.// Tell other joysticks we've latched this finger
236.
237.//for (  j : Joystick in joysticks )
238.
239.foreach (MPJoystick j in joysticks) {
240.
241.if (j != this)
242.
243.j.LatchedFinger( touch.fingerId );
244.
245.}
246.
247.}
```

```
248.if ( lastFingerId == touch.fingerId ) {
249.
250.// Override the tap count with what the iPhone SDK reports if it is greater
251.
252.// This is a workaround, since the iPhone SDK does not currently track taps
253.
254.// for multiple touches
255.
256.if ( touch.tapCount > tapCount )
257.
258.tapCount = touch.tapCount;
259.if ( touchPad ) {
260.
261.// For a touchpad, let's just set the position directly based on distance from initial touchdown
262.
263.position.x = Mathf.Clamp( ( touch.position.x - fingerDownPos.x ) / ( touchZone.width / 2 ), -1, 1 );
264.
265.position.y = Mathf.Clamp( ( touch.position.y - fingerDownPos.y ) / ( touchZone.height / 2 ), -1, 1 );
266.
267.} else {
268.
269.// Change the location of the joystick graphic to match where the touch is
270.
271.Rect r = gui.pixelInset;
272.
273.r.x =  Mathf.Clamp( guiTouchPos.x, guiBoundary.min.x, guiBoundary.max.x );
274.
275.r.y =  Mathf.Clamp( guiTouchPos.y, guiBoundary.min.y, guiBoundary.max.y );
276.
277.gui.pixelInset = r;
278.
279.}
280.if (touch.phase == TouchPhase.Ended || touch.phase == TouchPhase.Canceled)
281.
282.ResetJoystick();
283.
284.}
285.
286.}
287.
288.}
289.if (!touchPad) {
290.
291.// Get a value between -1 and 1 based on the joystick graphic location
292.
293.position.x = ( gui.pixelInset.x + guiTouchOffset.x - guiCenter.x ) / guiTouchOffset.x;
294.
295.position.y = ( gui.pixelInset.y + guiTouchOffset.y - guiCenter.y ) / guiTouchOffset.y;
296.
297.}
298.// Adjust for dead zone
299.
300.var absoluteX = Mathf.Abs( position.x );
```

```
301.
302.var absoluteY = Mathf.Abs( position.y );
303.
304.
305.
306.if (absoluteX < deadZone.x) {
307.
308.// Report the joystick as being at the center if it is within the dead zone
309.
310.position.x = 0;
311.
312.}
313.
314.else if (normalize) {
315.
316.// Rescale the output after taking the dead zone into account
317.
318.position.x = Mathf.Sign( position.x ) * ( absoluteX - deadZone.x ) / ( 1 - deadZone.x );
319.
320.}
321.if (absoluteY < deadZone.y) {
322.
323.// Report the joystick as being at the center if it is within the dead zone
324.
325.position.y = 0;
326.
327.}
328.
329.else if (normalize) {
330.
331.// Rescale the output after taking the dead zone into account
332.
333.position.y = Mathf.Sign( position.y ) * ( absoluteY - deadZone.y ) / ( 1 - deadZone.y );
334.
335.}
336.}
337.}
```

```
using UnityEngine;

/**

 * File: MPJoystick.cs
```

导出 build and run 看看在 iPhone 上的效果，通过触摸游戏摇杆可以控制人物的上，下，左，右 ，左上，右上，左下，右下 8 个方向的移动啦，不错吧，哇咔咔～～

最后欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，本来昨天就想发表这篇文章，结果晚上去打高尔夫球连挥 N 杆，打的回家后浑身酸痛，回家就睡觉啦～希望大家在学习的同时别忘了多运动。哇咔咔～～～ 附上 Unity3D 工程的下载地址，Xcode 项目我就不上传了，须要的自己导出。

下载地址：http://download.csdn.net/detail/xys289187120/3784191

# 第十三章 Unity3D 游戏引擎之平面小球重力感应详解

手机重力感应应该对大多数开发者并不陌生，在新一代智能手机 Android IOS WP7 很多游戏都是使用手机自带重力感应功能制作的，强大的 Unity3D 游戏引擎当然对这个也是完美支持的，今天由 MOMO 带大家学习 3D 世界中的手机重力感应。本章我们的目标是实现一个小球在屏幕中通过摇晃手机重力加速度让小球在屏幕中移动。以前的 Android 系列开发文章中貌似也写过，其实原理都是一样一样一样的，废话不多说了。哇咔咔～～

先看一看 Unity3D 在 iPhone 上的重力分布图。如下图所示我们可以清晰的看出 X Y Z 三个方向的重力分量。Unity3D 中重量的取值范围是 -1.0 到 +1.0.

X 轴：home 按键在下手机面朝天向右旋转 90 度重力分量为+1.0 向左旋转 90 度重力分量为-1.0

Y 轴：home 按键在上手机背朝自己重力分量为+1.0 home 按键在下手机面朝自己重力分量为-1.0

Z 轴：手机面朝地面重力分量为+1.0 手机面朝天空重力分量为-1.0

OK！ 有了这三组重要的数值我们就可以控制手机重力感应啦，紧接着我们看看小球重力感应的这个游戏小例子。


打开 Unity3D 我们将给摄像机绑定一个脚本，用来响应用户控制手机来重力感应游戏小球的移动。

在 Input 这个重要的类中，Unity3D 帮我们封装了重力加速的方法。

Input.acceleration.x; 重力感应 X 轴的重力分量

Input.acceleration.y; 重力感应 Y 轴的重力分量

Input.acceleration.z; 重力感应 Z 轴的重力分量

new.js

[javascript] view plaincopyprint?

```
1.  //小球的贴图
2.  var round : Texture2D;
3.
4.  //小球在屏幕中显示的 X Y 坐标
5.  var x = 0;
6.  var y = 0;
7.
8.  //小球屏幕显示的最大 X Y 范围
9.  var cross_x = 0;
10. var cross_y = 0;
11.
12.
13. function Start(){
14.     //初始化赋值
15.     cross_x = Screen.width -  round.width;
16.     cross_y = Screen.height -  round.height;
17. }
18.
19. function OnGUI () {
20.
21.     //整体显示 x y z 重力感应的重力分量
22.     GUI.Label(Rect(0,0,480,100),"position is " + Input.acceleration);
23.
24.     //绘制小球
25.     GUI.DrawTexture(Rect(x,y,256,256),round);
26. }
27.
28. function Update(){
29.
30.     //根据重力分量修改小球的位置这里乘以 30 的意思是让小球移动的快一些
31.     x += Input.acceleration.x * 30;
32.     y += -Input.acceleration.y * 30;
33.
34.
35.     //避免小球超出屏幕
36.     if(x < 0){
37.         x = 0;
38.     }else if(x > cross_x){
39.         x = cross_x;
40.     }
41.
42.     if(y < 0){
43.         y = 0;
44.     }else if(y > cross_y){
45.         y = cross_y;
46.     }
47. }
```

```
//小球的贴图
var round : Texture2D;

//小球在屏幕中显示的X Y坐标
var x = 0;
```

重力感应的图片不太好截取，我们看看下面的示意图，小球可以根据我的手机的重力而移动。屏幕左上方正常打印手机当前 X Y Z 三个方向的重力分量。


说到这里，可能会有盆友问我如何为 3D 的物体添加重力感应？其实方法是一样的，因为都是一样的道理，比如我可以给一个模型绑定一个这样的脚本，通过 Input.acceleration 去拿到当前手机的重力感应分量，然后根据这个分量去计算当前模型的位置。细心的盆友你们可以试一试，其实很简单的哇咔咔～～

最后欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，总的来说这一章还是比较简单的，哇咔咔～～～ 附上 Unity3D 工程的下载地址，Xcode 项目我就不上传了，须要的自己导出。

下载地址：http://download.csdn.net/detail/xys289187120/3794307

# 第十四章 Unity3D 游戏引擎之游戏场景的切换与持久化简单数据的储存

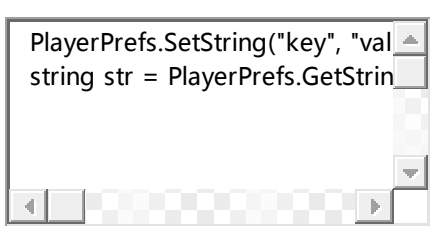
持久化简单的数据储存在 Unity3D 中提供了一个简单有效的方法，如果之前的你做过 Android 的开发你会发现在 Unity3D 中持久化数据的储存和 Android 非常的想象。那么下面 MOMO 将用一个简单有效的例子向大家介绍 Unity3D 中持久化数据。

首先我们须要熟悉一下 Unity3D 中的 PlayerPrefs 这个类。这个类中一共帮助我们封装了 9 个方法，用来数据的储存与读取。

举一个例子

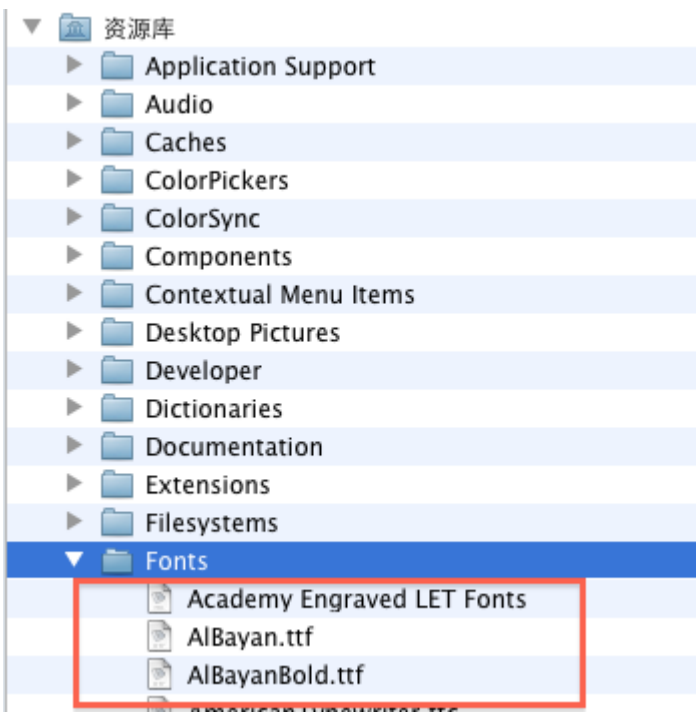
[csharp] view plaincopyprint?

```
1.  PlayerPrefs.SetString("key", "value");
2.  string str = PlayerPrefs.GetString("key", "defaule"));
```




我们发现它是以键值对的形式进行储存与读取，每一个 Key 对应一个 Value，储存过后通过 Key 可以得到之前储存的 Value。这里说一下 GetString()方法中的第二个参数， 它代表默认值。意思是如果通过第一个参数的 Key 没有找到对应的 Value 的话 GetString()方法就会返回我们写的第二个参数的默认值。怎么样？很简单吧~ 感觉和 Android 完全一样哈。

Unity3D 默认的字体的 size 只有 16 ，这就意味了放在 iPhone4 (960 X 640)上 字体会显示的非常小。字体的来源有很多，大家可以在互联网上下载，或者从自己的电脑中拷贝，在 Mac 电脑下字体放在 Finder -> 资源库 -> Fonts



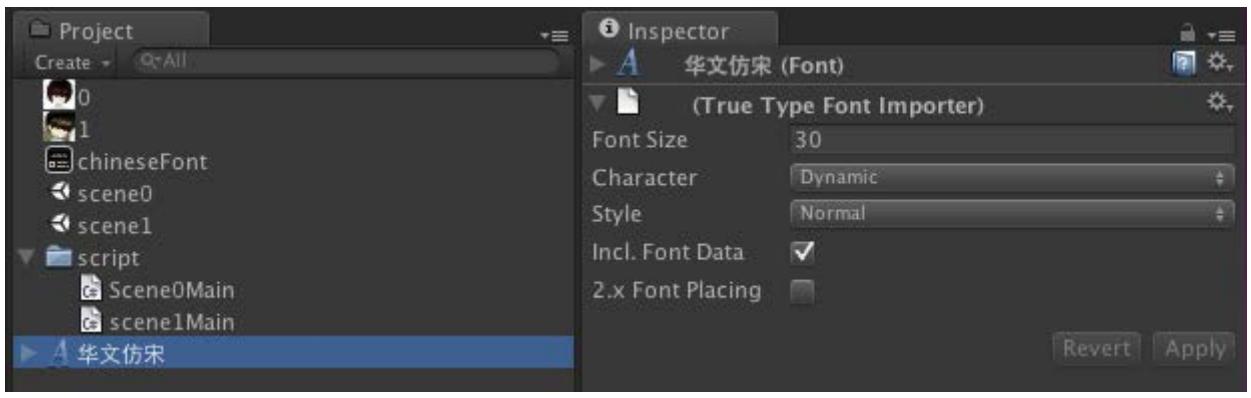

我们可以看见电脑中存在了很多字体，我这里随便选一个，将 华文仿宋.ttf 用鼠标拖动到 Project 中。

选中： 华文仿宋

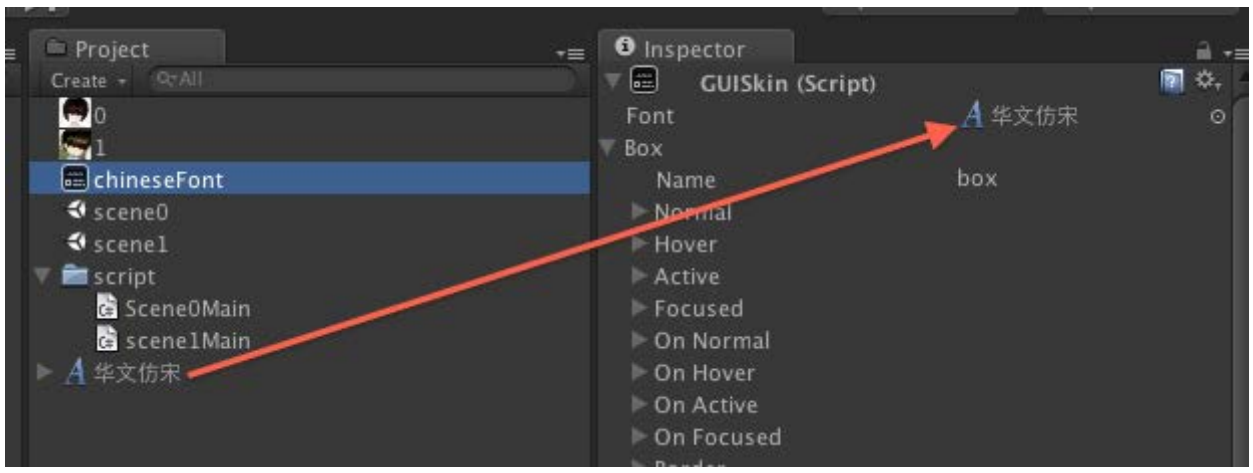
FontSize 30 ：毫无疑问是字体的大小，这里写 30 让字体几乎放大 1 倍。

Character: 设置字体的文字编码 Unicode ASCLL 编码

Style:设置字体的风格，粗体 斜体




点击 Cretae ->GUISkin 创建一个 GUI 的皮肤，将 华文仿宋 拖动到箭头所指向的方向。发现下面存在很多 GUI 皮肤相关控件设置的，可以在这里设置每一个高级控件～大家可以手动的修改一下看看效果哈。



游戏场景在游戏制作中是一个非常重要的部分，因为任何一款游戏都是由若干的场景组成，Unity3D 的游戏场景做的非常贴心。

创建 2 个游戏场景，一个是 scene0 一个是 scene1 ，本章的目标是在第一个游戏场景中保存一些基本游戏数据，然后切换到第二个场景中显示第一个场景中保存的数据，实现场景的切换已经数据的储存。

在 scene0 中创建一个 c# 脚本名称为 Scene0Main.cs 将它绑定在摄像头中。

## Scene0Main.cs

[csharp] view plaincopyprint?

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class Scene0Main : MonoBehaviour {
5.
6.      //储存数据的显示
7.      public string testStr;
8.      public string testInt;
9.      public string testFloat;
10.
11.     //GUI 皮肤 为上面我们添加的皮肤
12.     //在外面用鼠标拖动上为它赋值
13.     public GUISkin fontSkin;
14.     //显示的图片
15.     public Texture Imagetexture;
16.
17.     // Use this for initialization
18.     void Start () {
19.         //读取 key 的值
20.         testStr = PlayerPrefs.GetString("testStr", "default");
21.         testInt = PlayerPrefs.GetInt("testInt", 0).ToString();
22.         testFloat = PlayerPrefs.GetFloat("testFloat", 0).ToString();
23.
24.     }
25.
26.     // Update is called once per frame
27.     void Update () {
28.
29.     }
30.
31.
32.     void OnGUI() {
33.
34.         //将 GUI 的皮肤设置为我们创建的皮肤
35.         GUI.skin = fontSkin;
36.
37.         //贴上图片
```

```
38.     GUI.DrawTexture(new Rect((Screen.width - Imagetexture.width) >>1, 10, 120, 120), Imagetexture);
39.
40.     //添加输入框让用户输入信息，这里面我没有捕获异常，因为用户有可能输入一个不合法的数值
41.     testStr = GUI.TextField (new Rect(10, 200, 200, 50), testStr, 50);
42.     testInt = GUI.TextField (new Rect(10, 250, 200, 50), testInt, 50);
43.     testFloat = GUI.TextField (new Rect(10, 300, 200, 50), testFloat, 50);
44.
45.     //点击按钮保存所有数据
46.     if (GUI.Button(new Rect(220, 200, 150, 100), "commit all"))
47.     {
48.
49.         PlayerPrefs.SetString("testStr", testStr);
50.         PlayerPrefs.SetInt("testInt", int.Parse(testInt));
51.         PlayerPrefs.SetFloat("testFloat", float.Parse(testFloat));
52.         //切换场景到 scene1
53.         Application.LoadLevel("scene1");
54.     }
55.  }
56.
57.
58. }
```

```
using UnityEngine;
using System.Collections;

public class Scene0Main : Mono
```

## Scene1Main.cs

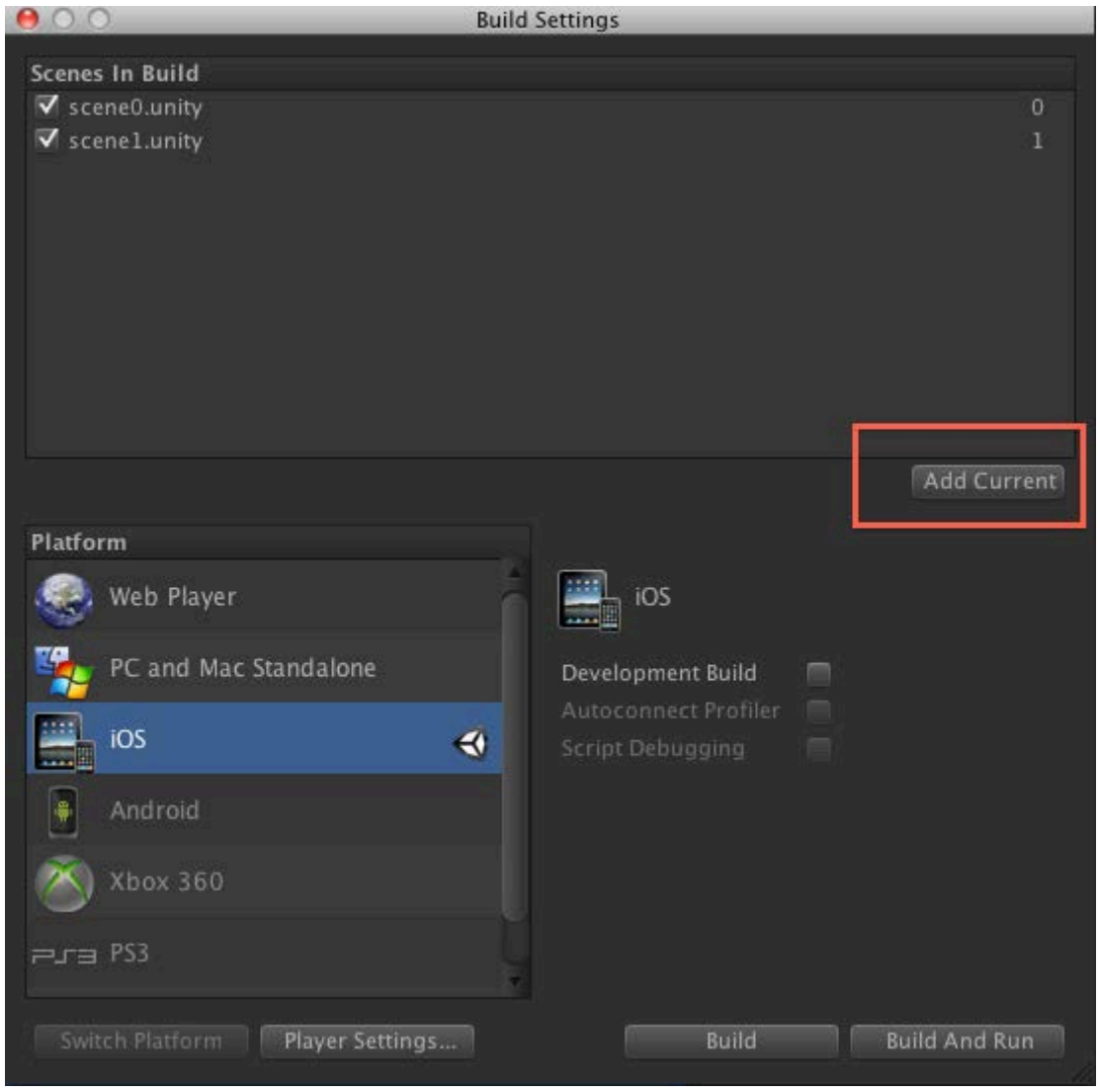
[csharp] view plaincopyprint?

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class scene1Main : MonoBehaviour {
5.
6.      public string testStr;
7.      public string testInt;
8.      public string testFloat;
9.
10.     public GUISkin fontSkin;
11.     public Texture Imagetexture;
12.
13.     // Use this for initialization
14.     void Start () {
15.         testStr = PlayerPrefs.GetString("testStr", "default");
16.         testInt = PlayerPrefs.GetInt("testInt", 0).ToString();
17.         testFloat = PlayerPrefs.GetFloat("testFloat", 0).ToString();
18.
19.     }
20.
21.
```

```
22.     void OnGUI() {
23.         GUI.skin = fontSkin;
24.
25.         GUI.DrawTexture(new Rect((Screen.width - Imagetexture.width) >>1, 10, 120, 120), Imagetexture);
26.
27.         //显示 label
28.         GUI.Label(new Rect(10,150,300,50),"testStr = "+ testStr);
29.         GUI.Label(new Rect(10,200,300,50),"testInt = "+ testInt);
30.         GUI.Label(new Rect(10,250,300,50),"testFloat = "+ testFloat);
31.
32.         if (GUI.Button(new Rect(220, 200, 150, 100), "clean all"))
33.         {
34.             //删除所有键值
35.             PlayerPrefs.DeleteAll();
36.             // 返回场景 0
37.             Application.LoadLevel("scene0");
38.         }
39.
40.         if (GUI.Button(new Rect(220, 320, 150, 100), "only return"))
41.         {
42.             // 返回场景 0
43.             Application.LoadLevel("scene0");
44.         }
45.     }
46. }
```

```
using UnityEngine;
using System.Collections;

public class scene1Main : Monc
```

File -> Build Settings 点击 Add Current 添加场景，这一步很重要，如果不添加的话在代码中切换场景会抛异常，盆友们还得注意一下~




build and run 导出运行项目，如下图所示我分别输入 string int float 三种类型的数据，然后点击 commit all ，将所有数据全部保存下来，游戏场景切换到 scene1 场景中。

切换到 scene1 中可以正常的显示 scene0 中储存的数值，点击 clean all 将清空储存的所有信息后返回场景 scene0，点击 only return 直接返回场景 scene0。


## 另外两个重要的方法

[csharp] view plaincopyprint?

```
1.  //删除 PlayerPrefs 中某一个 key 的值
2.  PlayerPrefs. DeleteKey (“key”);
3.
4.  //判断 PlayerPrefs 中是否存在这个 key
5.  bool b = PlayerPrefs.HasKey(“key”);
```

```
//删除 PlayerPrefs 中某一个key的
PlayerPrefs. DeleteKey (“key”

//判断 PlayerPrefs中是否存在这个
bool b = PlayerPrefs.HasKey(“
```

最后欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，总的来说这一章还是比较简单的，哇咔咔～～～ 附上 Unity3D 工程的下载地址，Xcode 项目我就不上传了，须要的自己导出。

下载地址：http://download.csdn.net/detail/xys289187120/3806498

# 第十五章 Unity3D 游戏引擎之详解游戏开发音频的播放

游戏音频的播放在任何游戏中都占据非常重要的地位，音频的播放还可以分为两种，一种为游戏音乐，另一种为游戏音效。前者适用于较长的音乐，如游戏背景音乐。第二种试用与比较短的游戏音乐，如开枪，打怪 时“砰砰”一瞬间播放的游戏音效。今天 MOMO 将用下面的例子带盆友们去剖析 Unity3D 游戏音乐与音效的播放。

Unity3D 游戏引擎一共支持 4 个音乐格式的文件

.AIFF 适用于较短的音乐文件可用作游戏打斗音效

.WAV 适用于较短的音乐文件可用作游戏打斗音效

.MP3 适用于较长的音乐文件可用作游戏背景音乐

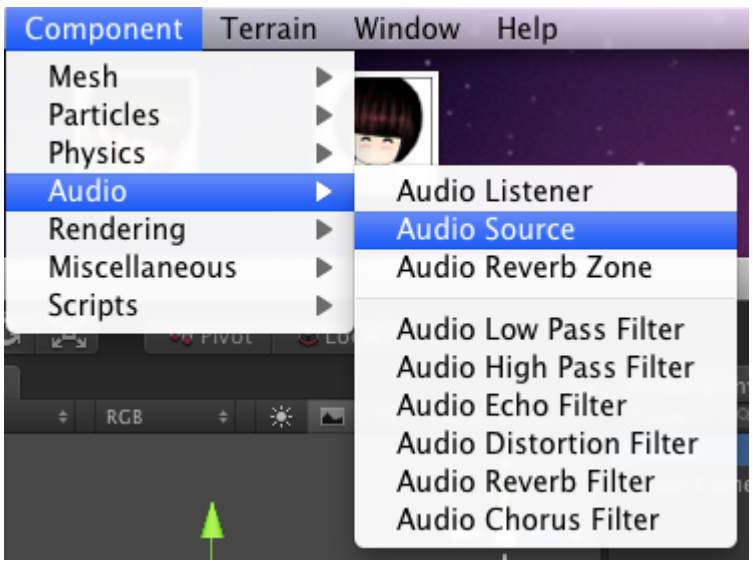
.OGG 适用于较长的音乐文件可用作游戏背景音乐

在场景中创建一个空的游戏对象。

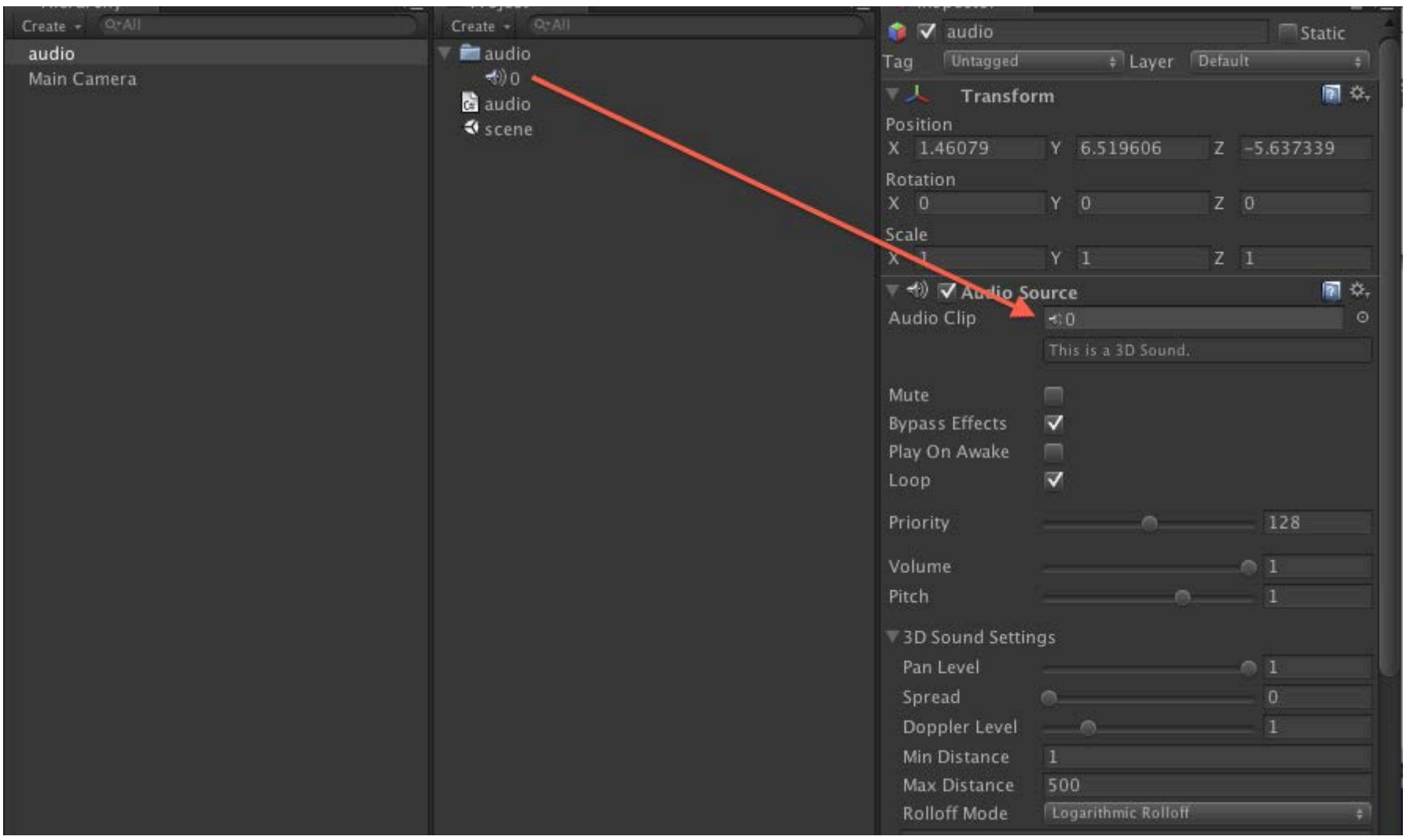
GameObject -> CreateEmpty 创建成功后我命名为 audio。

给 audio 添加一个 AudioSource 属性,这个属性非常的重要，Unity 播放音乐主要就是要靠这个东西。

Component -> Audio - > Audio Source。



找一个音乐文件，我这里使用了一个.mp3 音乐文件，我命名为 0.mp3  如下图所示将它拖动到右侧 Audio Clip 处。




我们发现 Audio Source 有很多设置的属性，那么 MOMO 将一些比较重要的属性诺列出来。

AudioClip ：声音片段，还可以在代码中去动态的截取音乐文件。

Mute： 是否静音

Bypass Effects: 是否打开音频特效

Play On Awake: 开机自动播放

Loop:循环播放

Volume: 声音大小，取值范围 0.0 到 1.0

Pitch:播放速度，取值范围在 -3 到 3 之间 设置 1 为正常播放，小于 1 为减慢播放 大于 1 为加速播放。

创建一个脚本我命名为 audio.cs 用来音乐的播放。本章将实现 3 个按钮 点击实现 播放音乐、 停止音乐、 暂停音乐 ，与一个横向拖动条通过手指的拖动实现动态修改音乐声音。

## audio.cs

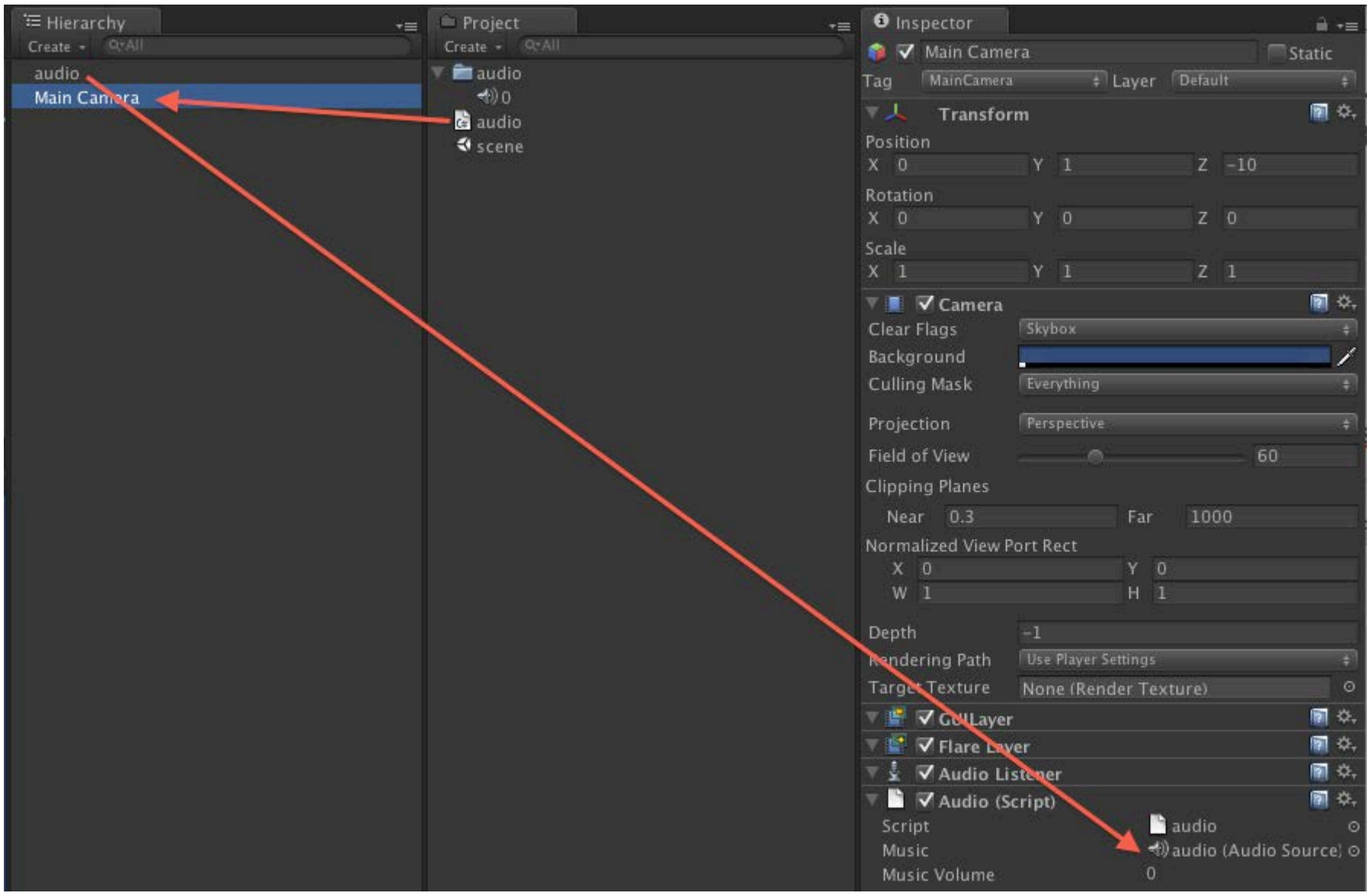
[csharp] view plaincopyprint?

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class audio : MonoBehaviour {
5.
6.      //音乐文件
7.      public AudioSource music;
8.      //音量
9.      public float musicVolume;
10.
11.     void Start() {
12.         //设置默认音量
13.         musicVolume = 0.5F;
14.     }
15.     void OnGUI() {
16.
17.         //播放音乐按钮
18.         if (GUI.Button(new Rect(10, 10, 100, 50), "Play music"))  {
19.
20.             //没有播放中
21.             if (!music.isPlaying){
22.                 //播放音乐
23.                 music.Play();
24.             }
25.
26.         }
27.
28.         //关闭音乐按钮
29.         if (GUI.Button(new Rect(10, 60, 100, 50), "Stop music"))  {
30.
31.             if (music.isPlaying){
32.                 //关闭音乐
33.                 music.Stop();
34.             }
35.         }
36.         //暂停音乐
37.         if (GUI.Button(new Rect(10, 110, 100, 50), "Pause music"))  {
38.             if (music.isPlaying){
39.                 //暂停音乐
40.                 //这里说一下音乐暂停以后
41.                 //点击播放音乐为继续播放
42.                 //而停止以后在点击播放音乐
43.                 //则为从新播放
```

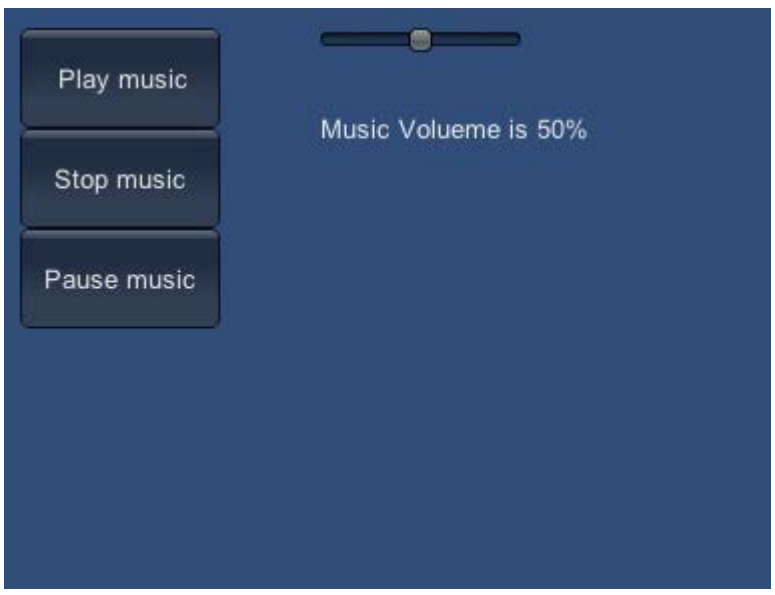
```
44.                 //这就是暂停与停止的区别
45.                 music.Pause();
46.             }
47.         }
48.
49.         //创建一个横向滑动条用于动态修改音乐音量
50.         //第一个参数 滑动条范围
51.         //第二个参数 初始滑块位置
52.         //第三个参数 起点
53.         //第四个参数 终点
54.         musicVolume = GUI.HorizontalSlider (new Rect(160, 10, 100, 50), musicVolume, 0.0F, 1.0F);
55.
56.         //将音量的百分比打印出来
57.         GUI.Label(new Rect(160, 50, 300, 20), "Music Volueme is " + (int)(musicVolume * 100) + "%");
58.
59.         if (music.isPlaying){
60.             //音乐播放中设置音乐音量 取值范围 0.0F 到 1.0
61.             music.volume = musicVolume;
62.         }
63.     }
64. }
```

```
using UnityEngine;
using System.Collections;

public class audio : MonoBehav
```

将 audio.cs 绑定在摄像头上，将 audio 游戏对象拖动赋值给 Music 这个 AudioSource 这个对象。这里强调一下 AudioListenr ，它音频监听器，用来监听音乐文件的播放。这是一个重要的属性，一定要勾选，只有勾选后才可以进行音乐的播放。




build and run 运行我们这个游戏 Demo，一切功能完美实现，哇咔咔～




最后欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，大家一起学习一起进步，哇咔咔～～～ 附上 Unity3D 工程的下载地址，Xcode 项目我就不上传了，须要的自己导出。今天心里有点不高兴！ 5555555555。就这样晚安～

下载地址：http://download.csdn.net/detail/xys289187120/3811441

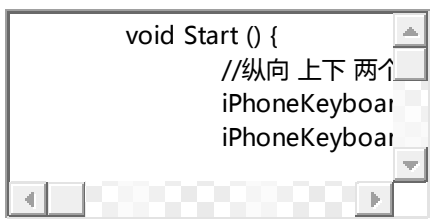
# 第十六章 Unity3D 游戏引擎之感应 IOS 设备旋转与 iPhone 键盘事件

iPhone iPad iTouch 旋转设备都支持屏幕 4 个方向的任意旋转，那么强大的 Unity3D 游戏引擎当然也支持啦，虽然很多游戏都为了避免麻烦强制的不让屏幕旋转，但是做为学习我们还是知道一下为好，因为 Unity3D 在处理屏幕旋转实在是非常方便，下面 MOMO 将以一个例子向各位盆友们介绍 Unity3D 屏幕的哪些事儿~ ~。

强制屏幕四个方向不旋转的方法

[csharp] view plaincopyprint?

```
1.  void Start () {
2.      //纵向 上下 两个方向
3.      iPhoneKeyboard.autorotateToPortrait = false;
4.      iPhoneKeyboard.autorotateToPortraitUpsideDown = false;
5.
6.      //横向 上下两个方向
7.      iPhoneKeyboard.autorotateToLandscapeLeft = false;
8.      iPhoneKeyboard.autorotateToLandscapeRight = false;
9.  }
```




自动旋转屏幕的方法，此方式适用于 Unity3.3 及一下的版本。

Input.deviceOrientation 可以得到当前 IOS 设备屏幕的方向状态。

Screen.orientation 设置屏幕的反转情况

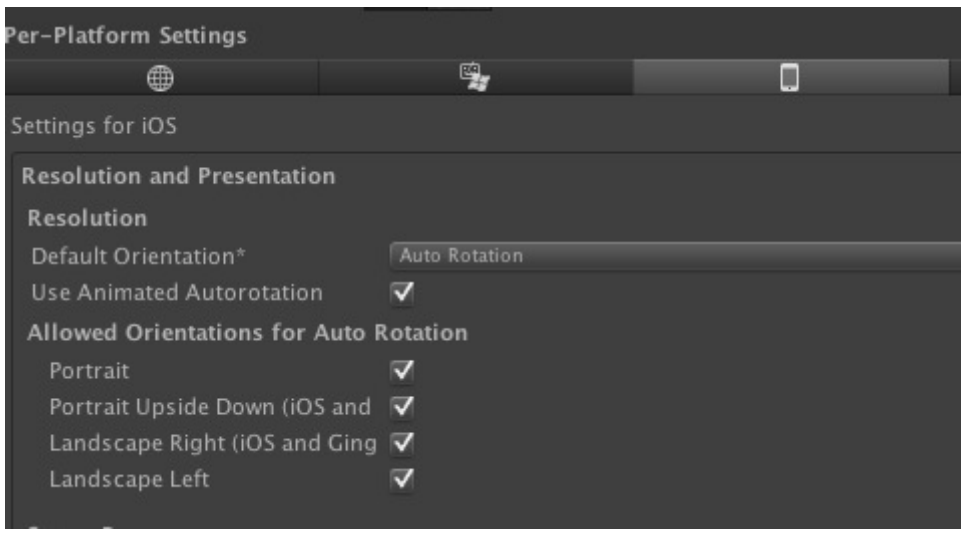
[csharp] view plaincopyprint?

```
1.  void Update () {
2.      //处理横向两个方向旋转
3.      if(Input.deviceOrientation == DeviceOrientation.LandscapeLeft)
4.      {
5.          if (Screen.orientation != ScreenOrientation.LandscapeLeft) {
6.              Screen.orientation = ScreenOrientation.LandscapeLeft;
7.          }
8.      }else if(Input.deviceOrientation == DeviceOrientation.LandscapeRight)
9.      {
10.         if (Screen.orientation != ScreenOrientation.LandscapeRight) {
11.             Screen.orientation = ScreenOrientation.LandscapeRight;
12.         }
13.
14.     }else
15.     //处理纵向两个方向的旋转
```

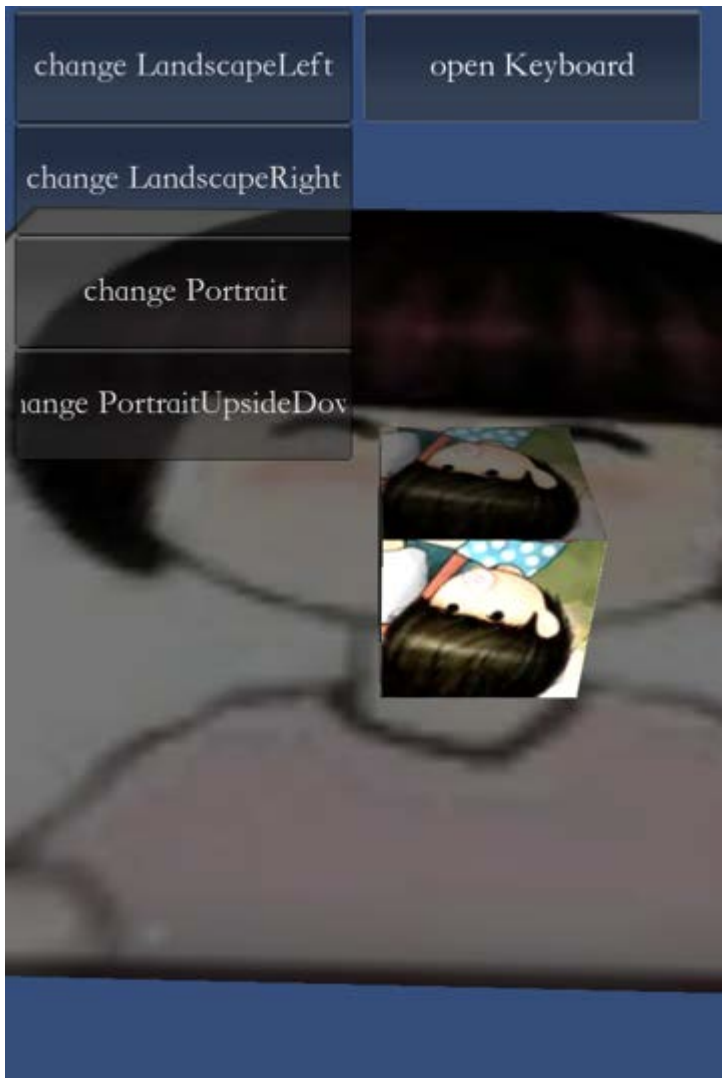
```
16.         if(Input.deviceOrientation == DeviceOrientation.Portrait)
17.         {
18.             if (Screen.orientation != ScreenOrientation.Portrait) {
19.                 Screen.orientation = ScreenOrientation.Portrait;
20.             }
21.         }else if(Input.deviceOrientation == DeviceOrientation.PortraitUpsideDown)
22.         {
23.             if (Screen.orientation != ScreenOrientation.PortraitUpsideDown) {
24.                 Screen.orientation = ScreenOrientation.PortraitUpsideDown;
25.             }
26.         }
27.     }
```

```
void Update () {
                    //处理横向两个方
                    if(Input.deviceO
                    {
                                if (Scre
```

3.4 及以上的版本可以在 Setting for IOS  设置中直接设置屏幕旋转。




下面的游戏例子，通过左边的按钮直接切换屏幕旋转状态，右边的按钮打开 iPhone 输入状态框。




[csharp] view plaincopyprint?

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class Main : MonoBehaviour {
5.
6.      //键盘输入
7.      private iPhoneKeyboard keyboard;
8.
9.      //字体皮肤
10.     public GUISkin fontSkin;
11.
12.     // Use this for initialization
13.     void Start () {
14.     }
15.
16.     // Update is called once per frame
17.     void Update () {
18.     }
19.
20.
21.     void OnGUI() {
22.         //设置皮肤
23.         GUI.skin = fontSkin;
24.
25.         //强制屏幕纵向
26.         if (GUI.Button(new Rect(10, 10, 300, 100), "change LandscapeLeft"))  {
```

```
27.             Screen.orientation = ScreenOrientation.LandscapeLeft;
28.         }else if (GUI.Button(new Rect(10, 110, 300, 100), "change LandscapeRight"))  {
29.             Screen.orientation = ScreenOrientation.LandscapeRight;
30.         }else
31.
32.         //强制屏幕横向
33.         if (GUI.Button(new Rect(10, 210, 300, 100), "change Portrait"))  {
34.             Screen.orientation = ScreenOrientation.Portrait;
35.         }else if (GUI.Button(new Rect(10, 310, 300, 100), "change PortraitUpsideDown"))  {
36.             Screen.orientation = ScreenOrientation.PortraitUpsideDown;
37.         }
38.
39.
40.         if (GUI.Button(new Rect(320, 10, 300, 100), "open Keyboard"))  {
41.             //打开 iphone 输入框
42.             //第一个参数 默认显示 test
43.             //第二个参数 设置输入框类型，这里为默认，什么都可以输入
44.             keyboard = iPhoneKeyboard.Open("test",iPhoneKeyboardType.Default);
45.
46.         }
47.
48.         if(keyboard != null){
49.
50.             if (keyboard.done){
51.                 //输入完毕后 点击 done 输入输入内容
52.                 Debug.Log( keyboard.text)   ;
53.             }
54.         }
55.
56.     }
57. }
```

```
using UnityEngine;
using System.Collections;

public class Main : MonoBehav
```

iPhoneKeyboardType 键盘类型几个比较重要的参数，盆友们可是输入试一试就知道效果啦。我就不截图了 ~

iPhoneKeyboardType.NumbersAndPunctuation： 输入标点符号与数字

iPhoneKeyboardType.URL:输入网址

iPhoneKeyboardType.PhonePad:输入电话

iPhoneKeyboardType.NumberPad:输入数字

iPhoneKeyboardType.EmailAddress:输入 Email


屏幕方向不仅可以感应 IOS 设备平面 4 个方向，还可以感应屏幕上下方向。

屏幕面朝上：LandscapeLeft.FaceUp

屏幕面朝下：LandscapeLeft.FaceDown

最后欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，总的来说这一章还是比较简单的，代码我就不上传了。哇咔咔～

强烈感谢四角线技术大牛～　我愿和 大家好好学习 !!!

# 第十七章 Unity3D 游戏引擎之游戏对象的访问绘制线与绘制面详解

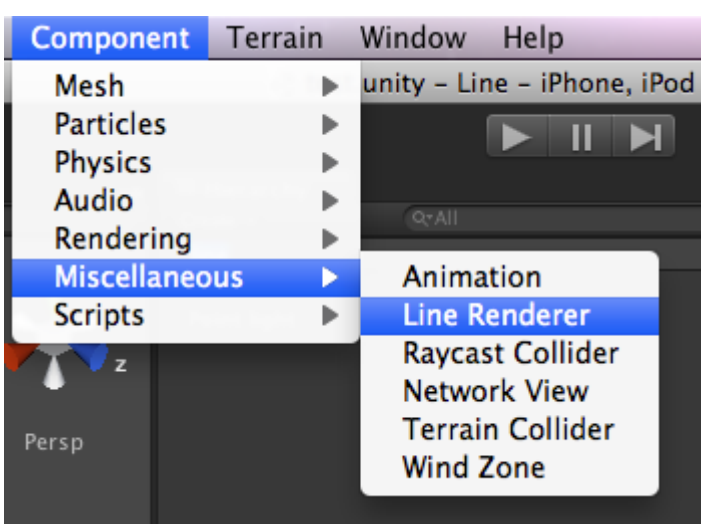
一眨眼学习 Unity3D 也有一段时间了，基本已经拿下了这套游戏引擎，回过头来想想以前写的 RPG 游戏引擎，越来越发现以前写的就是垃圾。人果然是要不断学习与不断进步，好好学习，天天向上。哇咔咔～ 加油 !!

最近做一个项目须要去绘制线与绘制面，那么把这两天的学习笔记整理一下，一是给自己留作备忘，二是方便初学者学习。

任何一个无规则曲线它都是有若干个线段组成，及时是圆形它也是又若干个线段组成的，也就是说将若干个线段拼接起来就是我们须要的无规则曲线～那么在 3D 的世界中我们须要知道 X Y Z 三个点来确定一条 3D 线段。

首先先使用 Unity 编辑器的方式来添加一条线～

Unity -> GameObject -> Create Empty 创建一个空的对象，我命名为 line。然后点击 Component -> Miscellaneous -> Line Renderer 给 line 添加一个线渲染器的属性，Line Renderer 是非常中的属性，下面我会详细的说明。



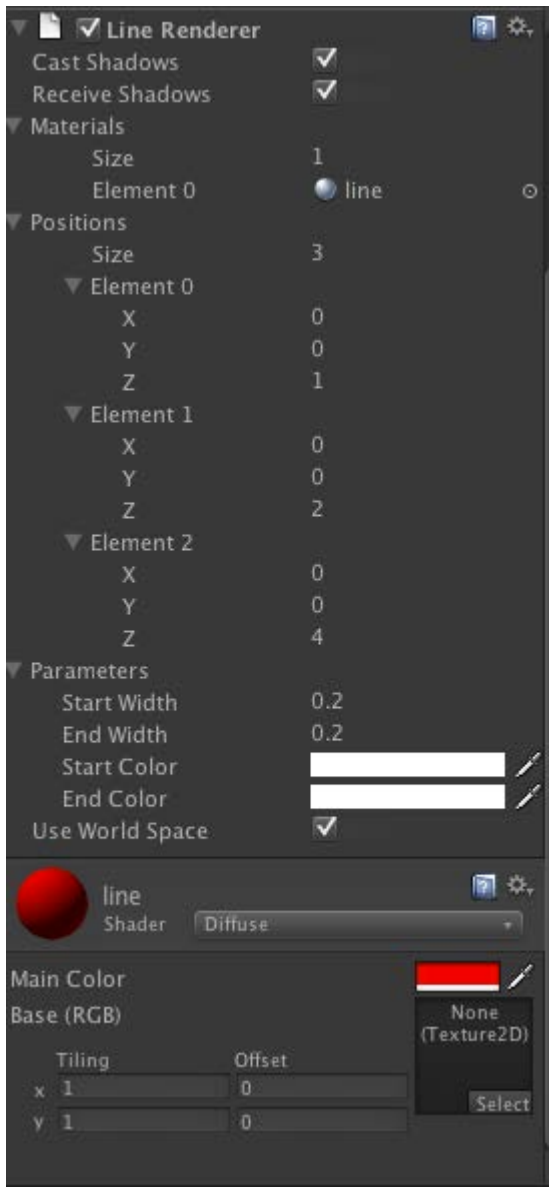

Create -> Material 创建一个材质，做来这个线段的贴图，下面我们看看 Line Renderer 的一些重要参数。

Cast Shadows: 是否投射阴影。

Receive Shadows: 是否接收阴影。

Materials :设置材质，这里可以设置多个材质， line 就是上面我们创建的材质，这里我给 line 这个材质涂上了红颜色。

Positions:这个属性就比较重要了 ,它是专门设置线段在 3D 世界中的点的坐标 ,size 设置点的数量 为 3 那么将会有 3 个点 ,Element 0 Element 1 Element 2 这三个点将确定这条曲线分为两段，第一段是（0,0,1） 到 （0,0,2），第二段是（0,0,2）到（0,0,4）。




Paramerters

StartWidth :设置起点的宽度

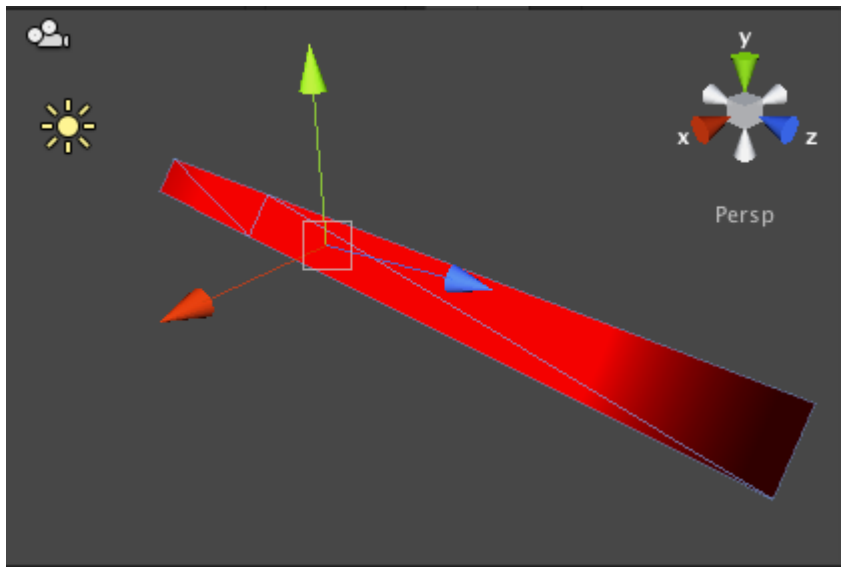
EndWidth:  设置终点的宽度 ，这两项数值默认为 1，但是现实起来很宽，所以一般都设置为 0.几~

Start Color:  设置起点颜色

Start Color:  设置终点颜色

Use World Space  使用世界坐标系

大家看看效果，清楚的可以看到曲线分为两部分，第一部分较短  第二部分较长。




怎么样？ 绘制线的方法不难学吧？在已知线段位置的情况下我们可以使用上面的方法去设置这条线，但是如果线段的位置是在游戏过程中动态的产生就得在代码中去动态的去设置。

创建脚本 Main.cs 绑定到摄像头上，运行游戏绘制线的话须要在 Main 中去拿到 line 这个对象的实例。这就是一个比较重要的知识点了。

一些重要的方法，在编辑器中编辑的一些东西，在代码中也可以做到。

LineRenderer.SetWidth(0.1,0.1)； 设置线段起始点与结束点的宽度 （参数 1 为起始点 参数 2 为结束点）

LineRenderer.SetColor(Color.black,Color.white); 设置线段起始点与结束点的颜色 （参数 1 为起始点颜色 参数 2 为结束点颜色）

LineRenderer.SetVertexCount(5); 设置线段数量。

LineRenderer.useWorldSpace = true; 是否使用世界坐标系，和上面编辑器对应。

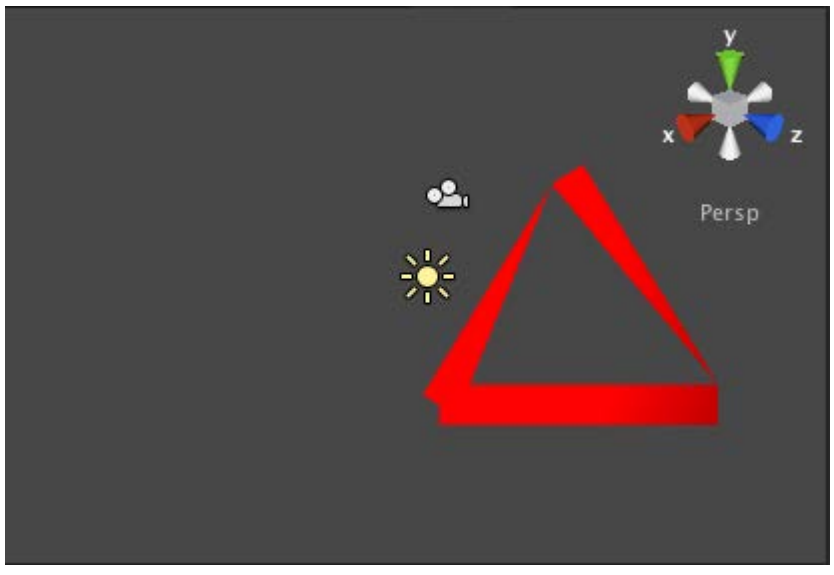
[csharp] view plaincopyprint?

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System.Threading;
4.
5.  public class Main : MonoBehaviour {
6.
7.      //游戏对象，这里是线段对象
8.      private GameObject LineRenderGameObject;
9.
10.     //线段渲染器
11.     private LineRenderer lineRenderer;
12.
13.     //设置线段的个数，标示一个曲线由几条线段组成
14.     private int lineLength = 4;
15.
16.     //分别记录 4 个点，通过这 4 个三维世界中的点去连接一条线段
```

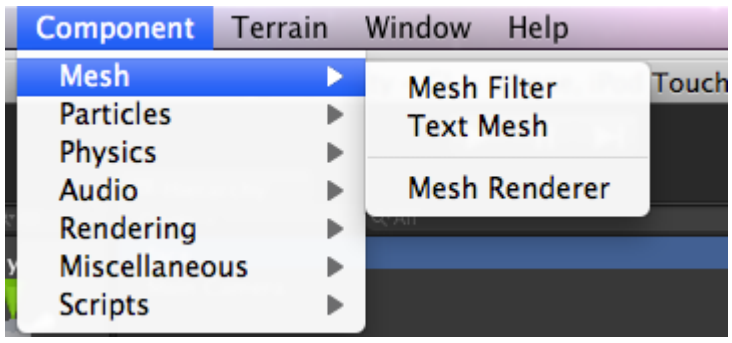
```
17.     private Vector3 v0 = new Vector3(1.0f,0.0f,0.0f);
18.     private Vector3 v1 = new Vector3(0.0f,1.0f,0.0f);
19.     private Vector3 v2 = new Vector3(0.0f,0.0f,1.0f);
20.     private Vector3 v3 = new Vector3(1.0f,0.0f,0.0f);
21.
22.     void Start(){
23.
24.         //通过之前创建的对象的名称，就可以在其它类中得到这个对象，
25.         //这里在 main.cs 中拿到 line 的对象
26.         LineRenderGameObject = GameObject.Find ("line");
27.
28.         //通过游戏对象，GetComponent 方法 传入 LineRenderer
29.         //就是之前给 line 游戏对象添加的渲染器属性
30.         //有了这个对象才可以为游戏世界渲染线段
31.         lineRenderer = (LineRenderer)LineRenderGameObject.GetComponent ("LineRenderer");
32.
33.         //设置线段长度，这个数值须要和绘制线 3D 点的数量想等
34.         //否则会抛异常~ ~
35.         lineRenderer.SetVertexCount(lineLength);
36.
37.
38.     }
39.
40.
41.     void Update() {
42.
43.         //在游戏更新中去设置点
44.         //根据点将这个曲线链接起来
45.         //第一个参数为 点的 ID
46.         //第二个 参数为点的 3D 坐标
47.         //ID 一样的话就标明是一条线段
48.         //所以盆友们须要注意一下！
49.
50.         lineRenderer.SetPosition (0, v0);
51.         lineRenderer.SetPosition (1, v1);
52.         lineRenderer.SetPosition (2, v2);
53.         lineRenderer.SetPosition (3, v3);
54.
55.
56.     }
57.
58. }
```

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

public class Main : MonoBehav
```

通过上面代码的设置 ,运行游戏 ,发现全新的一个三角形曲线赫然的映入我们的眼帘 ,有了上面的方法我们就可以组合的绘制出各种各样的 3D 游戏曲线了 ,这里 MOMO 使用的是颜色，大家也可以添加一个贴图~




绘制面的话~,因为 3D 世界中游戏面全都是用三角形来拼起来的( 出于效率的考虑 ),所以大家所看到的面其实都是用三角面拼接起来的。那么下面我们看看在 Unity3D 中如何绘制三角形平面。先创建一个 GameObject 对象,我命名为 face，然后给这个对象添加两个 Mesh 属性， 添加 MeshFilter(网格过滤器)与 mesh Renderer(网格渲染器)属性，这样子就可以绘制网格面了。




为了给网格面贴图，我给 mesh Renderer 渲染器添加一个材质，贴上一张图片~ 在 3D 的世界中两点可以确定一条线，那么三点就肯定能确定一个面，3D 世界中的面一定是三角形拼凑组成，任何 3D 的面的所需要的坐标点的数量 一定是 3 的倍数。和绘制线一样，MOMO 还是创建一个脚本绑定在摄像头中，然后去访问上面创建的 face 游戏对象 去渲染三角面~

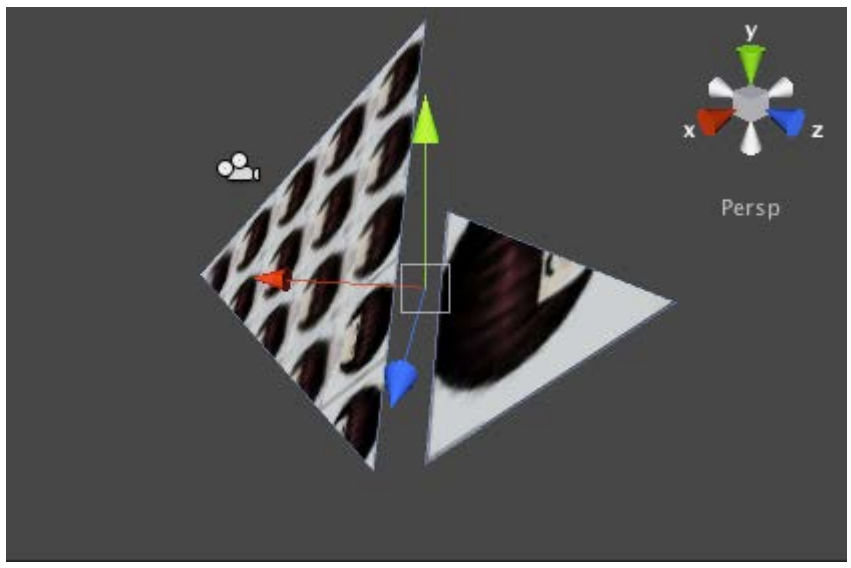
[csharp] view plaincopyprint?

```
1.  using UnityEngine;
2.  using System.Collections;
3.
```

```
4.  public class Main : MonoBehaviour {
5.
6.
7.      // Update is called once per frame
8.      void Update () {
9.
10.         //通过 object 对象名 face 得到网格渲染器对象
11.         MeshFilter meshFilter = (MeshFilter)GameObject.Find("face").GetComponent(typeof(MeshFilter));
12.
13.         //通过渲染器对象得到网格对象
14.         Mesh mesh = meshFilter.mesh;
15.
16.         //API 中写的不是提清楚，我详细的在说一遍
17.
18.         //设置顶点，这个属性非常重要
19.         //三个点确定一个面，所以 Vector3 数组的数量一定是 3 个倍数
20.         //遵循顺时针三点确定一面
21.         //这里的数量为 6 也就是我创建了 2 个三角面
22.         //依次填写 3D 坐标点
23.         mesh.vertices = new Vector3[] {new Vector3(5, 0, 0), new Vector3(0, 5, 0), new Vector3(0, 0, 5),new Vector3(-5, 0, 0), new Vector3(0, -5, 0), new Vector3(0, 0, -5)};
24.
25.         //设置贴图点，因为面确定出来以后就是就是 2D
26.         //所以贴纸贴图数量为 Vector2
27.         //第一个三角形设置 5 个贴图
28.         //第二个三角形设置一个贴图
29.         //数值数量依然要和顶点的数量一样
30.         mesh.uv = new Vector2[] {new Vector2(0, 0), new Vector2(0, 5), new Vector2(5, 5),new Vector2(0, 0), new Vector2(0, 1), new Vector2(1, 1)};
31.
32.
33.         //设置三角形索引，这个索引是根据上面顶点坐标数组的索引
34.         //对应着定点数组 Vector3 中的每一项
35.         //最后将两个三角形绘制在平面中
36.         //数值数量依然要和顶点的数量一样
37.         mesh.triangles= new int []{0,1,2,3,4,5};
38.
39.     }
40. }
```

```
using UnityEngine;
using System.Collections;

public class Main : MonoBehav
```

Build and Run 运行效果后清楚的可以看到两个三角形。MOMO 提醒大家 善用三角平面可以绘制出各种各样的游戏面出来噢～哇咔咔～




最后欢迎各位盆友可以和 MOMO 一起讨论 Unity3D 游戏开发，总的来说这一章还是比较简单的，不过也是非常重要的一章。代码我就不上传了。哇咔咔～ MOMO 愿和 大家好好学习，大家一起进步哈～ !!!