官网M2H网络游戏教程(翻译中)

http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif(2013-01-13 18:02:07)

[[http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif](javascript:;)转载▼](javascript:;)

|  |  |
| --- | --- |
| 标签：  [it](http://search.sina.com.cn/?c=blog&q=it&by=tag) | 分类： [Unity3d](http://blog.sina.com.cn/s/articlelist_1324845813_1_1.html) |

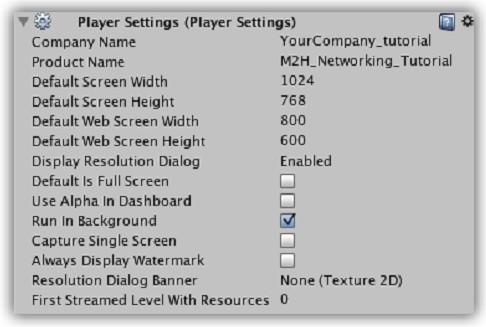
结合这个文档是一个包括用于本教程的Unity工程的压缩文件.设想你已经熟悉Unity3D编辑器和一些基础脚本知识了.如果你还没做到请查看视频教程.

多个玩家需要至少有两个程序案例(服务器和用户),该教程中建议你在编辑游戏视窗运行服务器而在网络视窗运行注册用户.

也许你要使用到教程资源,在你修改这个项目文件之前最好备份一份.因为你自己的项目必须确保"Run in backgroud"是开启的,以便能够在运行服务器的同时在后台运行而不会休眠.这将保持服务器游戏在后台运行.否则,你在前台运行用户程序时不能够连接到你的服务器.你能在"Edit->Project settings->Player"菜单中找到这个选项。

打开第一个教程场景：在这个场景里你能找到”Tutorial 1/Tutorial\_1”。这个场景有一个摄像机，一个带有网络连接脚本的游戏对象和一个显示场景标题的游戏对象。

建立一个Webplayer然后运行它。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=4ef78af501017iut&url=http://s4.sinaimg.cn/orignal/4ef78af5nd33526165173)

也在编辑器中启动场景点击打开“Start a server”按钮。（用默认的IP地址和端口）。

在Web窗口点击“Connect as client”。

你现在在你的两个实例窗口可以看到“Connection status:Client!”和“Connection startus:Sverver!”了。

恭喜你，你已经连接成功了！

这个都很容易，幸运的是代码不太难。看看这个位于你熟悉的编辑器里的“Tutorial 1/Connect.js”文件。所有用在这个教程里的代码都能从OnGUI();函数里找得到。看一看这个函数确保你能够明白它是怎样工作，这个代码有很好的注释，否则。我们很容易就执行了最重要的部分。

在脚本的最上面你有个变量分别是（connectToIP和connectPort）用来获得在GUI中输入的内容。这些值是在按连接按钮里用的。GUI函数被分成四个部分。分别是：servers,connected clients,connection clients和disconnected clients。我们用这个提供网络状态叫“Network.peerType”很轻易检测我们当前的连接状态。我们调用这个Network.Connect函数把用户连接到服务器上。这个函数让IP地址，端口和任意一个密码参数。启动一个相似的简单的函数：Network.InitializeServer.这个带有一个端口和允许最大的连接人数的参数。要知道也可以在一运行的服务器上降低最大连接数。但不能设置初始化服务器时设置的更高的的连接数。在你连接到一个服务器或初始化一个服务器之前你的脑海里要确保一个更高的设置数量。在相应的connection/initiallizing代码上面你还能看到”Network.useNat”

NAT connection(Network.useNat)

由于我们不打算用Network Address Tanslation所以我设置Network.useNat为false(假).NAT对于支持路由器（在局域网内（LAN））是非常有用的。这个网络demo只能在LAN内部运行。你不能与你朋友家的电脑连接（除非你或者他（她）有一个无限制的防火墙）。更多关于NAT信息请看：<http://unity3d.com/support/documentation/Components/net-MasterServer.html>

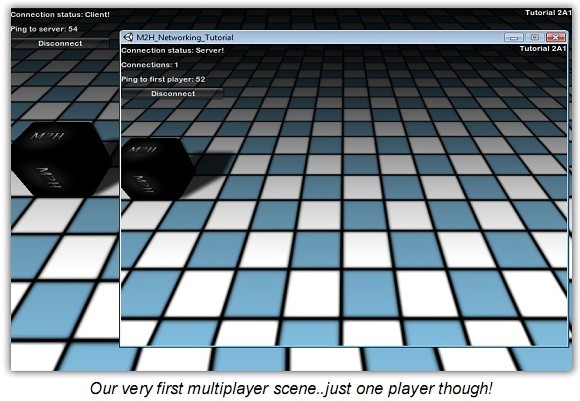
现在再看最后的几行代码。函数自动被Unity调用。值得注意的是，你不需要这些功能在您的代码里，如果你不想使用可以删除它们，这个demo将仍然工作。用户和服务器应该完全可以自己说明,只在用户机或只在服务器上被调用.如果你想通过函数传递参数,可以查阅Unity指南研究这些函数.

最后三个函数是不一样的.OnFailedToConnectToMasterServer是当一个用户不能连接到主服务器上时调用的.关于主服务器更详细的介绍后面会继续的,OnNetworkInstantiate这个函数是实例化对象里被调用的.稍后我也们也能看到.OnSerializeNetworkView是在服务器与用户之间发送消息的两个方式之一.RPC是调用网络消息的(你自己定义的函数).在接下来的指南中我会看到serialization 和RPC是如何使用的.

有关于这个教程的你可以看Network这个章节.包括"Messages Sent","Class Variables"和"Class Functions":

<http://unity3d.com/support/documentation/ScriptReferecne/Network.html>

现在你知道在手册的哪儿找到这些信息了.我们已经讨论了大约这些的75%!

[](http://photo.blog.sina.com.cn/showpic.html#blogid=4ef78af501017iut&url=http://s12.sinaimg.cn/orignal/4ef78af5n7b87e525afab)

**指南2A:Server plays,client observes,no instantiating.**

Tutorial 2/Tutorial 2A1

别被标题吓倒了,打开"Tutorial 2/Tutorial 2A1".指南1里的连接(connection code)代码被挂在"Connect"游戏对象上.另外"PlayerCube"物体也有一个"Tutorial 2A1.js"脚本和一个"NetworkView"组件.每个发送或接收网络信息的对象都需要一个NetworkView组用.在整个被脚本引用的游戏中你仅使用一个networkview(网络视窗),但在场景不影响.通常只能你要使用网络的每个对象添加网络视窗.

运行带有一个服务器和用户的demo.在用户窗口能看到服务器窗口中方块体的任意移动.这一切的神奇都得感谢网络视窗和移动代码.看一看绑定到方块体上的"Tutorial 2A1.js"脚本.这个代码只在服务器上运行:当服务器使用移动键,它会立刻移动方块.别担心在前三个指南里运动延迟的问题,我会在这儿解决这个问题.

现在怎样做才能让用户知道服务里的角色运动呢?看一看指定给这个方块体的NetworkView组件.它应该是方块的"transform"(变换),是指Unity会自动发送变换信息包括(三维的位置、旋转和缩放)。它只是从服务器发送到用户（不是其它方式）因为服务器是这个NetworkView的“主人”（用户不发送它们的信息。它们只能接收信息）。

让我们看看其余的NetworkjView选项完成这个项目。PlayerCube's(玩家方块体的)网络视窗的“State synchronization”(数据同步)选项被设置成“Reliable compressed”。这意味着如果数值改变了它将仅仅只发送过去现在的游戏对象的值。如果服务器移动方块体不到15秒钟，它不会发送任何数据。哈哈，聪明吧！“不可靠的”选项在数据是否改变都会发送一个忽视信息。最后设置“State synchronization”为"Off"在此个网络视窗上会停止所有的网络同步。如果你的网络视窗看来到物体对象，它将不发送任何数据并且同步选项会（但不是必须的）被设置成“Off”(关闭).如果你想问为什么你不用这样一个网络视窗。“Remote Procedure Calls”需要一个网络视窗，但不用“state synchronization”和"observed"选项。你可以RPCs与可见对象一起用。RPCs会在指南2A3里介绍。它基本上是你自己定义网络消息（调用）。

**Tutorial 2/Tutorial 2A2**

如果你想要让方块体在Y方向移动。或者我们想用Unity更进一步作同步控制。打开并运行“Tutorial 2/Tutorial 2A2”。这个游戏播放时完全相同，但是在后台运行的代码改变了。这个指定到PlayerCube网络视窗组件现在应该是"Tutorial 2A2.js"脚本。这表示网络视窗现在正搜索位于那个脚本里的“OnSerializeNetworkView”函数。看一看这个函数。我们现在明确地定义我们想要做网络同步。你能用这个网络同步做你想要的。并且当使用“Reliable delta compressed”时你只需要改变你正在发送的数值。OnSerializedNetworkView函数看起来总是这样奇怪。这个函数是用来发送和接收数据的。Unity确定你是否会能过检查网自己的络视窗发送（“istream.isWriting”）,否则你只能接收（else模块里）。

**Tutorial 2/Tutorial 2A3**

最后有一种发送消息的方式是我最喜欢的。Remote Procedure Calls.之前我提到了它，打开“Tutorial 2/Tutorial2A3”看看它的真面目。这个demo象后两种。这个网络视窗不在显示任何东西（因而网络同步已经被设置成“关闭”）。“Tutorial 2/Tutorial2A3”里特别是这一行：networkView.RPC("SetPosition",RPCMode.Others,transform.position);一个RPC被服务器调用。它作为一个有效的请求其它用户调用“SetPostion”函数，而这个函数拥有服务器的tranform.position作为它的参数（例如：5.2）。然后SetPosition(5.2)被在用户机上调用。这些处理如何运动的：

1。服务器玩家按下一个运动键移动他/她所拥有的游戏角色（14-18行）。

2。服务器检查是否由于网络的上一次更新改变它的最小值，如果是这样就发送一个RPC给每一个玩家除了它自己带有一个新位置的参数（20-25行）。

3。所有的注册用户都接收带有由服务器建立的参数的RPC SetPosition，它们在“它们自己的世界”执行这个代码。

4。现在服务器和用户视窗里方块的位置完全一致！

要能或允许一个函数用作一RPC你需要在javascript里那段代码上面加上[“@RPC](mailto:%E2%80%9C@RPC)”（或者在C#里加[RPC]）。当发送一个RPC里你能指定如下的接收器：

|  |  |
| --- | --- |
| RPCMode.Server | Only send to the server |
| RPCMode.Others | Send to everyone,but the caller itself |
| RPCMode.OthersBuffered | Send to everyone,but the caller itself.Buffered |
| RPCMode.All | Send to everyone,including the caller itself. |
| RPCMode.AllBuffered | Send to everyone,including the caller itself.Buffered |

Buffered意思是无论什么时候有新的玩家连接，他们会接收到这个消息。一个buffered RPC对于一个新生的玩家来说是一个很有用的模版，这个新产生的玩家调用的将被记住并且当一个新的玩家连接时会接收到这个新生的RPC所生成的在这个新玩家加入之前已经一直在玩的玩家。

到目前为止你是否自豪于你已经基本上明白了一切，我们完成了基本的工作和这个项目的绝大部分。我们现在只需要更深入地细化这项工作。

**Tutorial 2B：Server and client(s)play,with instantiating**

我们打算通过一个真实的简单FPS游戏来弄明白一些细节的东西。我们希望能有多个玩家，包括服务器，尽管作为一个专用的服务器，它的玩家可以轻易地删除运行的游戏。为此当他们连接时我们将实例化玩家对象，而不是在场景中事先放好的角色对象。打开“Tutorial 2/Tutorial 2B”场景，这个场景带有一个服务器和一个用户。走一走看看通过网络改变两个玩家移动是否正常。

PlayerCube（玩家的方块体）已经被从场景中删除了，而是用带有Spawnsrcipt.js的生成脚本的游戏对象。当一个玩家（既可以是服务器也可以是用户）启动场景，生成脚本就会实例化我们在脚本里定义的预制对象。实例化对象带有位置、旋转以及组参数。我们将复制这个生成脚本对象的位置和旋转，另外由于我们用0作为组（现在感觉可以忽略它）在生成脚本的On disconnection函数中将删除实例的对象。调用一个Network.Instatiate(网络实例)的对象是由这个对象的拥有都自动实现的.这就是为什么