摘 要

LeapMotion体感控制器是一款通过手势来控制软件或硬件的一种设备，目前

**ABSTRACT**

目 录

1. 研究背景
2. LeapMotion基本原理
3. LeapMotion手势控制的建立
4. LeapMotion与Unity3d的结合实例
5. LeapMotion手势控制的优点与局限性
6. 总结与展望

第一章 研究背景

1.1 虚拟现实

虚拟现实（VR）是一种计算机生成的场景，通过感官和感知来模拟体验。身临其境的环境可以与现实世界类似，也可以是幻想的，创造出在普通物理实验中不可能的体验。增强现实系统也可被视为VR一种形式，其通过将实时摄像头馈送到耳机或者通过智能手机或者平板设备的虚拟信息分层，是用户能够观看三维图像。

目前的虚拟现实技术通常使用虚拟现实耳机或多投影环境，有时结合物理环境或道具来生成模拟用户在虚拟或虚拟环境中的物理存在的逼真图像，声音或者其他感觉。使用虚拟现实设备的人能够观察人造世界，在其中一栋并且与虚拟特征或物品进行交互行为。这种效果通常由头戴式VR设备产生，头戴式显示器在眼镜前方有一个小屏幕，但也可以通过特别设计的带有多个大屏幕的房间创建。

包括通过游戏控制器或其他设备向用户传输振动和其他感觉的VR系统被称为触觉系统。这种触觉信息通常被称为医疗，视频游戏和军事训练应用中的力反馈。

1.2 LeapMotion体感控制器

LeapMotion控制器是一块小巧的USB外围设备，设计用于面朝上的方式放置在桌面上。它也可以安装到虚拟现实的头戴式设备上。使用两个单色红外摄像机和三个红外LED，该设备可以观察一个大约距设备距离大约1米的半球区域。LED产生图案的IR光，相机每秒产生几乎200帧的反射数据。然后通过USB电缆将其发送到主机，由LeapMotion软件通过公司“复杂数学”用以公司尚未公开的方式对其进行分析，以某种方式通过比较两台相机产生的2D数据帧来合成3D位置数据。在2013年的一项研究中，控制器的整体平均精确度显示为0.7毫米。

LeapMotion最初将数千个单元分发给有兴趣为该设备创建应用程序的开发人员。LeapMotion控制器与2013年7月首次发货。2016年2月，LeapMotion发布了其核心测试软件的主要测试版本更新。这款名为Orion的软件专为虚拟现实中的手部追踪而设计。

1.3 Unity

Unity是一款多用途游戏引擎，支持2D和3D图形，拖放功能以及使用C#编写脚本。支持另外两种编程语言：Boo，在Unity5发布时启用，而JavaScript在2017年8月发布的Unity后开始弃用。

该引擎针对以下图形API：Windows和Xbox One上的Direct3d；Linux, macOS和Windows上的OpenGL；Android和iOS上的OpenGL ES; Web上的WebGL；以及视频游戏控制台中的转悠API。此外，Unity支持iOS和macOS上的地基API METAL, Android, Linux和Windows上的Vulkan以及Windows和Xbox One上的Direct3D 12.

在2D游戏中，Unity允许引入sprites和先进的2D游戏渲染器。对于3D游戏，Unity允许为游戏引擎支持的没饿平台指定纹理压缩， mipmap和分辨率设置，并为凹凸贴图，反射映射，视差映射，屏幕空间环境遮挡（SSAO）， 动态阴影贴图，渲染到纹理和全屏幕后处理效果的阴影提供支持。Unity还为开发者体统服务，包括：Unity广告，Unity分析，Unity认证，Unity云构建，Unity Everyplay，Unity IAP，Unity多人游戏，Unity性能报告和Unity写作。

Unity支持使用Cg(Microsoft的高级着色语言修改版本)创建自定义顶点，片段（或像素），向前计算着色器和Unity自己的表面着色器。

第二章 LeapMotion基本原理

2.1 LeapMotion的传感器结构

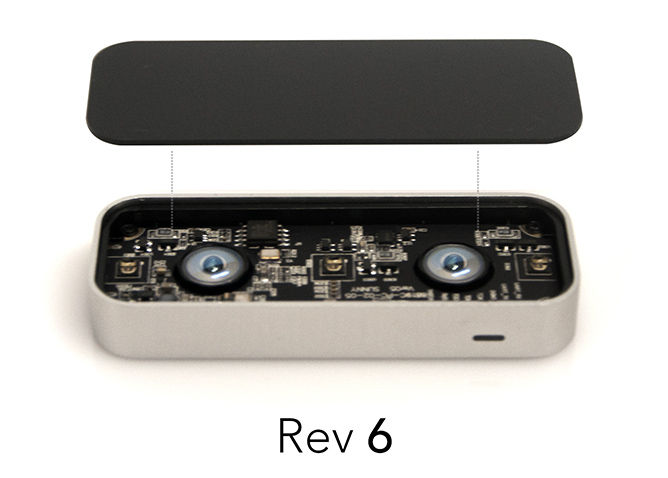
从API的角度来看，LeapMotion传感器的结构如下：

图2-1-1

大体上，Leap传感器根据内置的两个摄像头从不同角度捕捉的画面，重建出手掌在是世界三维空间的运动信息。检测的范围大体在传感器上方25毫米到600毫米之间，检测的空间大体是一个倒四棱锥体。

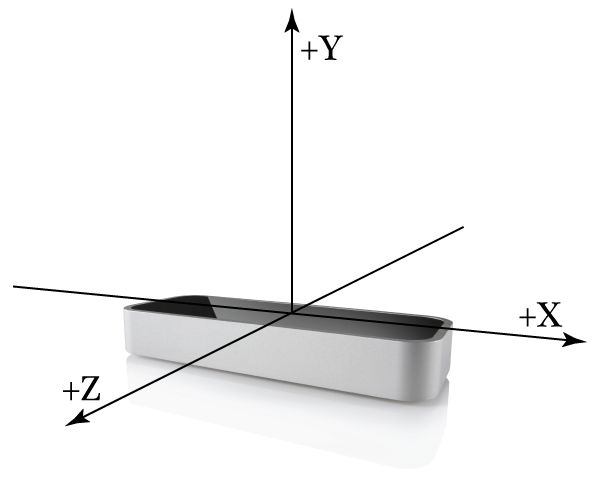
首先，Leap Motion传感器会建立一个直角坐标系坐标的原点是传感器的中心，坐标的X轴平行于传感器，指向屏幕右方，Y轴指向上方，Z轴指向背离屏幕的方向，单位为真实世界的毫米。如图所示：

图2-1-2

在使用过程中，Leap Motion传感器会定期的发送关于手的运动信息，每份这样的信息成为帧（frame）.每一个这样的帧包含检测到的：

·所有手掌的列表及信息；

·所有手指的列表及信息；

点手持工具（细的，笔直的，比手指场的东西，例如一支笔）的列表及信息

·所有可指向对象（Pointable Object），即所有手指和工具的列表及信息；

2.2 Leap Motion的系统架构

Leap Motion软件作为服务（Windows）或后台进程（Mac和Linux）运行。它通过USB总线连接到Leap Motion Controller设备。启用Leap的应用程序访问Leap Motion的服务以接受运动跟踪数据。Leap Motion SDK提供了两种用于获取Leap Motion数据的API：一个本地接口和一个WebSocket接口。本地接口是一个动态库，可以使用它来创建启用了Leap的新应用程序。WebSocket接口允许创建支持Leap的Web应用程序。这些API可以让开发者使用几种编程语言创建支持Leap的应用程序，包括在浏览器环境中运行的JavaScript.

2.2.1 本地应用程序接口

本地应用程序接口通过动态加载的库提供。该库连接到Leap Motion服务并将跟踪数据提供给开发者的应用程序。同时，可以直接在C++和Objective-C应用程序中或者通过为Java，C#和Python提供的语言绑定链接到库。

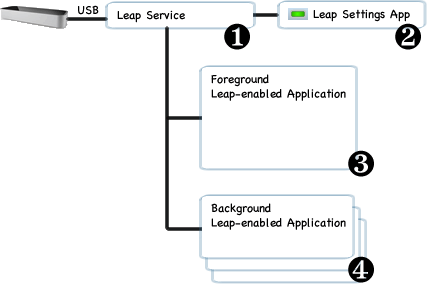


图2-2-1-1

Leap-enabled applications:

·Leap Motion服务通过USB总线从Leap Motion控制器接收数据。它处理该信息后将其发送到运行中的Leap应用程序。默认情况下，该服务仅将跟踪数据发送到前台应用程序。但是应用程序可以请求它们在后台接受数据（可由用户拒绝的请求）。

·Leap Motion应用程序与服务分开运行，并允许计算机用户配置其Leap Motion的安装。Leap Motion应用程序是Windows上的控制面板程序和Mac OS X上的菜单栏应用程序。

·前台的Leap应用程序从服务接种运动的跟踪数据。启用了Leap的应用程序可以使用Leap Motion本地库连接到Leap Motion服务。开发者的应用程序可以直接与Leap Motion本地库（C++和Objective-C）或可用的语言包装库（Java, C#和Python）之一进行链接。

·当Leap的应用程序失去操作系统焦点时，Leap Motion服务将停止向其发送数据。打算在后台工作的应用程序即使在后台也可以请求接受数据的权限。在后台时，配置设置由前台应用程序确定。

2.2.2 WebSocket接口

Leap Motion服务在localhost域上的端口6437上运行WebSocket服务。WebSocket接口以JSON消息的形式提供跟踪数据。JavaScript客户端库可以用于使用JSON消息并将跟踪数据显示为常规JavaScript对象。

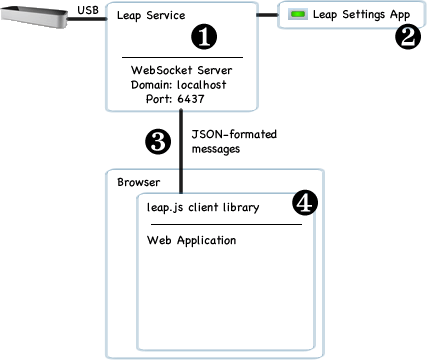


图2-2-2-1

Leap-enabled web applications:

·Leap Motion服务提供了一个监听<http://127.0.0.1:6437>的服务器。

·Leap Motion控制面板允许最终用户启用或者禁用WebSocket服务器。

·服务器以JSOIN消息的形式发送跟踪数据。应用程序可以将配置消息发送回服务器。

·应用程序中应该使用leap.js客户端的JavaScript库。该库建立到服务器的连接并使用JSON消息。JavaScript库提供的API在原理和结构上与本地API类似。

2.3 Leap Motion的数据收集

Leap传感器会给所有这些分配一个唯一标识（ID），在手掌、手指、工具保持在视野范围内时是不会改变的。根据这些ID，可以通过Frame::hand()，Frame::finger()，等函数来查询每个运动对象的信息。

Leap可以根据每帧和前帧检测到的数据，生成运动信息。例如，若检测到两只手，并且两只手都朝一个方向移动，就认为是平移，若像握着球一样转动，则记为旋转。若两只手靠近或分开，则记为缩放。所产生的数据包含：

·旋转的轴向向量

·旋转的角度（顺时针为正）

·描述旋转的矩阵

·缩放因子

·平移向量

对于每只手，可以检测到如下信息：

·手掌中心的位置（三维向量，相对于传感器坐标原点，毫米为单位）

·手掌移动的速度（毫米每秒）

·手掌的法向量（垂直于手掌平面，从手心向外）

·根据手掌弯曲的弧度确定虚拟球体的中心；

·根据手掌弯曲的弧度确定的虚拟球体的半径

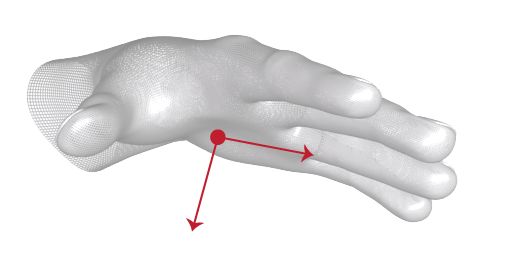
其中，手掌的法向量和方向如下图所示：

图2-3-1

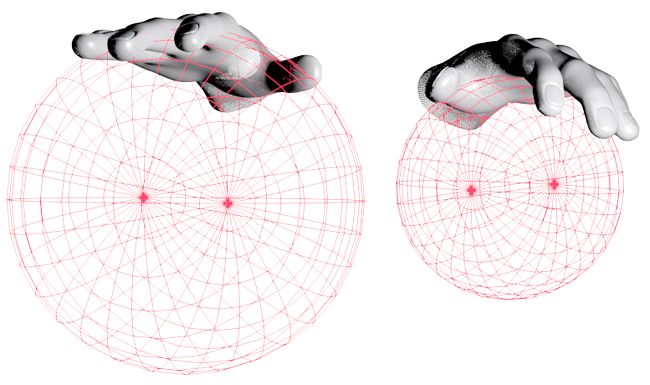
[手掌球]的圆心和半径

图2-3-2

对于每个手掌，可以检测出平移，旋转（如转动手腕带动手掌转动）、缩放（如手指分开聚合）的信息。检测的数据如全局变换一样，包括：

·旋转的轴向向量

·旋转的角度

·描述旋转的矩阵

·缩放因子

·平移向量

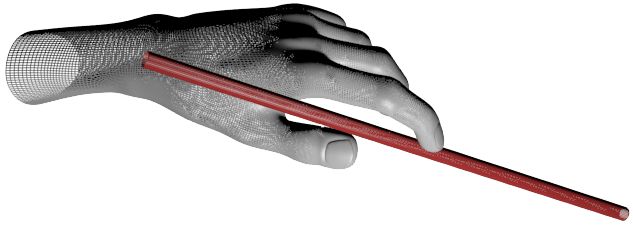
Leap除了可以检测手指外，也可以检测手持的工具。像上文所说的，就是细的，笔直的，比手指长的物件

图2-3-3

对于手指和工具，会统一地成为可指向对象（Pointable Object），每个Pointable Object包含了这些信息：

·长度

·宽度

·方向

·指尖位置

·指尖速度

方向和指尖位置如下图

图2-3-4

Leap Motion可以识别特定的运动模式，用户可以靠这些手势发令。手势和手指，手等其他运动跟踪数据的传回方式一样，每发现一个手势就像帧中自动添加一个手势对象，可以从帧中的手势列表获取手势对象。

Leap Motion可以识别的运动模式包括：

·circle画圆，一根手指画一个圆

·swipe挥扫，手的线性运动

·key tap击键，敲键盘一样的轻击

·screen tap触屏，触摸垂直屏幕一样的轻击

Leap Motion首次识别出一个手势后将其加入帧，如果这是一个持续性动作，Leap Motion将一个更新的手势对象加入后续帧。画圆和挥扫是持续性动作，Leap Motion在每一帧更新这些手势，tap轻击是不联系的动作，所以每次敲击只需一个收拾对象。

程序中，想使用某个手势钱，需要调用Controller类中的enableGesture()方法启用该手势的识别。

画圆：可以用手指和工具画圆，手势开始后Leap Motion会一直更新这个过程直到手势结束或者工具离开园的轨道或动作过慢视为手势结束。相关API：CircleGresture

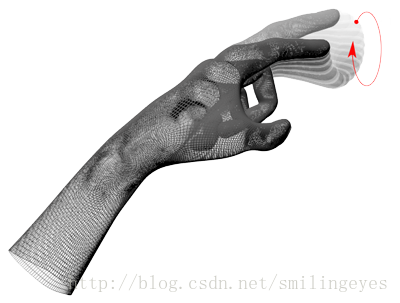


图2-3-5

挥扫，相关API：SwipeGesture

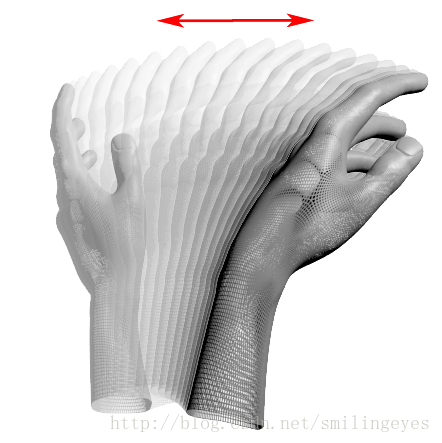


图2-3-6

击键：快速的向下敲击被视为一次击键手势。相关API：KeyTapGesture



图2-3-7

触屏：快速向前轻击被视为一次触屏手势，相关API：ScreenTapGesture



图2-3-8

根据全局的信息、运动变换，手掌、手指和工具的信息和变换，开发者就可以靠这些来制作游戏和程序了。

2.4 LeapMotion的工作原理

2.4.1 LeapMotion的立体视觉原理

Leap Motion 控制器采用立体视觉原理，配备双摄像头的控制器如同人眼一样，能够对空间物体进行坐标定位。为了方便理解，大家可以试着快速轮流闭合左右眼可以看到物体的位置会发生平移，这就是视差。一旦出现视差，就可以在人脑中产生空间纵深的感觉，这就是3D电影的基本工作原理。电影在制作的时候分别拍摄给左右眼看的画面，人为制造视差，所以在观看平面画面时会产生三维空间错觉，当然这个视差是不能随意定义的，必须和人眼瞳孔的间距匹配起来，这个称之为基线长度。不同年龄，性别，种族的人群的诗句基线长度都略有差别，所以3D电影不一定适合所有人群观看。

这种应用于立体视觉的测量方法，被称之为三角测量法。三角测量法是用于定位目标空间位置最常用和最基本的方法，应用场合小到常见的Kinect，激光反求等，大到飞机装配等高精度作业环境。在Leap Motion控制器的双摄像头基线距离固定后，就可以进行设备校准。校准后的控制器可以精确计算出目标对于摄像头的空间坐标。

2.4.2 LeapMotion控制器的工作过程

以上为Leap Motion的基本工作原理，接下来介绍控制器的工作过程。当我们把手伸到控制器的工作区时，两个摄像头需要同时捕捉目标，并且实时计算目标的视差，就可以得到它的空间信息。这里所说的目标是指已经过滤后的目标信息，如指尖和掌心，在这样的情况下，通过控制器再来做三维扫描的方法是无法实现的。控制器的工作区域必须是双摄像头的公共视场区域，所以过于复杂的多点姿势操作，Leap Motion控制器是无法识别的。

2.4.3 LeapMotion的照明部分与计算效率

为了更加方便地识别目标，控制器上的LED灯需要对目标进行照明，加强目标与背景的亮度对比，使之更加容易识别，同时使设备的光线较暗的环境中也可以使用。相反，如果在室外有太阳或者红外光比较充足的地方，则会影响控制器的正常使用。

Leap Motion在算法上主要采用了TBD技术。该技术的一大优点就是能够准确跟踪目标，但是缺点是对于内存有一定压力而且也会需要一定的计算量，并且还采用了高帧率的双摄像头。从官方的文档上可知，在未来Leap Motion会通过数学模型的优化再来减低CPU的占用资源。

第三章 LeapMotion手势控制的建立

3.1 LeapMotion手势在Unity中的建立

3.1.1 Leap Motion资源以及插件

Leap Motion的Unity资源包提供了一个简单的方式将运动控制的手模型加入到Unity游戏去。

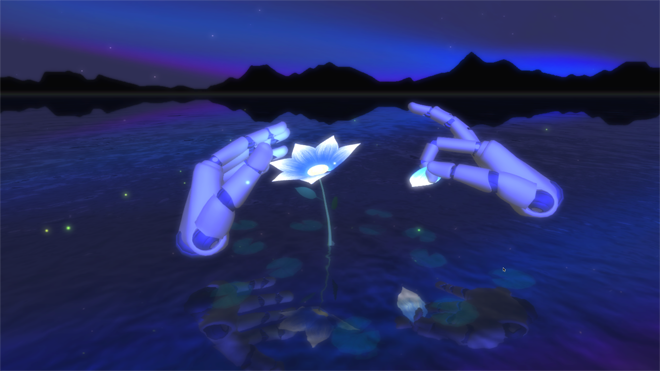


图3-1-1-1

系统要求：

·Leap Motion版本2.3.1+

·Unity 5.1+

·Windows 7，Windows 8，Mac OS X

3.1.1.1 安装

·从Leap Motion 的官方网站上下载最新的资源包。

·打开或者创建一个项目。

·选择Unity中的Assets>Import Package>Custom Package菜单命令。

·选中下载的资源包并且打开，之后便可以导入资源包。

3.1.1.2 故障排除

如果在加载HandController预制体到场景中并运行后没有看到手模型出现，可以检查一下原因：

·手在场景摄像机的事业内，并且不被另一个3D对象所遮挡，同时足够大以便可见。3D指针显示在HandController预制体的位置上方。

·暂停游戏并检查手的模型是否在Unity编辑器的Hierachy结构视图中。如果在的话，在编辑场景视图中找到它们。

·确保Unity Editor（或应用程序，如果在编辑器外部运行）具有OS的输入焦点。（Leap Motion服务仅将数据发送到活动的应用程序）。

·检查任务栏中的Leap Motion图标是否变为绿色。如果图标颜色变暗，Leap Motion的服务将不会发送数据。在这种情况下，仔细检查Leap Motion硬件是否已插入并检查服务是否正在运行。

·从任务栏图标菜单中打开Leap Motion Visualizer。如果在Visualizer中看到了问题报告，那么问题很可能出现在Unity的插件，脚本或者开发者自己的应用程序里。如果Visualizer中没有手，那问题是在Unity引擎之外的。

3.1.2 Unity插件概述

Leap Motion Controller跟踪手部和手指，并以低延迟和高准确度报告位置，速度和方向。该控制器可用于桌面或安装在VR耳机上。

Leap Motion控制器系统包含一个硬件设备和一个作为主机上的服务或后台运行程序运行的软件组件。软件组件分析硬件产生的图像并将跟踪信息发送给应用程序。Leap Motion Unity插件连接到此服务以获取数据，随插件包含的脚本将Leap Motion坐标转换为Unity坐标系。这些脚本和附加图形资源可以轻松地讲3D动作控制的手部添加到Unity场景中。

3.1.2.1 坐标系统

Unity3D为其坐标系统使用左手坐标，Leap Motion API使用右手坐标。（本质上，Z轴指向相反的方向。）Unity也使用默认的米单位，Leap Motion API使用毫米。插件脚本内部转换数据以使用左手坐标系并将距离值缩放到米。

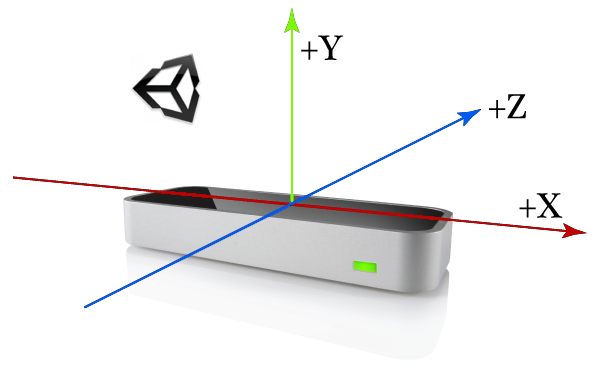


图3-1-2-1-1

桌面方向上，Unity左手坐标系统叠加在Leap Motion设备上。

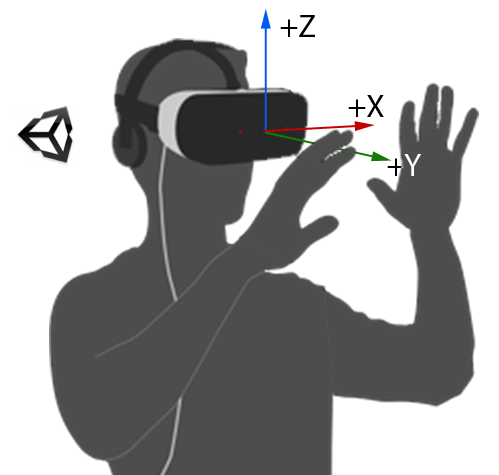


图3-1-2-1-2

Unity坐标系统以HMD方向叠加在Leap Motion设备上。

当你直接从Leap C#库中得到某个跟踪数据类的时候，数据位于本地Leap坐标系中，而不是Unity坐标系。开发者可以直接使用LeapUnityExtensions文件中的程序功能将Leap Motion左边转换为Unity坐标。ToUnity（）将轴从右手转换为左手并返回Unity Vector3类型的对象。

ToUnityScaled（）也可以将坐标从milimeters转换meters。ToUnity（）通常与方向向量一起使用。

3.1.2.2 手部追踪

Leap Motion控制器使用光学传感器和红外灯。这些传感器具有约150°的视野。Leap Motion控制器的有效范围从设备上方约0.03至0.6米。



图3-1-2-2-1

Leap Motion控制器的桌面模式

当控制器清晰，高对比度地查看物体轮廓时，检测和跟踪效果最佳。Leap Motion软件将其传感器数据与人手的内部模型相结合，以帮助应对具有挑战性的跟踪条件。

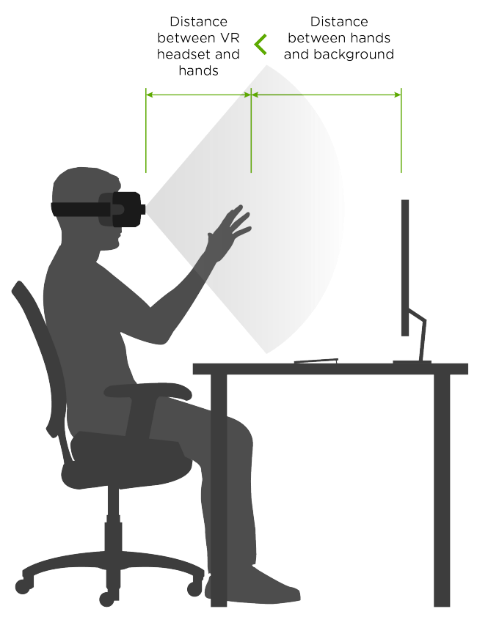


图3-1-2-2-2

特别是在HMD模式下，应该确保传感器与手之间的距离更小

3.1.3 VR设置

Leap Motion Core Unity Assets提供的LMHeadMountedRig预制件，可以为虚拟场景和增强现实场景提供完整的相机和手持控制器设置。这种预制可以实现简单的配置，可以使用Leap Motion相机的图像或3D模型来呈现手部模型。

LMHeadMountedRig中的脚本自动将立体摄像机位置调整到正确的瞳距，并自动补偿AR场景中的视频滞后。

要在Unity中设置虚拟或者增强现实场景：

·在Unity中打开或创建一个新项目。

·启用Unity虚拟现实支持。

·使用Edit>Project Settings>Player菜单命令现实播放器设置。

·在其他设置类别下，选中虚拟现实支持复选框。

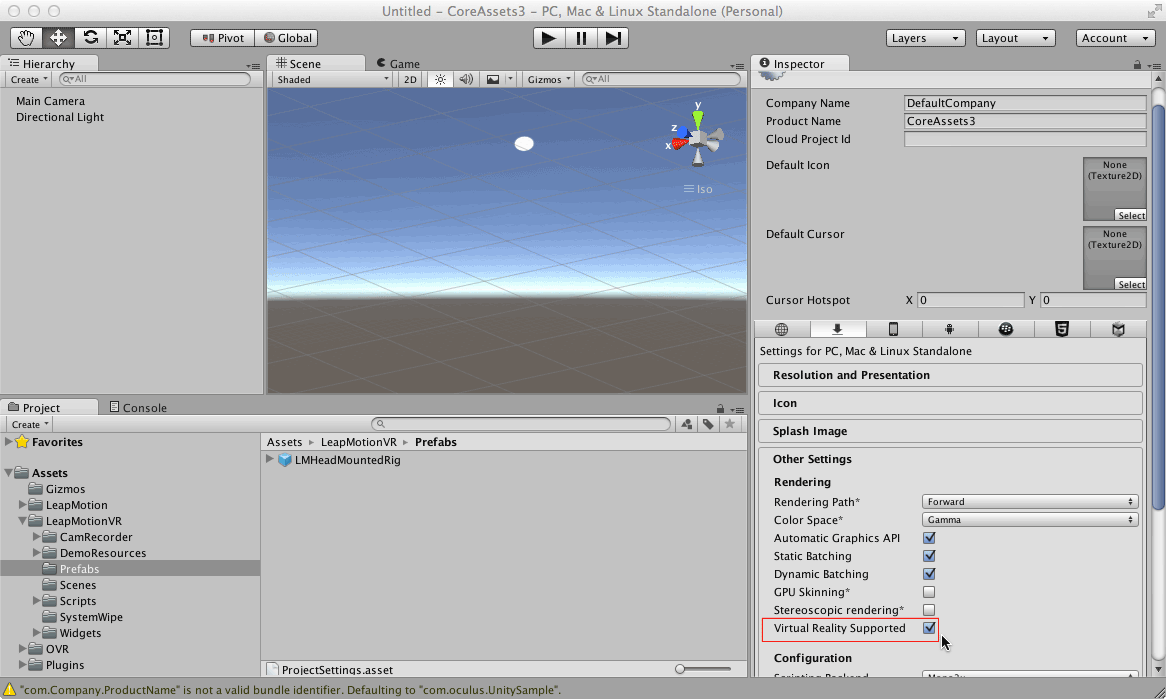


图3-1-3-1

·打开或创建一个新场景

·删除现有的不需要的相机

·从Assets/LeapMotionVR/Prefabs文件夹中将LMHeadMountedRig预制件拖到场景中。

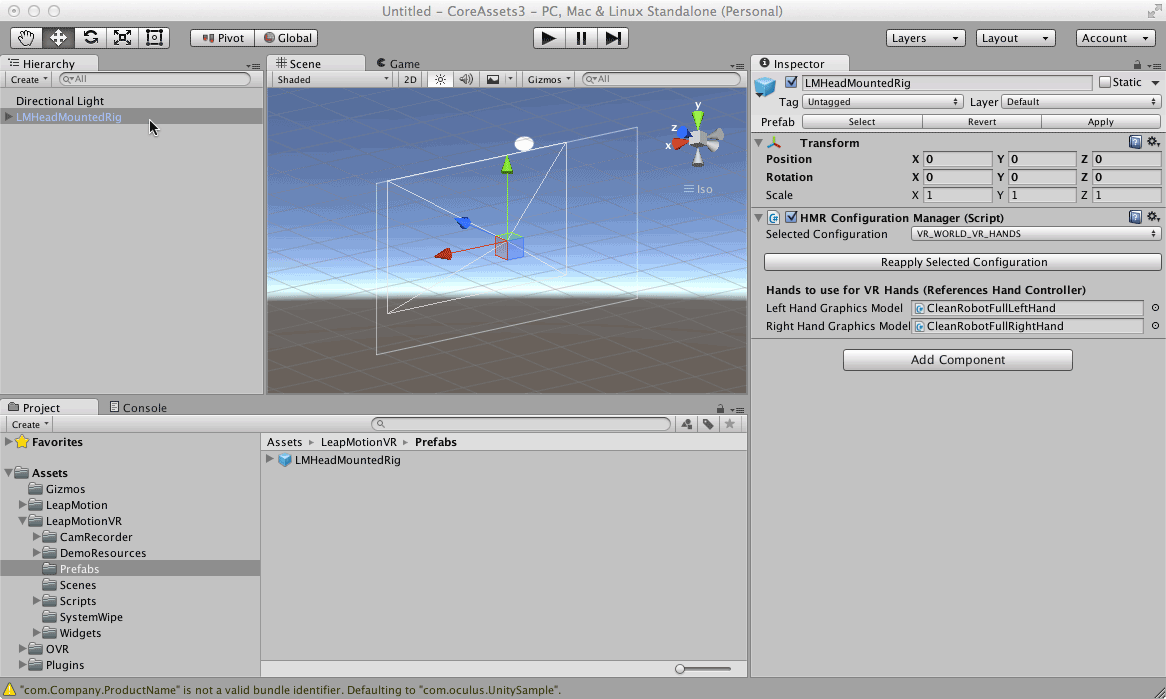


图3-1-3-2

·在场景中设置摄像机的方向和位置。

·根据配置选项，可以选择的有：

·VR World VR hands 不透明3D世界中的3D模型手

·VR World AR hands 带有相机图像指针的不透明的3D世界

·AR World AR hands 相机图像被用作3D世界和手部的背景

·如果使用VR指针选项，可以从Assets/LeapMotion/Prefabs文件夹中指定所需的预制体

·直接在HandController组件上设置手工预制件以及物理手工预制件，它们深深嵌套在LMHeadMountedRig对象内。更改所选配置将重置所有手型号，同样会点击“重新应用所选配置”按钮。

·按播放按钮进行测试。

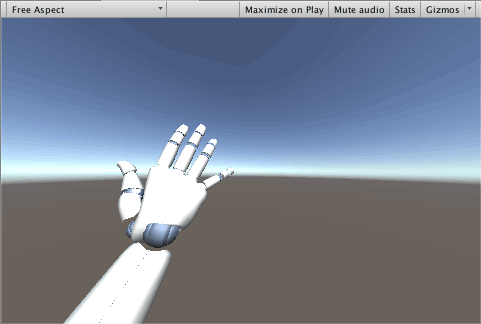


图3-1-3-3

·保存场景。

之后就可以按照所希望地向场景里添加其他内容了。

3.1.4 手的资源包

用于Unity的Leap Motion核心资源包包括大量预制体。开发者可以按照原样使用这些预制体，修改它们或者创建自己的预制件。

有几种不同的创建手的方法，包括为手部分创建独立组件，并分别移动这些部件，创建用骨架操纵的网格手，并通过旋转骨骼是网格变形，以及创建由代码驱动的手部模型后，在其上创建自己的图形。现有的预制件使用这三种方法。

HandController类协调跟踪数据到手和手指的采集和应用。HandModel和FingerModel类用作手和手指动画的基类。有几个扩展类比如HandModel和FingerModel类来实现特定类型的动画。这些特定的类，如SkeletalHand/Finger和RiggedHand/Finger,可用于多个手部设计，只要他们所连接的对象遵循相同的基本结构即可。

图形和物理手分成两个不同的预制件。开发者可以使用任何物理手开发成最适合的手模型。例如，RigidFullHand包括一个刚体和手臂的碰撞体，因此可以将它与包含手臂图形的模型一起使用，当手臂会与场景中其他物体发生碰撞时。

以下是几种手部模型的展示图：

·Image Hands；ImageFullHand；RiggedHand/Finger

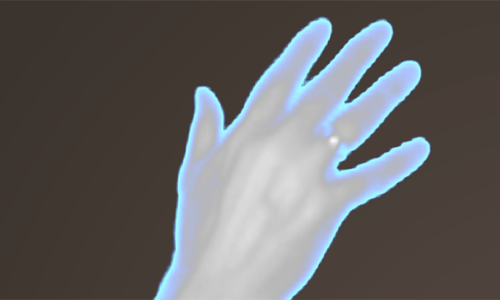


图3-1-4-1

·Procedural Hands；PolyHand1/2/3，DebugHand；PolyHand/Finger，DebugHand/Finger

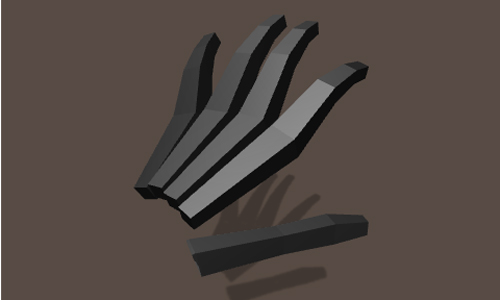


图3-1-4-2

·Component Hands；MinimalHand，CleanRobotHand，GlowRobotHand；SkeletalHand/Finger

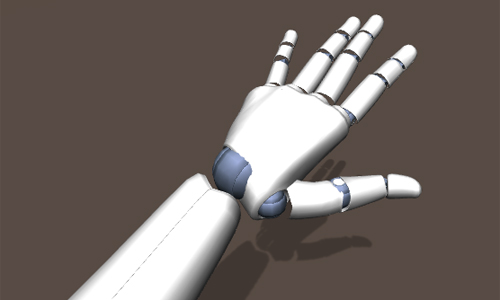


图3-1-4-3

·Rigged Hands；All the human models；RiggedHand/Finger



图3-1-4-4

·Physics Hands；RigidHand，RigidFullHand，RigidRoundHand，ThickRigidHand；RigidHand/Finger

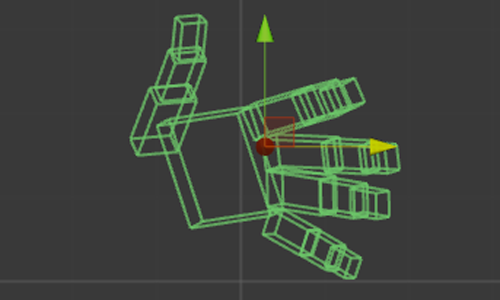


图3-1-4-5

将HandController添加到场景后，可以通过将所需要的手工预制件拖动到Hand Graphics Model插槽来设置图形模型。

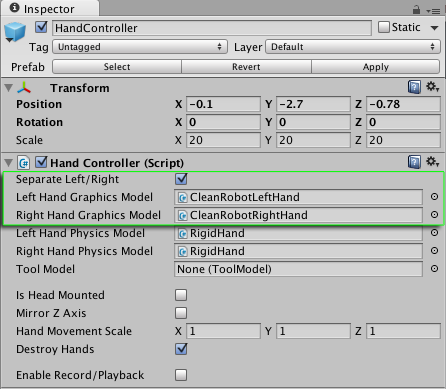


图3-1-4-6

如果选中了分离的左/右设置，则可以为左右手使用不同的预制件。否则，HandController使得双手为同一个镜像模型。现有的预制件在所有标有“右”或“左”的地方都有标注（如果使用错误的手型，错误将会很明显）。

想改变手模型的大小，需要修改HandController预制体的Transform Scale属性。Scale值为1的话表示为真实大小，但是通常太小，这取决于不同的场景。例如，默认的Unity立方体边长1米，这样会使手的尺寸看起来很小。将手变得更大也会增加它们的移动范围。通过增加HandController手部移动比例值，可以进一步增加动作范围而不会增加手部的大小。

在VR和AR的场景中，情况会有所不同。在实际情况下应该使用1:1比例。由于Leap Motion跟踪数据已经使用了现实世界的单位，因此将HandController缩放比例设置为1，并将HandController对象放置在相对于相机相同的位置，因为实际上设备是处于两眼之间的相对位置的。对于VR/AR应用程序，应该使用LeapMotionVR文件夹中的LMHeadMountedRig预制体替代HandController预制体，并且将这些物体进行自动调整。

设置Head Mounted将会提供Leap Motion服务的提示，即从Head Mounted位置查看手部模型并提高手部识别度。该设置不会自动旋转游戏界面中的手以显示手在HandController的前方而不是上方。为此，可以将HandController旋转至设置为{x=270，y=180，z=0}。

3.1.5 工具介绍（Widgets）

Leap Motion核心组件包括的UI控件包含以下几种：

·DemoDial – a scrolling list

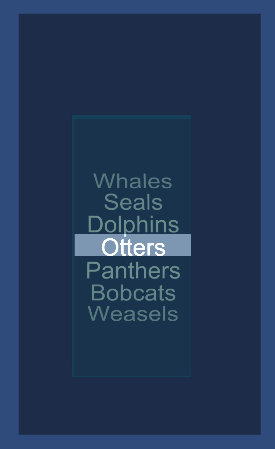


图3-1-5-1

·DemoScrollText – a scrolling text box

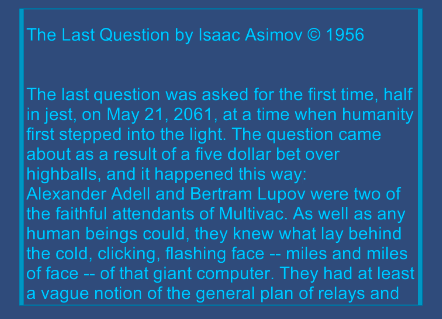


图3-1-5-2

·DemoSlider – a slider control

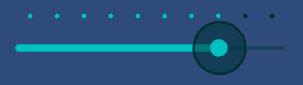


图3-1-5-3

·DemoToggkeButton – a two-state button



图3-1-5-4

以上这些小部件的预制件都可以在Core Assets中的LeapMotion/Widgets中的Prefabs文件夹中找到。

3.1.5.1 添加Widget到场景

要将小部件（Widget）添加到场景中，只需要将所需的预制件拖动到场景视图中，并根据需要来设置位置和缩放比例。放置和调整窗口小部件的主要考虑因素是，可以使用Leap Motion生成的手轻松地达到UI的功能部分，并且多个控件足够大并且间隔也足够大，可以方便地使用一个窗口小部件而不同意外触及另一个窗口。

需要注意的一个技术问题是，一些小部件在内部使用Unity Canvas对象。如果将这个小部件放置在另一个画布内，则可能必须手动使用缩放，因为Unity仅应用分配给最外面Canvas对象的缩放。

有两个场景演示Core Assets中包含的小部件。LeapMotion/Widgets中的场景展示了非VR布局。LeapMotion OVR/Widgets/Scenes中的场景展示了以VR头盔 Oculus Rift的VR布局

3.1.5.2 与对象属性挂钩的Widget

其中3个小部件，DemoDial，DemoSlider和DemoToggleButton分别表示的值为：拨号菜单选择，滑块数值以及按钮的切换状态。Widget的库定义了一个抽象的DataBinder类，必须实现该类才能将小部件挂接到场景中对象的属性。

DataBinder类是一个通用类型；为每个小部件定义一个特定的类的类型。这些在DataBinder.cs文件的末尾定义：

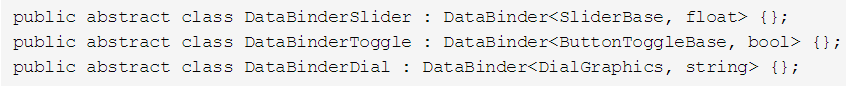


图3-1-5-2-1

在具体的类中，实现了两个函数：一个函数返回的值作为Widget的数据模型的属性；另一个更新该属性的值，并且在更改绑定空间的值时调用。例如，DataBinderSlider的实现会如下所示：

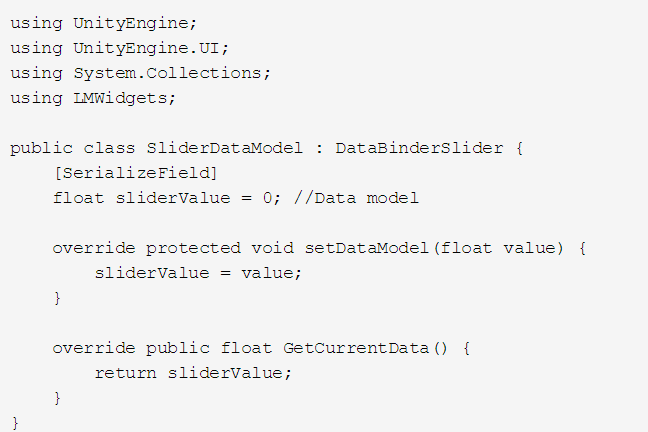


图3-1-5-2-2

本示例定义了数据模型属性本身（sliderValue），但其实可以使用某个场景的现场属性。事实上，数据模型根本不需要简单的属性。它只需要映射到小部件的数据类型。数据绑定脚本可以附加到任何更方便的对象，例如具有数据模型的属性的对象。

定义数据绑定类之后并将其添加对象后，必须将该小部件拖动到Unity的Inspector中对象的小部件属性。

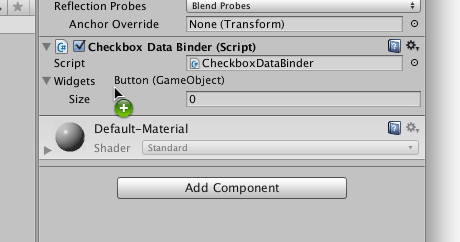


图3-1-5-2-3

3.1.5.3 与Events挂钩

如果对Widgets的交互事件有兴趣可以添加事件处理程序。Widget的库定义了以下句柄：

·StartHandler – buttons，sliders and dials

·EndHandler – buttons，sliders and dials

·ChangeHandler – sliders and dials

要添加事件处理程序，需要使用正确的签名定义事件处理程序函数，并为该Widget的事件句柄添加函数引用。事件处理程序的签名如下所示：



图3-1-5-3-1

其中T是与小部件关联的数据的类型。T对于切换按钮将是bool值变量，对于滑块是浮动类型变量，对于拨号是整形变量。

例如，可以定义并注册一个函数来处理滑块小部件的更改时间：

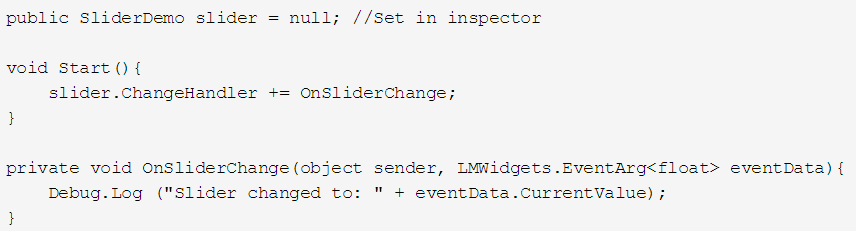


图3-1-5-3-2

3.1.6 C# API的引用

在这一部分介绍一下组成Leap Motion API的几种基本类的信息。

这些类中显示的任何坐标，方向和转换都是相对于Leap Motion坐标系而非Unity游戏世界的标准来的。要将位置向量转换为Unity坐标，需要使用Vector类中的扩展函数ToUnityScaled（）。要将方向向量转换为Unity坐标，需要使用ToUnity（）。这些脚本在Unity Core Assets中的LeapUnityExtensions中包含的C#脚本中定义。

同时，可以将表示为矩阵的变换从Leap Motion API转换为具有Matrix类扩展的Rotation（）的Unity四元数旋转和使用Translation（）的Vector3转换。

从Leap Motion中手的脚本获得矢量时，这些矢量相对于HandController对象已经被转换为Unity世界空间。换言之，如果在场景中，旋转控制器的对象，手的位置也会旋转。在条用ToUnityScaled（）后，可以使用Unity中的Transform.TransformPoint（）函数以相同的方式将点从Leap Motion坐标转换为Unity世界坐标。

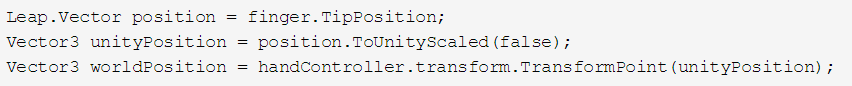


图3-1-6-1

开发者也用TransformDirection（）完成相同的功能。

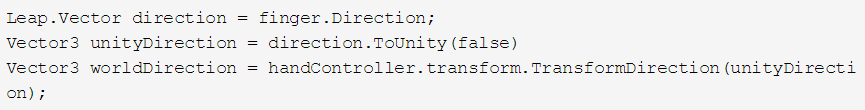


图3-1-6-2

要在场景中转换相对于控制器对象的旋转，需要将控制器的旋转值与对象的旋转值相乘。

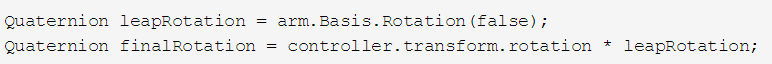


图3-1-6-3

3.1.1 SDK库

Leap Motion 库是用C++编写的。Leap Motion还使用SWIG（一种开源工具）为C#，Java，和Python生成语言绑定。SWIG生成的绑定将使绑定的编程语言编写的调用转换为基本C++ Leap Motion库中的调用。 每个SWIG绑定使用两个附加的库。对于JavaScript和Web应用程序开发，Leap Motion提供了一个WebSocket服务器和一个客户端JavaScript库。

除了Leap.js客户端JavaScript库外，Leap Motion SDK中还包含了开发支持Leap的应用程序和插件所需的所有库，代码和头文件。如果想使用这些文件，需要从Leap Motion Developer Portal下载Leap Motion SDK。每个支持的操作系统由一个SDK包。

Unity5和虚幻引擎4.9的插件与主SDK分开提供。Unreal插件包含在虚幻引擎4.9+。

完整的Unity资源包是单独提供的。SDK中不包含手型，Unity特定脚本和演示场景。通常情况下，下载和导入整个资源包比较容易，但是插件文件也包含在SDK lib/UnityAssets文件夹中。从Unity5开始，Pro和Personal都支持插件。

通常在Unity中，脚本语言使用的是C#。C#类定义是为独立库中的.NET 3.5和.NET 4.0提供的，使用代码可以引用LeapCSharp.NET3.5.dll或者LeapCSharp.NET4.0.dll（在所有支持的操作系统上使用相同的库名称）。这些库加载libCSharp.dylib（Mac），LeapCsharp.dll（Windows）或libLeapCSharp.so（Linux）。中间库加载libLeap.dylib, Leap.dll或libLeap.so。

3.1.2 创建项目

首先，以通常的方式创建一个新的Unity项目，即

·打开Unity编辑器

·选择File>New Project

·选择一个名称并保存位置

·点击保存项目

然后将Leap Motion资源包导入项目中。

接下来下载资源包，在下载完成之后，选择Unity中的Asset>Import Package>Custom Package menu command.找到下载的资源包，然后点击打开，资源包将被导入到项目中。如果要将资源包导入已包含Leap Motion资源的项目，建议先删除旧的资源。Unity导入过程指挥添加新文件并覆盖已修改的文件，它并不会删除过时的文件。

3.1.3 添加预制体

将手持控制器预制体放置与希望出现的点的下方。相对于真正的Leap Motion谁被，场景中的双手位于与真实双手相同的位置。并且可以通过更改HandController对象的缩放比例来更改手的大小，并更改“手部移动比例”设置来允许双手在较大的音量范围内移动。

通过将不同的预制件拖动到左/右图形模型插槽来更改手部图形。

接下来介绍为场景添加手势的基本步骤：

·在Project面板中，找到Assets/LeapMotionPrefabs中的HandController预制件

·将HandController预制件拖到场景视图中

·将预制件位置设置到所需要的坐标

·为了看到手，HandController必须位于摄像机的视野内

·设置手部比例尺以适当的数值，将手放入场景中，1的比例意味着双手呈现其实际尺寸

·将HandController的左右手图形模型属性设置为HandModerlsHuman或HandModelsNonHuman预制文件夹中所需要的预制体。（一般来说不需要从RigidHand中更改物理模型）

·将所有物体调试完毕后即可开始测试手的模型

当现实中的双手放在Leap Motion设备上时，可以看到双手出现在Hierarchy面板以及Game视图中。如果在Hierarchy中看不到手，可以在Leap Motion控制面板中打开展示台，确保Leap Motion设备已经连接并且发送跟踪数据。在Unity中可以暂停游戏界面并使用Scene视图来查看具体细节。

3.1.4 访问Leap Motion的API

除了资源包中包含的预制体和脚本以外，开发者还可以编写自己的脚本来访问Leap Motion API来追踪数据。Leap Motion类在Leap命名空间中定义。访问Leap Motion API的基本MonoBehavior类将如下所示：

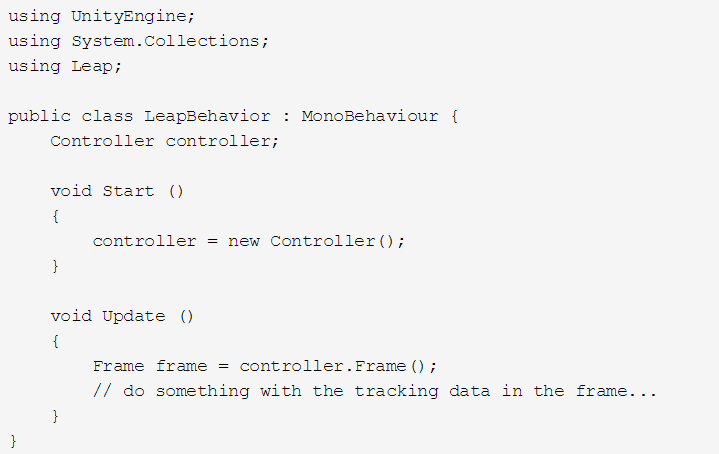


图3-1-4-1

需要注意的是，上图的示例不使用Leap.Listener子类。由于Unity应用程序具有自然的帧频，因此当Unity引擎调用函数时，Update（）函数可以从Leap Motion控制器获取当前数据帧。

Assets/LeapMotion/Scripts/Utils中的脚本LeapUnityExtensions,cs提供了将向量和矩阵从Leap Motion API类转化为Unity API类以及转换比例和坐标系的功能。

3.2 LeapMotion手势的形成

3.2.1 手固定不动

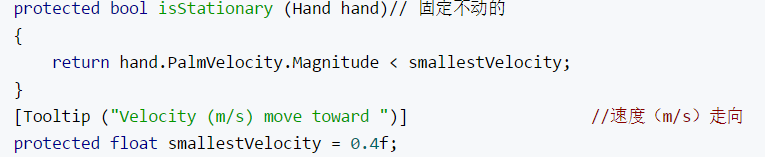


图3-2-1-1

上图为判定Leap Motion手固定不动的代码。图中PalmVelocity代表的是手部模型在X轴上移动的速度，算法如下，设定一个最小的X轴移动速度smallestVelocity，当手在水平面上的移动速度小于这个值时，认为手此时处于静止不动的情况，并返回一个bool值。

3.2.2 手划向左边

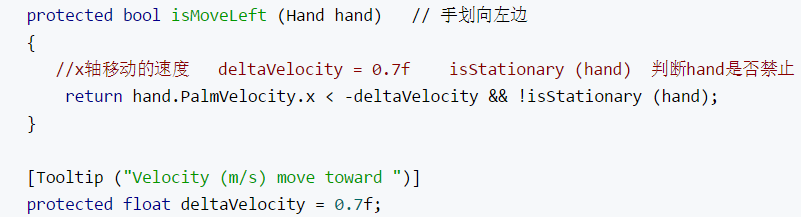


图3-2-2-1

上图为判定Leap Motion手向左划动的代码，算法如下，如果Hand在X轴上移动速度小于-deltaVelocity的值（向左为X轴负方向），并且同时isStationary函数的返回值为false，则视为手在向左划，并返回一个bool值。

3.2.3 手划向右边

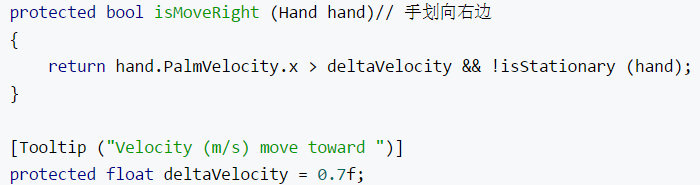


图3-2-3-1

上图为判定Leap Motion手向右划动的代码，算法和向左滑动类似，如果Hand在X轴上移动速度大于deltaVelocity的值（向右为X轴正方向），并且同时isStationary函数的返回值为false，则视为手在向右划，并返回一个bool值。

3.2.4 手划向上边

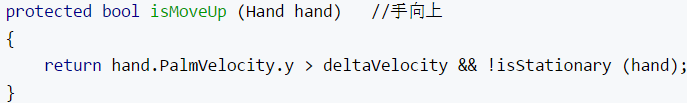


图3-2-4-1

上图为判定Leap Motion手向上滑动的代码，算法与之前相同，只是方向变化，如果Hand在Y轴上移动速度大于deltaVeclocity的值（向上为Y轴正方向），同时isStationary函数的返回值为false，则视为手在向上划，并返回一个bool值。

3.2.5 手划向下边

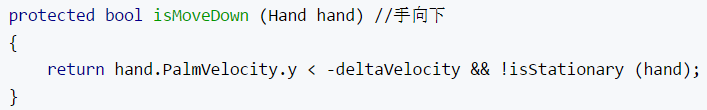


图3-2-5-1

上图为判定Leap Motion手向下滑动的代码，算法同上，不详细叙述了。

3.2.6 手是否抓取

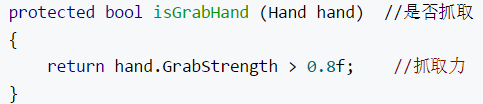


图3-2-6-1

Hand.GrabStrength是Leap Motion中自带的函数，函数返回一个手势的抓取力，通过帧与帧之间的不同来判断抓取力的大小，这里定义大于0.8手的状态为抓取，并且返回一个bool值。

这里还有另一种方法，通过hand.GrabAngle来判断是否抓取。GrabAngle值表示手指与手模型本身构成的角度。通过查看4个手指的方向之间的角度来计算一个新的角度。计算角度时不考虑拇指，对于张开的手，角度为0弧度，当姿势是握紧拳头时达到π弧度。这时可以通过定义一个合适的角度来判断手是否出去抓取状态。

3.2.7 手是否握拳



图3-2-7-1

上图为判断手是否握拳的代码，算法如下，创建一个List，获取当前手的所有手指信息，finger.TipPosition是当前手指指尖的坐标，每个手的指尖坐标减去hand.PalmPosition（手掌掌心的坐标）得到的向量的长度如果小于设定值，则视为当前手处于握拳状态，并返回一个bool值。

在上一个是否抓取的章节中提到了hand.GrabAngle，更为简单的方法是直接判断hand.GrabAngle的值是否为π，即可判断是否握拳。

3.2.8 手是否完全张开

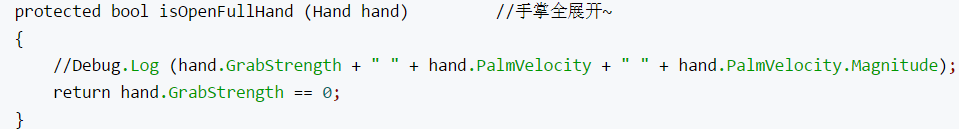
****

图3-2-8-1

上图为判断手是否完全张开的代码，算法十分简单，直接判断手的GrabStrenth值是否为0，为0则是完全张开并返回一个bool值。除了此方法外，也可以判断hand.GrabAngle的值是否为0.

3.2.9 向量的判断

·向量转化为角度

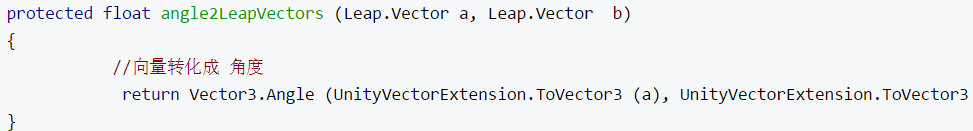


图3-2-9-1

这段代码的作用是将Leap Motion中的两个向量转化为Unity中的角度。

·判断两个向量是否相同方向

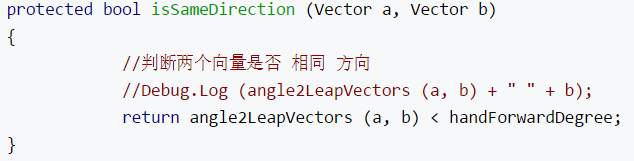


图3-2-9-2

设定了一个浮点数handForwardDegree为30F，默认两个向量a，b之间的夹角如果小于30则判定它们为统一方向。

·判断两个向量是否相反方向

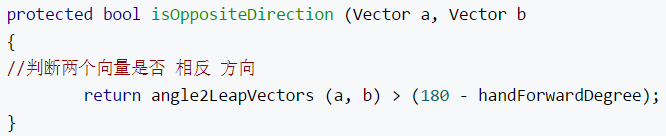


图3-2-9-3

方法与上面相同，只是判定的角度变为了180减去设定的handForwardDegree值。

·判断手掌心的向量方向是否与另一个向量相同

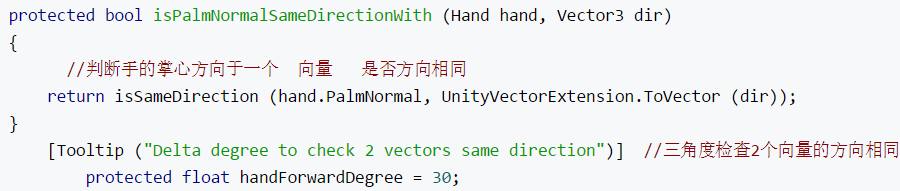


图3-2-9-4

算法十分简单，通过之前写的isSameDirection函数，函数传参的值变为hand.PalmNormal的值和另外一个向量的值即可。

3.2.10 向手掌的方向移动

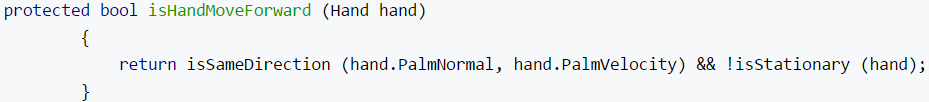


图3-2-10-1

上图为判断手是否在向掌心向量的方向移动。具体算法如下，用isSameDirection判断掌心的向量方向和手掌移动的向量方向是否一致，如果一致并且此时手不处于固定状态，则手在向掌心方向移动，并返回一个bool值。

3.2.11 手掌是否垂直（掌心水平）

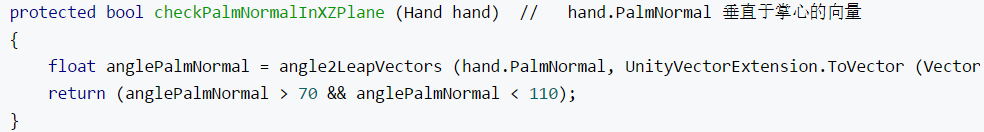


图3-2-11-1

上图为判断手掌是否垂直的代码。具体算法如下，用angle2LeapVectors判断hand.PalmNormal（垂直于掌心的向量）和vector.up（垂直向上的向量）之间的夹角，当hand.PalmNormal与竖直向量成90度是，即为手掌水质。但由于在实际操作中不会完全水平，所以给予一个误差上下20度，即两个向量夹角的值anglePalmNormal在70度到110度之间都认为是手掌水平。

3.2.12 判断大拇指是否竖直或向下



图3-2-12-1

从上图的代码中可以看到，用一个List将手指的信息全部储存下来，并只对里面有TYPE\_THUMB（大拇指）标记的手指进行处理，采用了两个值，一个finger.direction，手指的方向与数值方向判断，另一个是手指的坐标减去掌心的坐标形成向量进行判断。当这两个值与竖直方向的夹角小鱼设定的deltaAngleThumb时，则可以判断大拇指处于竖直或者向下的状态，并返回一个bool值。

3.2.13 判断四指是否靠拢掌心

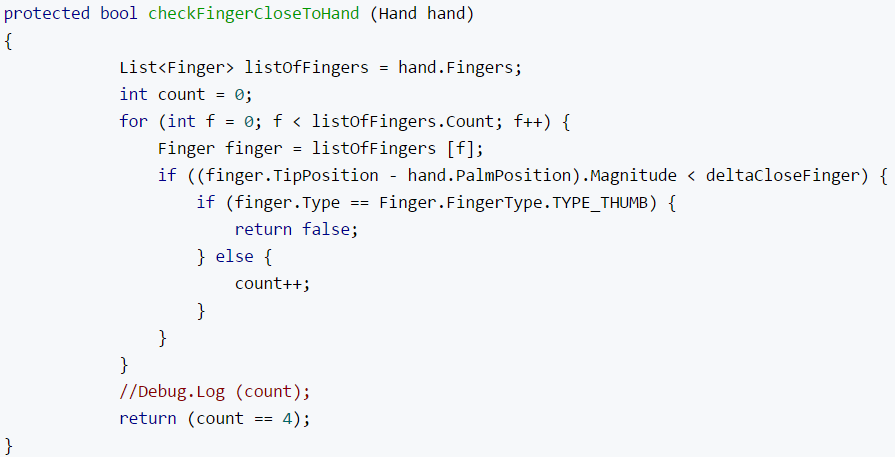


图3-2-13-1

上图为具体代码，具体算法如下，使用一个List存储手的手指信息，分别遍历List中除了大拇指的手指信息。当finger.TipPosition（手指的指尖位置）与掌心位置相减得到的向量的大小小于设定的deltaCloseFinger值后，即可认为手指此时靠拢掌心，并且返回一个bool值。

第四章 LeapMotion与Unity的结合实例

4.1 游戏介绍

本次的Unity游戏类型选择了跑酷游戏，与以往在电脑上通过键盘或者手机上通过屏幕滑动人物移动操控不同，这次加入了Leap Motion手势控制器，玩家可以通过手势的挥动直接控制游戏中人物的移动，使玩家获得不同的游戏体验。

游戏的基本玩法是玩家控制的角色在一个无限循环的道路上奔跑，玩家通过手势控制角色躲避不同的障碍的同时获得更多的金币以增加更多的分数。在奔跑的同时可以通过获得不同的道具来得到不同的效果，以此增加游戏的趣味性和挑战性。

4.1.1 人物介绍

下图是游戏里的角色



图4-1-1-1

该角色拥有几个不同的状态，分别是奔跑，左右移动，跳跃翻滚以及死亡状态。

奔跑：

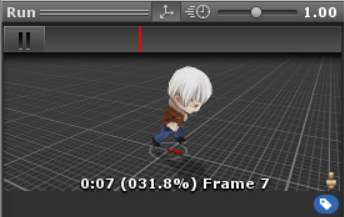


图4-1-1-2

左右移动：

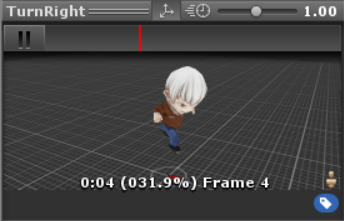


图4-1-1-2

跳跃翻滚：

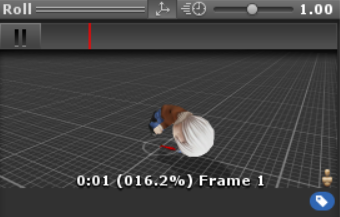


图4-1-1-3

死亡状态：

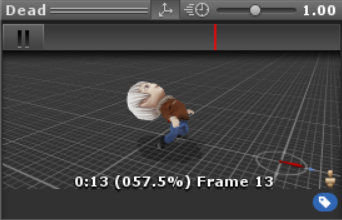


图4-1-1-4

以上的几种状态构成了角色的移动控制。

4.1.2 道具介绍

游戏中一共有5种道具，金币和4个状态道具。



图4-1-2-1

金币在游戏中旋转的，目的是在于使玩家控制的角色获取更多的分数。

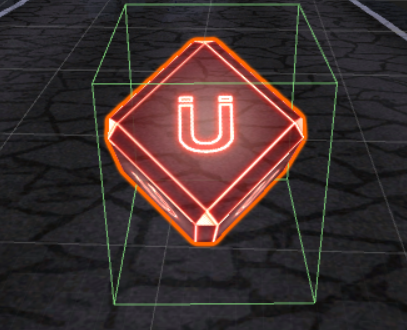


图4-1-2-2

道具Magnet（磁铁），作用是吸引身边一定范围内的所有金币，获得分数。



图4-1-2-3

上图这个跑鞋道具可以是游戏角色获得双连跳效果，即在空中可以进行再一次地跳跃，方便玩家通过一些平常跳跃无法越过的障碍。



图4-1-2-4

上图是双倍积分，效果是将一定时间内，获得的金币数乘以二，换言之就是获得双倍的分数。



图4-1-2-5

这是星型道具，效果使角色的移动速度加倍，可以使得游戏获得金币的过程变快，但同时也会增加游戏失败的概率，这个道具是一把双刃剑。

4.2 游戏实现

4.2.1 移动功能的实现

游戏里的人物的移动需要由Leap Motion控制器来实现，具体手势是通过上文设定的代码来确定。具体实现代码如下图所示：



图4-2-1-1

可以看见代码中有一个名为GetInputDirection的函数，这个函数的功能是判断此时人物所处的状态，在InputDirection中一共有5个状态，分别为null, left, right, down, up。代表上下左右。判定方法为，获取当前帧的手部信息，再用foreach函数遍历所有的hand的信息。这时候用if语句来判断，当前手的信息是否满足上下左右的条件，比如isMoveLeft(hand)，判断手是否向左划。如果满足条件，播放滑动的音效并且将状态该为移动方向的状态。

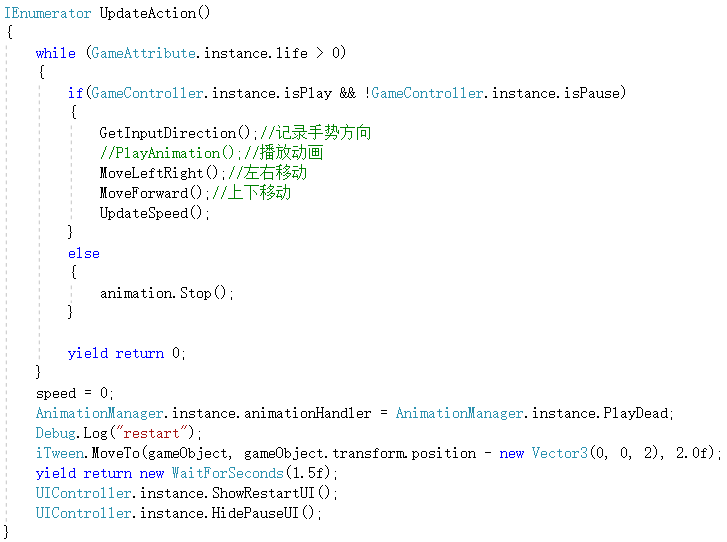


图4-2-1-2

上图是C#中的协程，控制动作的更新，如图可知，在玩家生命值大于0时，可以进行游戏动作判定。如上图所示，记录手势方向之后，会有左右移动和上下移动的函数分别调用，如果满足条件则会播放左右移动或者上下跳跃翻滚的动画。

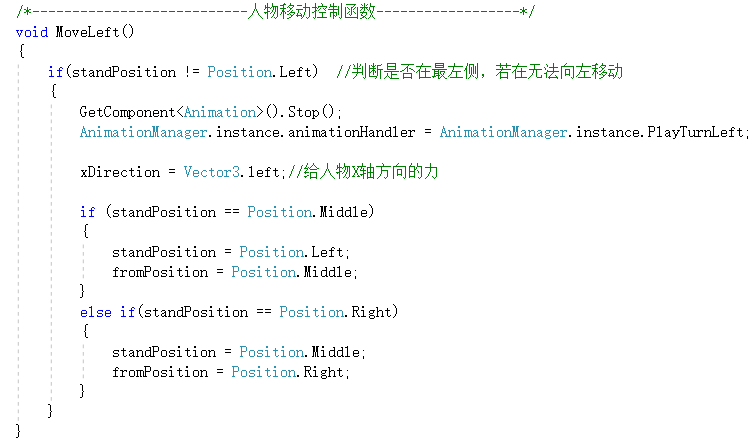


图4-2-1-3

上图是向左划动的控制代码，类似于神庙逃亡的三条道路，所以在最左边不能再往左划。采取的方法是当状态变成向left时，给人物一个向左的力，对应上时X轴反向的力，同时播放划动的动画，并且记录此时的位置和变化之前的位置。

向右划动的控制代码与此类似，所以就不再重复了。同时代码中还有一个MoveLeftRight函数，具体函数代码部分如下：

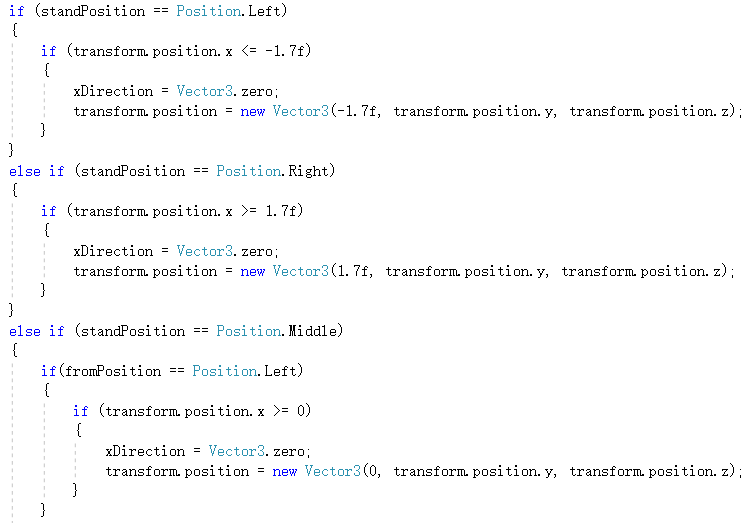


图4-2-1-4

主要功能是统一向左向右移动的函数并且在人物移动到指定位置后停止人物收到的X轴方向的力，避免人物一直移动出道路外面。

接下来的函数时MoveForward，函数的代码如下：

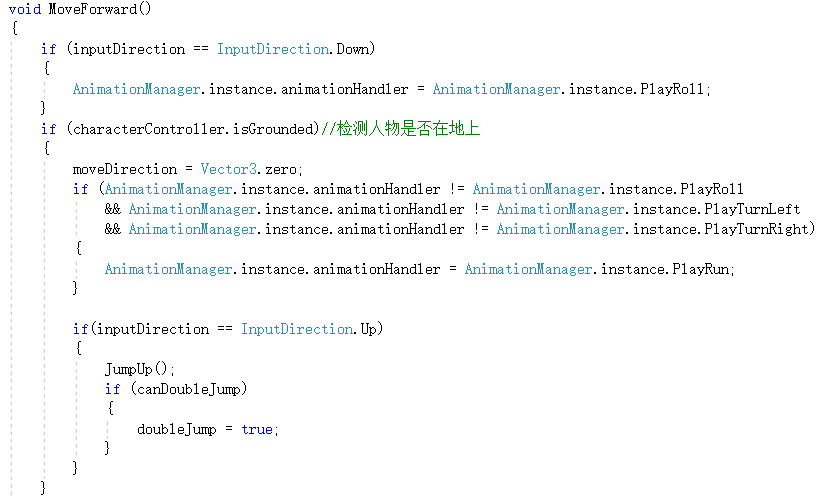


图4-2-1-5

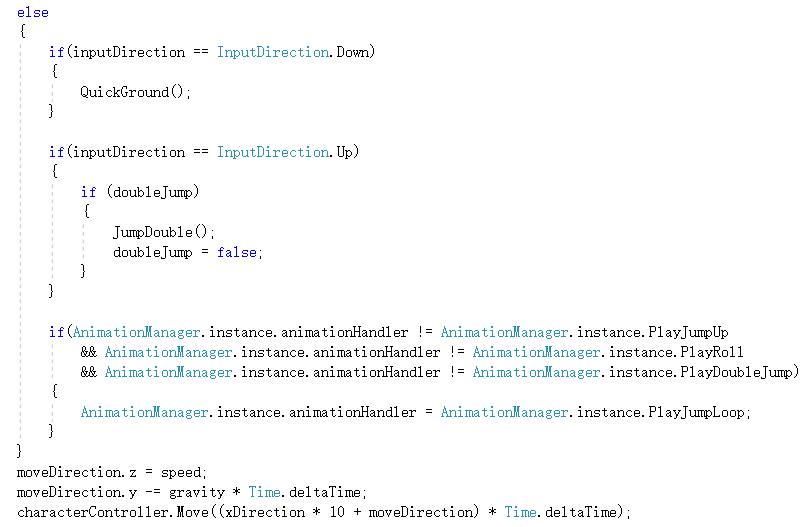


图4-2-1-6

这个函数的逻辑分为两层，第一是判断人物是否在地上，如果isGround为True，则再判断是状态是up还是down，并播放相应的动画。如果不再地上，则判断是否还能连续跳跃和翻滚直接落到地面上。在空中会有几个阶段，起跳，空中滑翔和落地，这几个时刻的动画都是不同的，所以在函数中都会做判断来确定人物当时的状态。

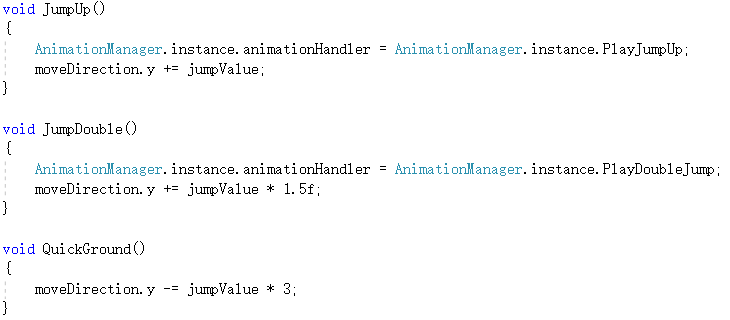


图4-2-1-7

具体的跳跃函数如图所示，基本原理都是给人物Y轴方向的力，形成一个跳跃的效果。

4.2.2 道具功能的实现

所有可以碰撞的道具都继承一个自定义的Item类，类的代码具体如下：

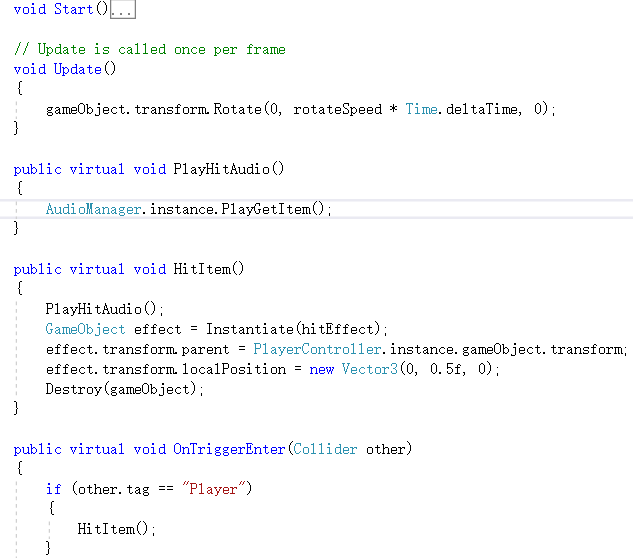


图4-2-2-1

这个Item类中定义了道具的基本属性，rotate旋转，碰撞时候播放相对应的音效，还有碰撞时候的效果。由于每个道具的音效和碰撞效果不一定一样，所以可以从上面代码发现这两个函数定义为virtual函数，使得继承该类的子类可以复写该函数。

4.2.2.1 金币

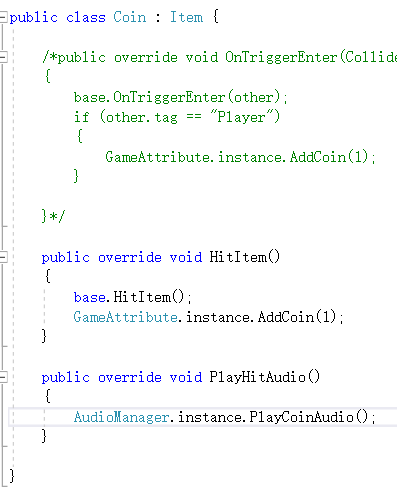


图4-2-2-1-1

金币效果比较简单，复写的函数中改写了碰撞时候的播放音效和分数加一的效果。

4.2.2.2 Magnet（磁铁道具）

Magnet的相关类有两个，一个是继承Item的Magnet类，复写了碰撞音效和碰撞效果。另外一个则是实现磁铁的功能，吸引一定范围内的金币。具体部分代码如下：



图4-2-2-2-1

具体实现方法是，给玩家控制的人物增加一个范围确定的圆形碰撞体，当金币碰撞到这个碰撞体后就会向人物移动，碰到人物时发生碰撞事件。在上图的代码中，可以看到使用了Vector3.Lerp插值函数，将金币平滑地移动到人物身边。

4.2.2.3 Shoe（跑鞋道具）

跑鞋道具代码上前面的道具类似，不同之处在与会是PlayController这个类中的canDoubleJump值从false变为true，使得人物可以在空中进行二次跳跃。

4.2.2.4 Multiply（双倍积分）

双倍积分效果是将获得金币的倍数改为2，并且同时有时间的判定，一定时间后结束双倍积分的效果。

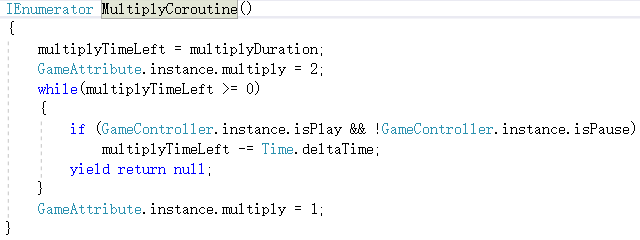


图4-2-2-4-1

4.2.2.5 Star（暴走鞋）

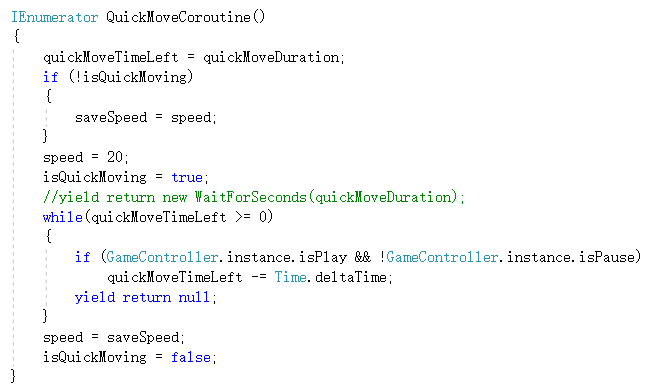


图4-2-2-5-1

在这个代码中有一个计时的问题，在测试时，发现如果在效果暴走效果时间结束之前再次获得了加速效果，会导致效果持续时间的计时发生问题，时间减少变快。这是由于没有用协程的关系，同时接受到两个效果后，Time.deltaTime的时间减少变快。使用协程（Coroutine）后就能有效地解决这个问题。

4.2.3 场景的设计与完善

4.2.3.1 道路系统的设计

这个由Leap Motion手势控制的跑酷游戏是无限模式的，即如果人物不死亡，可以一直跑下去，道路上的障碍物和各种道具也是随机生成的。这里需要介绍的就是如何使道路无限延长。

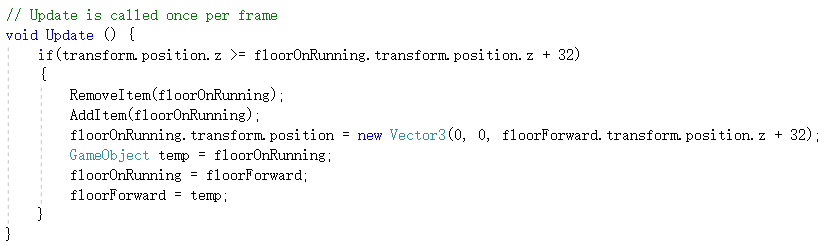


图4-2-3-1-1

上图为具体代码，基本的算法是当人物跑到了当前道路的起点时，将上一个道路上的东西移除，并添加新的障碍物道具后，改变之前道路的位置，放在当前道路的结尾，如此循环反复，使得路面无限延长。这个算法的要点是需要将floorOnRunning和floorForward区分清楚，这样才能顺利地进行道路的替换和移动。

接下来要介绍的是道路上的障碍物和道具的随机生成。



图4-2-3-1-2

上图所示的代码是完成了一个添加物体的功能。即在设计道路时，完成了一个场景的设计，可以通过如上代码写出来的功能点击AddPatternToSystem按钮完成对GameAttrinbute中Pattern的添加，这样能保存所有添加的场景，并且在之后的道路功能实现中随机地添加在新生成的道路上。

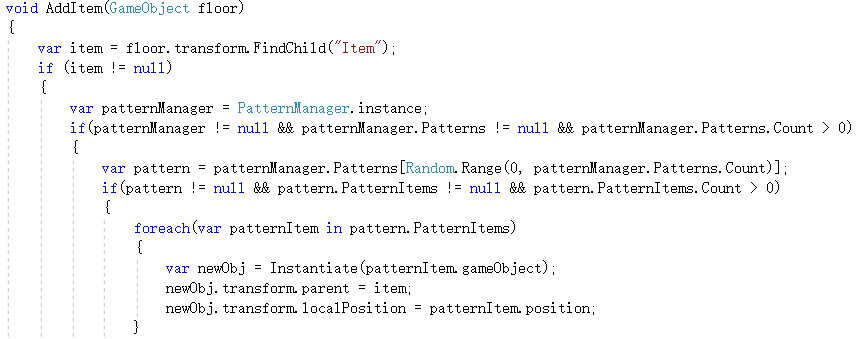


图4-2-3-1-3

上图是AddItem的具体代码，在生成新的道路时会执行上面的函数。Pattern里面的物体会记录类型和位置，所以在添加的时候会完全按照当时设计道路时的摆放位置添加。

4.2.3.2 障碍物与背景的实现

障碍物与背景的房屋是手动放置上去的，同时为了完成游戏功能的实现，需要自己添加碰撞体，并且区分致死碰撞体和正常的碰撞体（只是不让人物通过并不会减少人物生命值）。

背景方面需要注意的是贴图颜色的调配和亮度的选择，因为使用的是Curve的贴图渲染方式，会有一定的弯曲折射效果，亮度选择过高后会发生一定的透视效果，影响游戏体验。其次就是方向的放置，在Unity中可以较为便捷地实现这些背景的放置。

下面给出障碍物类的部分具体代码，所有游戏中的障碍物均继承与该类。

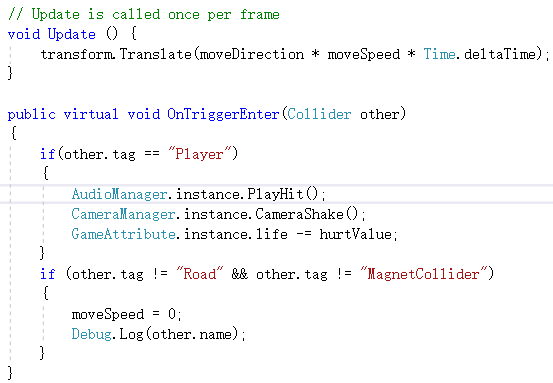


图4-2-3-2-1

图中给出了障碍物的一些功能实现方法，碰撞到玩家后，会触发碰撞音效和相机抖动，也会将玩家的生命值减1.

在其中值得一提的是，在障碍物Board（栅栏）的判定方法，障碍物图和碰撞体积如下：

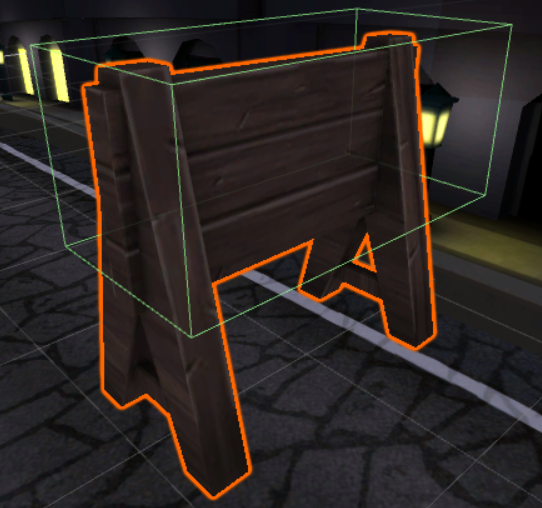


图4-2-3-2-2

人物在经过这个障碍物时如果不向下划动手使角色产生翻滚的动画，就会碰到栅栏的上方碰撞体，从而结束游戏。而为了实现这个功能，代码如下：

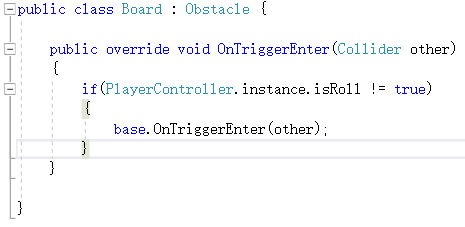


图4-2-3-2-3

实现的方法如上所示，即当玩家的isRoll状态不为真的时候，出发碰撞体效果。

4.2.3.3 UI界面设计