**广西大学行健文理学院毕业论文**

**基于Unity3D制作第三人称射击游戏的设计与实现**

学 部：理工学部

专 业：计算机技术与科学

学 生 姓 名：杨启玢

学 号：1738940341

指 导 教 师：谢聪

评 阅 教 师：

完 成 日 期：

广西大学行健文理学院

# 摘 要

随着最近几年游戏行业的蓬勃发展，诞生了许多非常优秀的游戏引擎。Unity3D,Unreal,Cocos,寒霜，这些引擎的但是使得制作游戏的难度大大降低。以往游戏的制作难度是十分耗费财力和精力的，高额的代码成本使得游戏公司并不愿意将自己的技术与其他的游戏公司分享。且游戏涉及的技术领域比较广，物理学，计算机科学，计算机图形学，美术，营销，策划。使得游戏公司并不能完全的将精力放在研究技术上。于是针对这一痛点，就有了专门研发的游戏引擎。

网络游戏的另一个重点是后端游戏服务器。游戏行业不像web行业有非常成熟的后端框架，这里的成熟是指有完备的文档，并且得到了广泛的验证。这里我选择了前网易游戏核心成员，杭州研究中心的总监云风编写的”skynet”后端框架。但是skynet在游戏行业只得到了广泛的验证，但是其文档并不完备。

既然是网络游戏，网络协议也将会成为一个需要关注的重点，这里使用google开发的protobuf序列化协议工具。这款工具能够方便的将收发的数据序列化以及反序列化，以方便的进行网络传输。

本文将结合Unity3D和skynet,google-protobuf研究一款第三人称网络游戏的设计以及实现的历程。本文将会涉及Unity3D的物理引擎系统，animator动画系统，C#网络通信，skynet socket通信，网络游戏同步算法，反向重力学，人物控制。

**关键词：unity3D;网络通信；网络游戏同步算法;人物控制**

目 录

[摘 要 II](#_Toc1125897854)

[引 言 1](#_Toc1032382504)

[1 Unity3D引擎介绍 2](#_Toc826598855)

[1.1 Unity3D引擎简介 2](#_Toc1681402402)

[1.2 Unity3D的摄像机 2](#_Toc83770946)

[1.3 Unity3D的动画系统(animator) 3](#_Toc1859647500)

[（1）状态机 3](#_Toc738621282)

[（2）1D混合树(blend tree) 3](#_Toc50331639)

[（3）2D混合树 4](#_Toc5890092)

[1.4 骨骼系统 5](#_Toc1453374184)

[1.4.1 Unity骨骼系统 5](#_Toc1471584868)

[1.4.2 反向动力骨骼 6](#_Toc488861973)

[1.5 人物移动 6](#_Toc392105830)

[1.5.1 Unity3D的坐标系 7](#_Toc320066200)

[1.5.2 用CharactorController控制移动 7](#_Toc1736288530)

[1.5.3 人物转向CharactorController.rotate() 7](#_Toc2066461493)

[1.5.4 碰撞体和落地检测 8](#_Toc1173222763)

[2 Skynet 介绍 8](#_Toc372897127)

[2.1 Skynet简介 8](#_Toc531655265)

[2.2 Actor模型 9](#_Toc1214075181)

[2.3 Skynet内部原理 10](#_Toc1296128241)

[2.4 Skynet 消息请求api () 11](#_Toc375006523)

[2.4.1 Skynet 服务的消息组成 11](#_Toc1609940414)

[2.4.2 Skynet 的分发与回应, 推送，调用 11](#_Toc2618787)

[1. 分发 11](#_Toc2128539988)

[2. 回应 11](#_Toc1728077160)

[3. 推送 12](#_Toc1630506407)

[3 第三人称设计游戏Unity3D前端 12](#_Toc425523458)

[3.1 人物的动画 12](#_Toc1817895360)

[3.1.1 人物的移动动画混合设计综述 12](#_Toc112447468)

[3.1.2 animator 状态机的设计 13](#_Toc113981865)

[3.1.3 blend tree设计 15](#_Toc796309567)

[3.2 视角变换 19](#_Toc1144829972)

[3.2.1 水平变换 19](#_Toc940580721)

[3.2.2 垂直变换 19](#_Toc330228321)

[3.3 准星瞄准 19](#_Toc1228600918)

[3.3.1 准星带动身体 20](#_Toc652744573)

[结 论 23](#_Toc1068849604)

[参考文献 24](#_Toc1278932557)

[致 谢 25](#_Toc658634665)

# 引 言

网络游戏客户端技术的难点是，怎样制作出符合用户直觉的交互逻辑，如何让玩家在游玩的时候不会产生严重的割裂感。如何使用animator去控制动画，让动画的过度更加的自然。

网络游戏服务端的难点是，如何让服务器能够稳定的运行，尽可能的压榨服务器的性能，设计出优秀的后端服务架构。后端服务器的后续拓展，配合客户端实现业务逻辑都成为了游戏服务端的难点。

网络游戏的另一难点，玩家与玩家的同步问题。射击游戏对网络的延迟容忍度是非常低的，但网络的延迟的变化因素非常的多，且难以控制。且网络游戏受一个人类无法突破的客观物理因素的影响——光速。光的速度是一秒钟绕地球7周半，在最理想的情况下，从地球的一段到另一端一个报文来回传输一次所需的时间也就是绕地球一周的时间也会约等于1/7秒，而我们现如今使用的最快的网络传输介质就是光。这一物理上的客观因素我们是无法突破的，更何况报文在进行路由中转的时候也是需要消耗时间的，这会带来的更大的网络延迟。 这对网络游戏的同步算法提出了更高的要求，如何设计出能够掩盖延迟，给玩家带来更好的体验的网络同步算法是客户端与服务端必须站在一起共同讨论的问题。

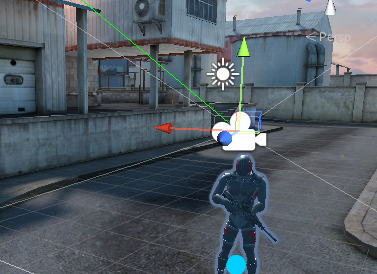
# 1 Unity3D引擎介绍

1.1 Unity3D引擎简介

Unity3D引擎的应用不仅仅在游戏行业，它还被广泛的应用在汽车、运输、制造、影视动画等等领域。其本质是一个用来虚拟世界的工具，通过操作3d模型，编程，和自带的3d物理引擎技术去虚拟一个世界。而游戏，正是看中了Unity3D构建虚拟世界的能力。Unity3D引擎必须与3D建模软件（3dmax,blender等）配合工作，才能完成虚拟3D世界的工作。其主要能力还是将各种资源整合拼装。3D建模软件，音乐制作软件负责为Unity3D提供素材，而Unity3D负责搭积木，将这些素材结合，处理素材与素材，素材与人之间的交互。在后面的有关Unity3D的篇幅中，请读者将自己想象成Unity3D虚拟的世界中的人物，这将有助于你去理解。

1.2 Unity3D的摄像机

Unity3D的摄像机就像是人的眼睛，是用来观察虚拟世界的窗口。



在这里我们将摄像机放在了人物的背后，这张图是我们站在人物的视图(注意，这张图并不是人物的背后，是他的前视图)。仔细观察，会发现这张图片上有白色的虚线。这些虚线是Unity用来提示当前摄像机的所能看到的视野范围。那这个摄像机究竟能看到什么呢？ 看下面这张图。



将摄像机想象成自己的眼睛，那么这就是你当前所能够看到的东西。因为我们将摄像机放在了人物的背后，所以我们可以看见这个人物的后脑。

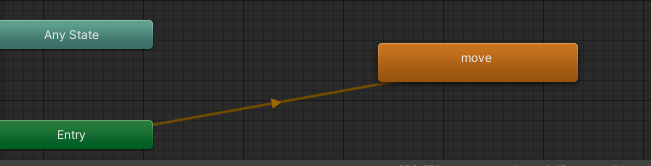
当游戏开始时，这个摄像机所拍摄到的东西将会显示在玩家的屏幕上。

1.3 Unity3D的动画系统(animator)

animator主要的作用是将3D模型身上的动画进行混合，播放。而模型的动画是建模时制作的，Unity3D并不负责制作动画。

（1）状态机

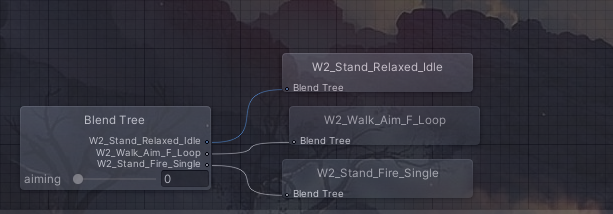
状态机是用来控制人物的状态的，他可以实现人物状态的切换控制。比如说我们可以使用状态机控制人物的站立和奔跑状态切换。



（2）1D混合树(blend tree)

但是，状态机对于人物动画状态的切换的效果并不好，动画与动画之间切换时，会有非常不平滑的割裂感，这个时候我们需要混合树。

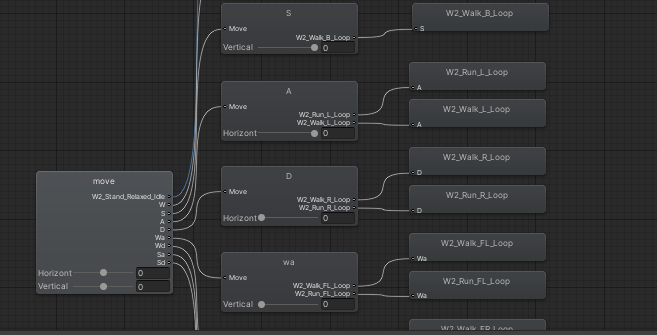
混合树是将多个动画按快慢排列起来，然后使用一个float型的参数去控制这些动画。由于float的变化范围不再是bool值的两种状态的变换，而是连续的值的变化。所以使用混合树去控制动画相比状态机会更加的平滑。



aiming是用来控制W2\_Stand\_Relaxed\_idle,W2\_Walk\_Aim\_F\_Loop,W2\_Stand\_Fire\_Single三种状态的float型参数，当aiming变化到一定的数值后，将会自动切换到不同的状态。

（3）2D混合树

2D混合树相较于1D混合树来说，会更加的复杂，但是能够实现更加复杂的功能。后面我会介绍我使用2D混合树制作的用于人物移动控制的混合树。

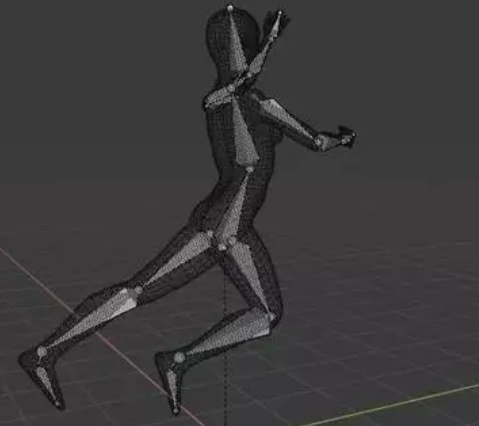


可以看到相较1D混合树，2D混合树的参数从1个变成了两个(Horizontal,Vertial)。

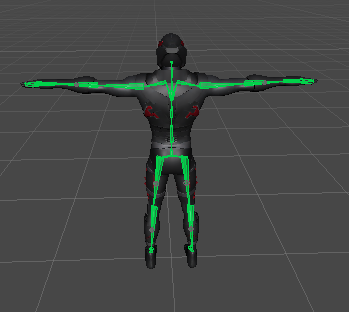
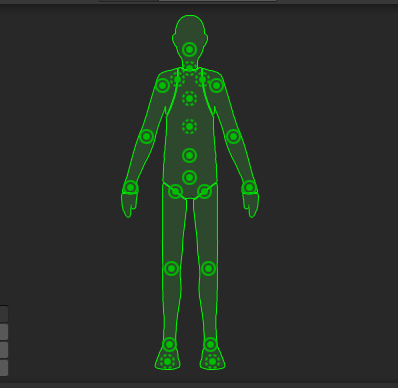
1.4 骨骼系统

1.4.1 Unity骨骼系统

骨骼系统并不是Unity3D系统自带的概念，骨骼其实是3D建模里面的概念。通过给3D模型绑定骨骼，我们就可以实现用移动骨骼的方法去使模型做动作。3D建模的骨骼系统其实就是模拟人体的骨骼系统，方便模型能够做出人的动作。



骨骼动作示意。



Unity中的骨骼点。

1.4.2 反向动力骨骼

反向动力骨骼是骨骼系统的一个非常重要的概念，是实现更加平滑的动画运动的关键。

先说正向动力骨骼，正向动力骨骼的动力是物体自身的动力。就像我们向做一个抬手的动作，这个抬手的力是我们自己产生的。

反向动力骨骼，反向动力骨骼正好相反，力是由外界产生的。这就像是有一个人，他把我们的收给拉了起来，同样手臂移动到了相同的位置，但是正向动力是我们自己抬起来的，而反向动力骨骼是由于外界某个人把手臂拉了起来。

由于模型是骨骼绑定的，当我们移动一块骨骼的时候，其余与之相连的骨骼也会跟着移动。这样，就形成了牵一发而动全身的效果。其具体应用，后面会有说，这里先跳过。

1.5 人物移动

1.5.1 Unity3D的坐标系

Unity3D坐标系与传统数学的三维坐标系的相同之处是都有三个轴向，但是与传统3D数学不同的是，Unity的Y轴是高度轴，而Z轴才是传统数学里的Y轴。这个概念在控制人物移动的时候，非常重要。



1.5.2 用CharactorController控制移动

CharactorController提供了move(x,y,z)这个方法来控制角色的移动，如果只是想在地面上移动的话只需要控制X轴和Z轴即可。

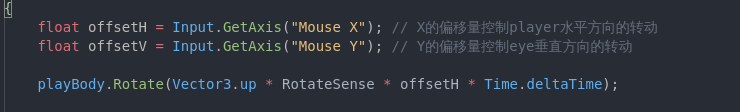
CharactorController 还提供了SimpleMove(x,y,z)这个方法会自动计算重力，这里我没有使用CharactorController控制角色移动所以不赘述。

1.5.3 人物转向CharactorController.rotate()

通过Unity3D的三元数可以控制人物的转动轴向。

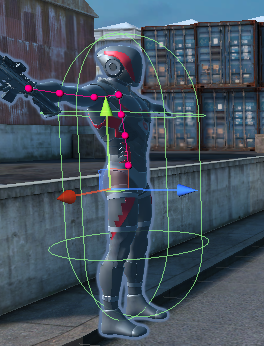
Vector3.up表示以Y轴为转动轴进行转动，这样就可以很方便的实现人物的水平方向的转动。也就是现实世界中的转身动作。

但是现实世界中还有抬头和低头，这里我们先不说，放在后面介绍。



1.5.4 碰撞体和落地检测

Unity3D的碰撞体是用来实现物体的碰撞检测的，而落地检测是通过物体与地面是否接触来实现的。碰撞体就像一个套在人物身上的网状的罩子，当有别的物体接触到这个罩子的时候，系统就会自动检测到碰撞事件。



绿色的虚线环绕起来的地方就是这个人物的碰撞检测范围。碰撞体还可以用来检测人物是否被子弹击中，不过图片上的碰撞体不够精细。精细的人物碰撞体是身体各个部位都会有一个类似的碰撞体，他们合起来组成了整个人体的碰撞体。这里只是简单的给人物套了一个大的碰撞体。

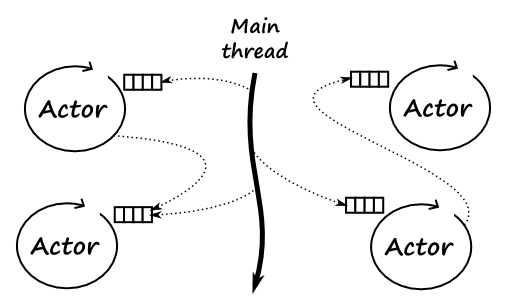
# 2 Skynet 介绍

2.1 Skynet简介

Skynet是一个广泛应用的服务端架构，他使用C编写了核心代码并暴露给了lua语言去做逻辑。实现了Actor。本质是一个消息转发服务架构，并且已经被用在了别的领域（比如web）。自带集群的功能。其实现的Actor模型理念可以非常好的帮助用户实现业务模块的划分

2.2 Actor模型

既然Skynet实现了Actor模型，那我们不可避免的要先去了解一下什么是Actor模型。

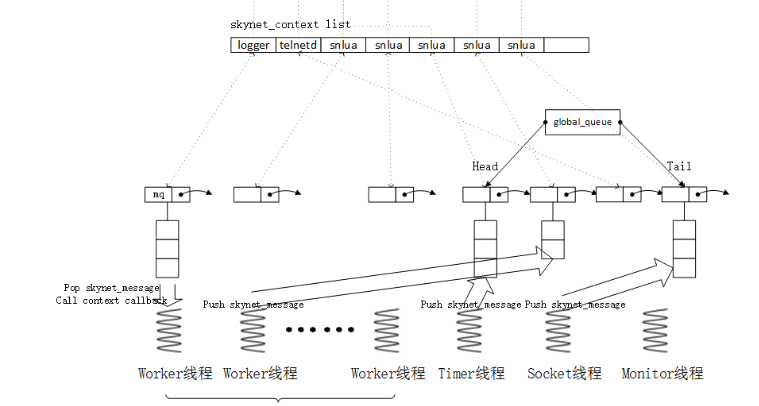
Actor可以想象成一个执行者，Actor的状态不能被别人直接读取或者修改，方法也不可以直接调用。Actor只能通过消息传递的方式与外界进行通信。传统的编程模型是用一个对象去直接操纵另一个对象，但是在Actor模型中这样的操作是不允许的，Actor是强隔离的。所有的操作必须由一个Actor去请求另一个Actor。

每一个Actor身上都会有一个mailBox（消息队列）,这些消息必须严格按照队列的顺序执行，也就是说，一次只能同步处理一个消息。处理消息过程中除了可以接受消息，不可以做任何其他的操作。由于每次只能处理一条消息，所以Actor内部可以安全的处理状态，不用考虑锁的机制。

Actor由于是请求，回应/不回应的模型。所以并不需要关心这个Actor是在本地还是在其他的节点，这可以帮助我们构建一个非常庞大的系统。

2.3 Skynet内部原理

Skynet中我们用service这个概念表达某项具体的业务，包括处理业务的逻辑以及关联数据的状态。service也就是上面提到的Actor。skynet使用lua虚拟机来实现各个service之间的隔离。并且skynet在启动是会开启n条线程去轮询监听消息队列。某个service的消息队列不为空的时候，skynet会把他的消息队列拿出来进行处理。



2.4 Skynet 消息请求api ()

每个服务的职责是处理其他服务发送过来的消息，或者向其他服务发送消息。

2.4.1 Skynet 服务的消息组成

session: 消息的唯一标识，当回应时，要将session带回（skynet.ret）

source: 消息源

type: 消息的类别 这里只需要关注lua类型的消息

message: 你要发送的消息

2.4.2 Skynet 的分发与回应, 推送，调用

1. 分发
   1. skynet.dispatch(type,function(session,source,cmd,...)): 注册特定消息的处理函数cmd是对应的操作。
2. 回应
   1. skynet.ret(msg,size): 这个api会将message size对应的消息附上当前消息的session
3. 推送
   1. skynet.send(address,typename,cmd,...): 这个API可以把typename类型的消息发送给address。这个API是非阻塞的
   2. skynet.call(address,typename,cmd,...): 这个API会生成唯一的session, 并向address 提起请求，等待对session的回应。这个消息是阻塞的

可能这就有点蒙了，其总结一句话，skynet消息必须先在dispatch里注册然后才能使用call，send进行调用，至于session 比较底层，可以先不去管他。

# 3 第三人称设计游戏Unity3D前端

3.1 人物的动画

3.1.1 人物的移动动画混合设计综述

上面有提到过，人物的移动其实有多种方式去实现。我目前知道的有

1. 使用CharactorControll 的move/simple方法
2. 使用物体身上的transform组件
3. 使用动画自带的移动方式

这里说明一下三者区别，

1. 在3D建模时，有的动画是带有移动功能的。也就是说，将模型放入unity世界，配置好相关动画之后，模型就可以直接在世界中进行移动。这种方式的好处是，动画与人物的移动距离完美匹配，不会因为CharactorControll 或transform组件移动速度过快而出现明显的移速与动画不匹配的情况。人物的走动，奔跑会非常的自然
2. transform 组件也可以实现人物的移动，但是相较于CharactorControll 和 动画自带的移动。他缺失了爬坡，物体碰撞，等等功能，这些都需要自己去实现，会比较麻烦。
3. 使用CharactorControll 这种方法自带物体碰撞检测，落地检测。甚至还有自动的重力计算集成，非常的方便。

我使用的方式是将三者结合起来使用。使用transform控制人物视角的水平转动。使用动画自带的移动产生距离上的移动，使用CharacorControll的碰撞以及落地检测。这三者混合以后就可以实现一个比较细腻的人物控制系统。

3.1.2 animator 状态机的设计

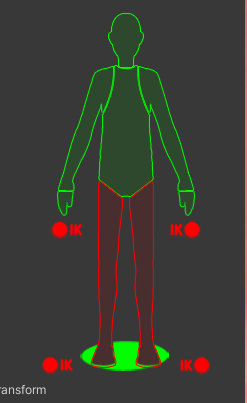
设计状态机的时候，我遇到了一个问题，unity3D中动画的播放只能够一个一个的进行。其实这无可厚非，毕竟引擎并不是那么只能的，要想实现比较好的动画肯定是要花功夫的。这个地方的困难点在于，我如何做出一个状态机，让他既能够播放走路的动画，又能够通知播放持枪射击的动画呢。

前面提到，Unity3D的animator只能一个一个的进行，其实是可以同时进行的，但是当我们同时播放两个不同的动画的时候，Unity3D的人物模型会产生怎杨的结果呢？ 我去尝试了，毫无意外，人物的动作非常的不协调自然。虽然animitor支持同时播放两个动画，但是如果不做任何的处理，这将毫无意义。没有人会想玩一款动作毫无章法可言的游戏。

那怎么去解决这个问题呢？我想到的方法是使用骨骼遮罩(avata mask)。原本同时播放两个动画时，人物动画会冲突。而avata mask 可以做到使人物上半身的动画与下半身的动画之间不会相互影响，上半身播放一套动画，下半身播放另一套动画。两者区分开来，互不打扰。你做你的，我做我的。

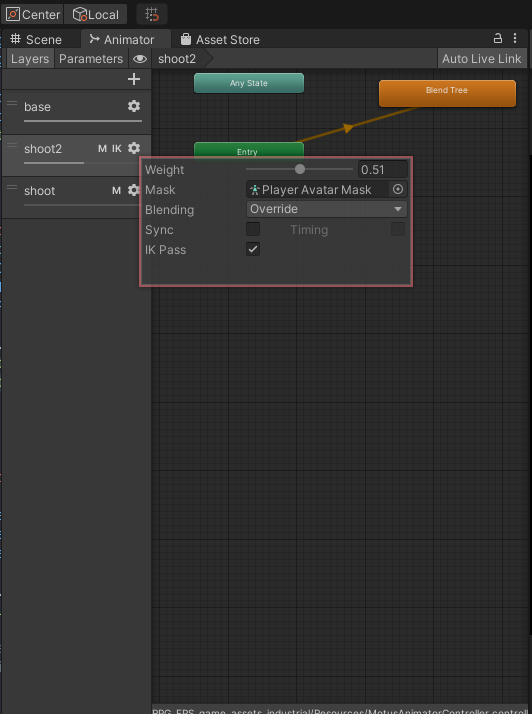
上面介绍过，人物的动画是通过移动骨骼来实现的，而avata mask 的原理就是

在走路的时候，屏蔽上身的骨骼。让动画只能够操纵下半身的骨骼。在设计时，我们屏蔽下身的骨骼，让走路继续走路。这里我们看一张图。



这是我为设计动画做的用来屏蔽下半身动作的骨骼，们只需要在unity中拷贝原骨骼。然后打开新的骨骼，将两腿全部屏蔽掉，就得到了一个遮罩骨骼。这时我们在animator界面新建一个layer。放入射击动画，将这层的mask 填入刚刚设置的骨骼。那么这个动画层就可以完全屏蔽下半身的动画了。

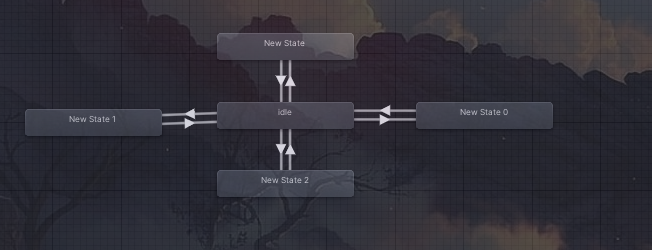
但是这样也带来了一个问题，由于走路的动画不再控制上身的动画。那么势必带来在我们移动的而不进行射击的时候，人物上身动作的僵硬。这个时候我们应该调整射击动画层的layer的weight权重值。让其稍微小一点，这样走路的动画就可以轻微的影响上身的动画，不至于在没有拿枪射击的时候人物上身如同僵尸一般没有任何的动作。



在这里我使用base层来控制人物的移动，shoot2层去控制人物的射击添加遮罩骨骼Player Avatar Mask 并设置Weight为0.5以期能够获得一个比较好的动画效果。

3.1.3 blend tree设计

我使用blend tree 来制作我的人物移动的动画，为什么没有使用上面的状态机来制作人物移动动画呢。因为我一开始尝试了，但是发现行不通。因为在人物朝8个方向的移动中若用状态机，人物不可避免的必须要先进入idle(站立状态) 然后才能进入预想的状态，这么做会产生割裂感。

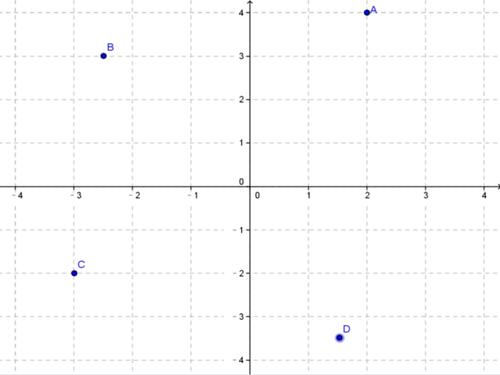


这是一开始使用的方案，可以看到走路动画之间在进行切换的时候，必须通过idle进行过度。其实这才是符合真是世界的移动方式，因为人在转向的时候，就是需要先停下来在进行转向的。但是这一套逻辑放在游戏里面，不适用，尤其是一款设计游戏。因为设计游戏要求高响应，左右的切换的过度站立状态会给玩家带来不好的体验，这也是我为什么会放弃这种设计方法的另一原因。

这种方法的代码控制太过于复杂，Blend tree 能够带来更加简单的处理方法。

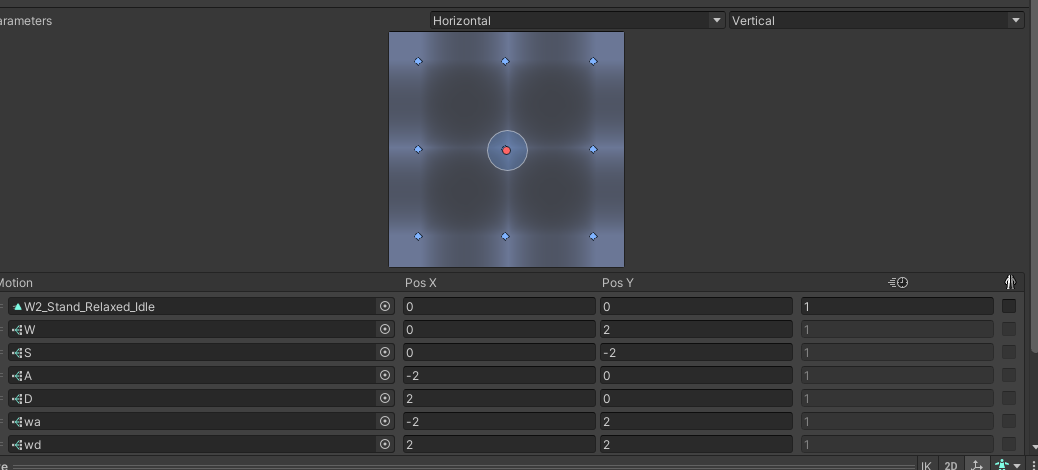
我一开始使用的是1D混合树，但是我发现这种1D混合树实现行走也不方便，只用一个参数去控制8个方位的行走，有点勉强，后来我又使用了2个参数去控制，但是也还是差强人意。最后，我使用了2dblend tree。这个是我认为的最好的用来解决移动动画过度的方法。

2D混合树可以看成是一个正交的坐标系，Horizontal代表水平轴，Vertical代表垂直轴，



看这张图，我可以通过混合horizontal和vertical的值去控制人物的方向，速度。比如A(2,4) 就代表我的人物应该朝右前方移动，c(-3,-2)代表我应该朝做后方进行一定。这种正交坐标系只需要我通过8组不同的正负混合的值，就可以做到控制人物朝向八个不同方位的移动。

那么落实到Unity3D的blend tree中呢？ uinty3d其实已经为我们想到了这种应用场景看下图



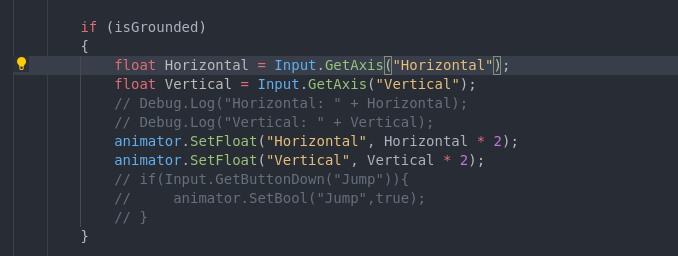
图片中我们并没有看见正交的坐标系，但是我们可以在脑海中虚拟一个以中心的红点作为正交点的坐标系。blend tree要求我们有一个在(0,0)点的初始状态，这里我们直接设置成idle状态就可以了。这里最左上方的位置我们放上向左前方行走的blend tree(为什么是blend tree 后面会在介绍移速的部分说明)。最右上方我们就放上向右前方行走的blend tree以此类推，我们就得到了控制8个方位的blend tree。

那么方位解决了，移速怎么解决呢。我们可以给方位blend tree添加两个动作，walk 和 run。 通过Horizontal 或 Vertical的值控制两者之间的切换（有关值是怎么控制状态切换的前面已经说明）



这样就实现了速度加方位的变换。这是我们只需要编码控制Horizontal 和 Vertical的值就可以了。

这个时候这套2D blend tree的优势就体现出来了，因为他与unity3D 原始绑定的Aixs(“Vertical”) Aixs(“Horizontal”) 是完全一致的。Axis(“Vertical”) 的值介于1和-1之间，正好对应了2d blend tree 中的Vertical值。这大大的减轻了我的编码负担，相较之前的编码方式，这个更加simple,简洁就是美。看代码。

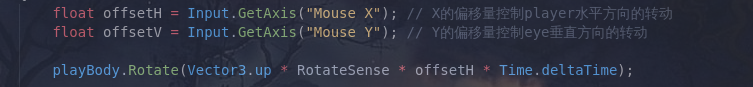


如果不是想代码能够更容易读懂，我可以只用两行。isGrounded表明，只有人物在地上的时候才可以走路。若人物处于置空状态，则无法移动。注意，这套方法只适合动画可以实现人物坐标移动的动画，如果是IPC(in progress 也就是状态动画)是无法使用这套方案实现行走的，必须配合CharactorController 或者transform组件。

3.2 视角变换

3.2.1 水平变换

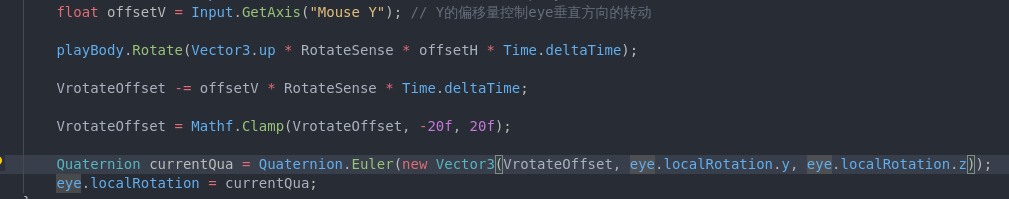
前面其实已经提到了，水平变换可以使用transform.rotate()组件来实现。由于摄像机是与人物绑定的，所以我只需要让人物绕Y轴进行旋转，就可以带动摄像机一起旋转了。



通过获取鼠标在x轴的水平偏移量赋值给Rotated的垂直轴，就可以达到使用鼠标来控制水平视角变换的目的了

3.2.2 垂直变换

垂直变换其实并不是控制人物的身体来实现的，我的垂直变换是通过改变摄像机的角度来实现的。因为摄像机就是人的眼睛，所以此时我应该控制的是摄像机而不是人物。摄像机的转动应该通过设置欧拉角，并将欧拉角转换成四元数。赋值给摄像机。



3.3 准星瞄准

准星瞄准的问题有两个困难点

1. 如何让枪口始终指向准星
2. 如何让人物的身体随着枪口变换身体姿态

前者好理解，如果枪口不指向准星，那枪中射出的子弹将无法准确的击中目标点。

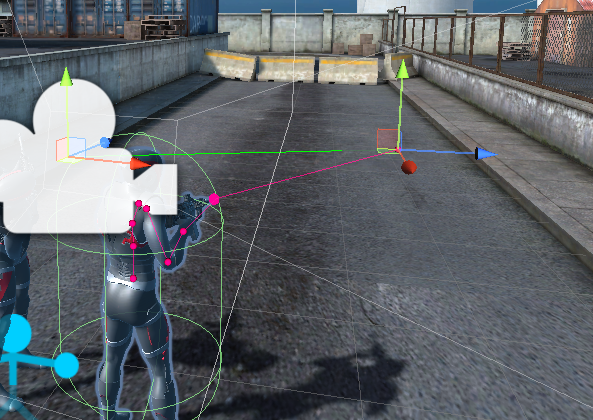
后者想要表达的意思是，人物身体应该随着枪口的太高，和降低，做俯身，仰头的变换。这样才符合直觉。

其实这两个问题我们只需要解决一个就行了。我们只需要解决第二个问题，第一个问题就自然解决了。

3.3.1 准星带动身体

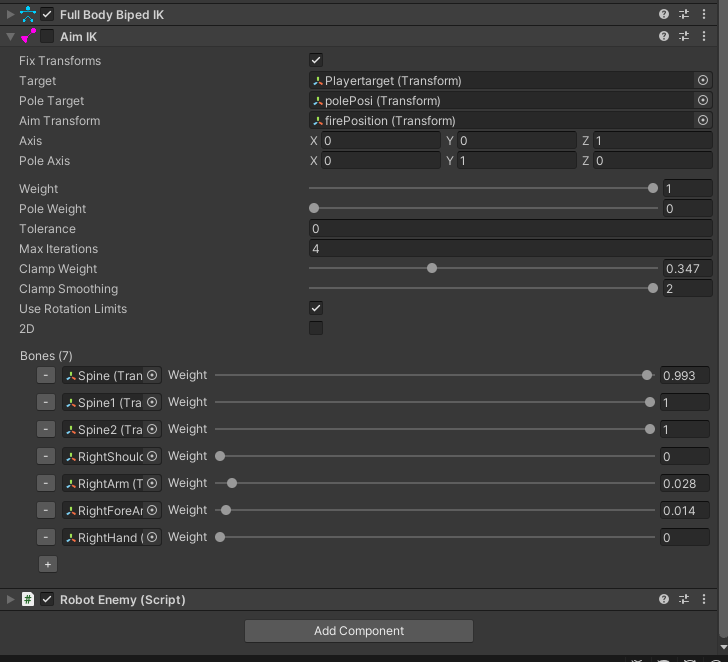
如何解决这个问题呢，前面有提到反向动力骨骼。这里就是他的应用之处。我们让摄像机的中心（摄像机的中心=屏幕中心=准星所指向的位置）发出一道射线，这条射线的终点位置就是我们枪口，准星应该对准的位置。

这时我们在目标位置创建一个空的物体，然后让这个物体拉动我们的枪口与之对齐，这时枪口，准星就在一条线上了。由于人体的骨骼是绑定的，拉动枪口必然会带动手臂，上身一起运动。这时前面所描述的两个问题就都得到了解决。看下面的图片.



绿色的射线就是摄像机中心投射的射线，空色的射线就是枪口指向的射线，这时两者相交的地方，就是准星指向的地方。

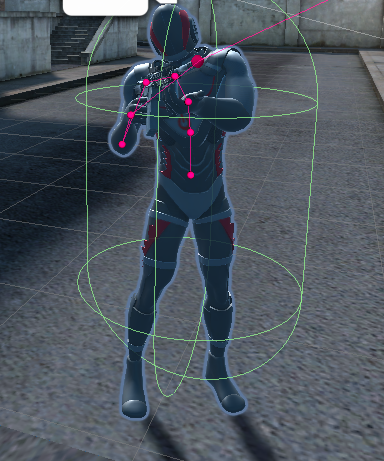
这个想法很好，但是究竟要如何才能实现呢。屏幕中央的射线很好做，但是带动全身骨骼是比较复杂的。这里我们可以使用finalIk这个插件。

这个插件可以非常方便的为角色绑定反向动力骨骼，并且自带有AimIk这一款专门为射击游戏实现的插件。现在介绍如何使用。 

在使用AimIk之前，先添加FullBodyBipedIK组件。然后添加AimIk组件，添加完成之后，就可以看到上面的参数面板了。有几个重要参数需要理解

* Target：枪口，准星所指向的目标物体
* AimTransform: 这里应该填入枪口。
* Bones：这里需要绑定你想要参加运动的骨骼。

这三个参数是必备的，填好之后,AimIk将自动为我们绑定如下骨骼。这里的骨骼我绑定的是三个胸部骨骼点，一个肩部骨骼点，和两个手臂骨骼点。这个可以根据你自己的模型决定。绑好之后，设置他们的权重关系。

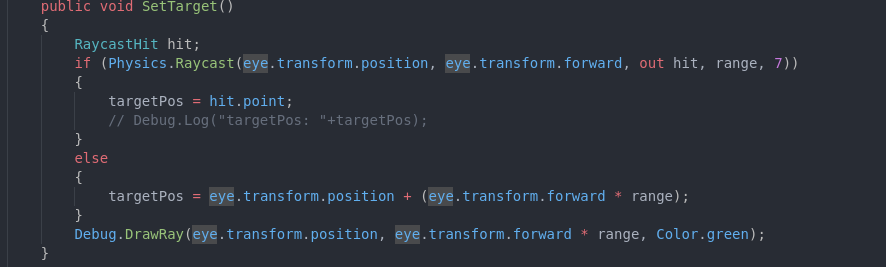


这里，红色的点就是我为这个人物绑定的，需要跟随准星移动的骨骼。

这时就剩下代码逻辑了。

在贴代码之前，必须要先说一件事。由于摄像机放置在人物的脑后，所以摄像机发出的摄像有可能因为接触到人物的身体而导致目标物体被放置在人物的身体生进而引起整个瞄准系统的异常。解决方案是，

我们将人物的Layer添加到射线的layer mask里面就可以了。这样，射线在触碰到人物的时候，就会自动忽略人体，直接穿过去了。



这里使用Physics.Raycast去投射一条射线，从eye的位置(也就是摄像机的位置), eye.transform.forward是向前投射的意思，range是射线的最大长度，7表示人物的Layer,上面说过这里应该忽略掉的。而RaycastHit就是射线击中的位置，让tartgetPost的位置等于hit.point就将前面在AimIk配置的目标物体移动到了指定的位置，这时AimIk的系统就会开始运作，自动带动枪口，胸部手臂，进行运动。如果之前骨骼的权重比配置的不是非常合理的话，人物的动作将会非常的奇怪。应该回去重新配置，尽可能的让动作协调。

# 结 论

# 参考文献

# 致 谢