2.3 武器装弹与角色冲刺

2.3.1 武器开火与装弹动画

(1) 迁移武器装弹药动画资产

从Lyra中迁移资产，我们打开Lyra进入`/All/Game/Weapons/Pistol/Animations`路径下，选取`Weap_Pistol_Fire`和`Weap_Pistol_Reload`两个动画，我们还会选择Rifle和Shotgun的这两个资产，分别在`/All/Game/Weapons/Rifle/Animations`和`/All/Game/Weapons/Shotgun/Animations`路径下。我们可以在`/All/Game/Weapons/`下创建文件夹`MigratedAnims`，并将上面我们需要的文件，复制进来。然后右键点击文件夹选择Migrate进行迁移，迁移后我们的文件夹中应该有如图2.3.1.1所示的动画资产。

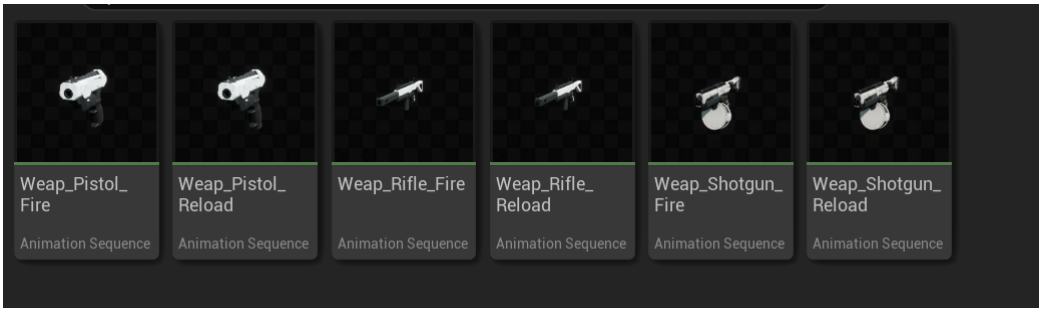

图2.3.1.1 需要迁移的资产

我们还需要迁移`/All/Game/Characters/Heroes/Mannequin/Animations/Actions`下的`MM_Pistol_Fire`，`MM_Rifle_Fire`，`MM_Shotgun_Fire`，`MM_Pistol_Reload`，`MM_Rifle_Reload`，`MM_Shotgun_Reload`资产，一并迁移到我们项目当中。

(2) 创建上半身蒙太奇

找到我们迁移进来的`MM_Pistol_Fire`资产，然后我们创建对应的Montage即可。进入刚创建的Montage中，我们在下方的选项卡中选择对应Slot，将DefaultSlot改成UpperBody，如图2.3.1.2所示。这样可以确保特定的动画仅影响角色的上半身，而不干扰下半身的动画状态。

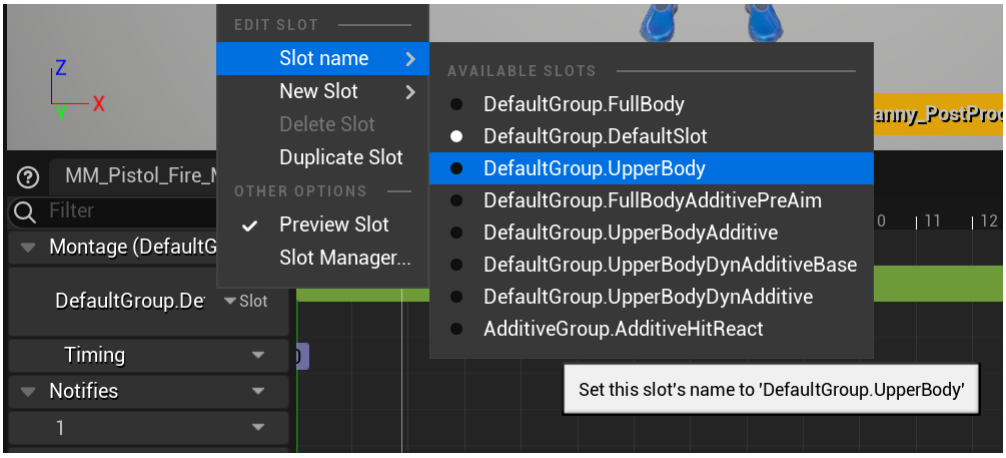

图2.3.1.2 修改Slot

同样地，我们将`MM_Rifle_Fire`和`MM_Shotgun_Fire`也创建对应的Montage并且修改Slot的DefaultSlot改成UpperBody。

我们修改这三个对应的AssetDetails的Blend Option/Blend In/Blend Time为0.1和Blend Option/Blend Out/Blend Time为0.1，那么最终混合出来的这段动画就是花费0.2秒。

然后我们打开我们迁移进来的三个Reload资产，修改对应的Slot将DefaultSlot改成UpperBody。

（3）创建键盘点击映射

添加一个Input Action，如图2.3.1.3所示。然后在GunInputMapping中设置对应的按键R，将Input Action添加到进来，如图2.3.1.4所示。

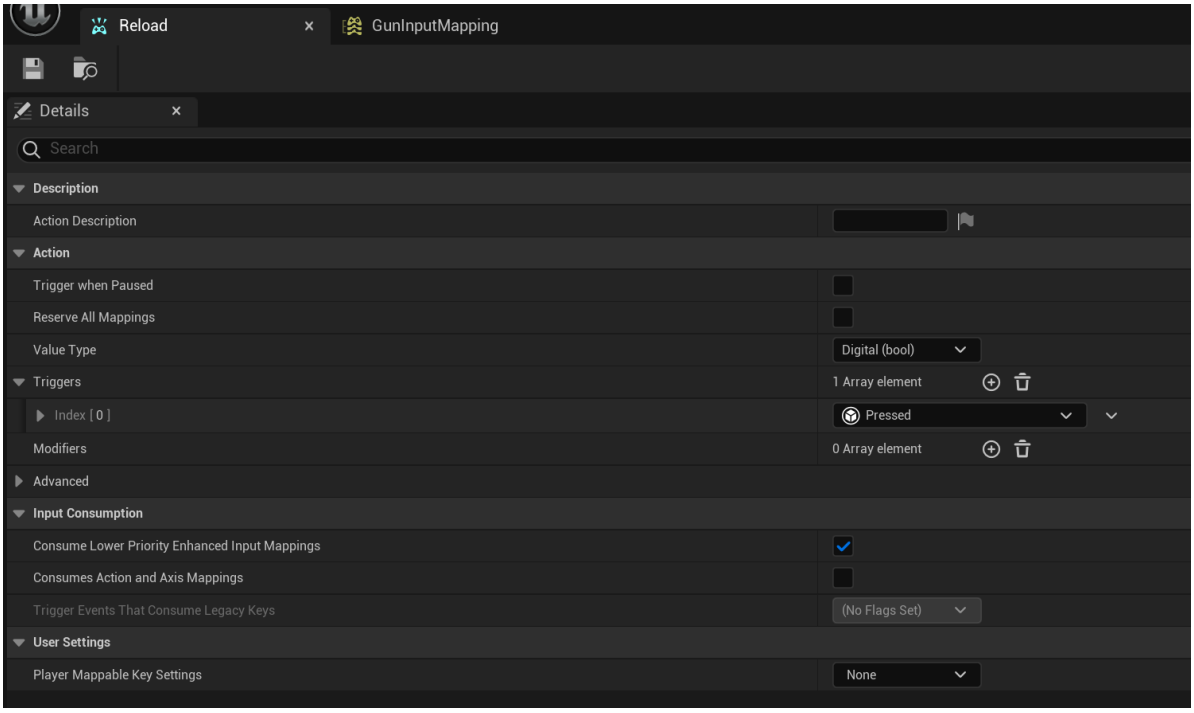

图2.3.1.3 添加一个InputAction

图2.3.1.4 添加进GunInputMapping中

（4）修改AnimInstance

点开MyTutorialCharacterAnimInstance中的AnimGraph，创建一个Slot 'DeaultSlot'节点，如图2.3.1.5所示。

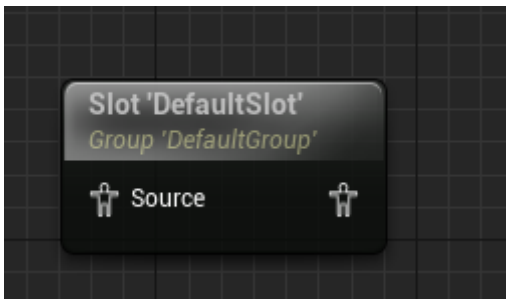

图2.3.1.5 创建Slot'DeaultSlot'节点

选中这个节点，然后修改Details的Settings/SlotName为UpperBody，选择上半身Slot。因为我们仅仅想要让我们的上半身进行蒙太奇的播放，因此，我们将其放在Layered blend per bone节点之前。因此，我们将这个节点插入在Blend Poses by bool 节点与Layered blend per bone之间，如图2.3.1.6所示。

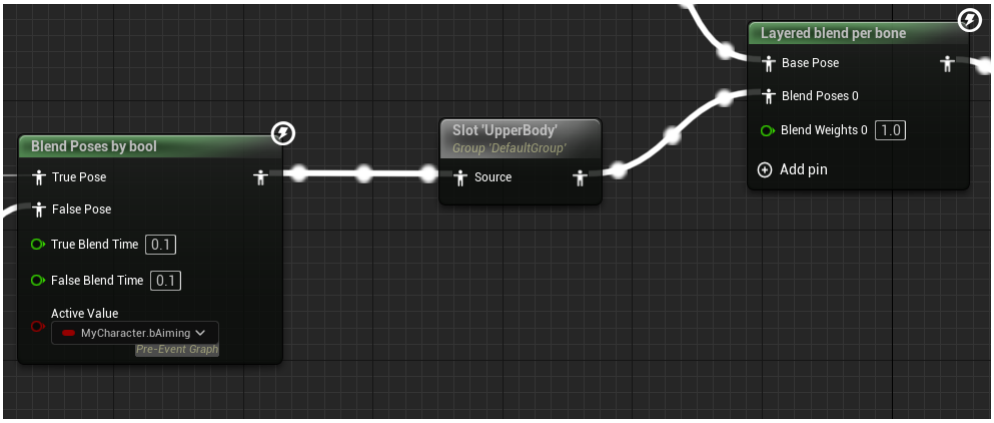

图2.3.1.6 插入Slot节点

（5）添加开火动作

我们之前射击的火焰，在其他玩家的视口中并没有进行显示，因为我们没有进行复制和RPC转发。对于特效，声音或动画蒙太奇之类的东西，我们需要手动进行复制和RPC转发。我们可以将这些东西先利用RPC从客户端发送到服务端，然后通过服务端转发到每一个Controller上.。因此，我们创建以下几个函数供我们使用。

ATutorialCharacter.h

```
void Play_FireMontage();

UFUNCTION(Server, Reliable, Category = "Lobby")
void Server_FireMontage();
void Server_FireMontage_Implementation(FRotator ParamAimRotation);

UFUNCTION(NetMulticast, Reliable, Category = "Lobby")
void Multicast_FireMontage();
void Multicast_FireMontage_Implementation(FRotator ParamAimRotation);

//需填写，在编辑器中
UPROPERTY(EditAnywhere, BlueprintReadWrite)
UAnimMontage* MM_Pistol_Fire_Montage;

UPROPERTY(EditAnywhere, BlueprintReadWrite)
UAnimationAsset* Weap_Pistol_Fire;
```

ATutorialCharacter.cpp

```
void ATutorialCharacter::GunShoot(const FInputActionValue& Value)
{
    ...
    Play_FireMontage();
}
void ATutorialCharacter::Play_FireMontage()
{
    if (HasAuthority()) {
        //在服务器上直接转发给所有的客户端
        Multicast_FireMontage();
    }
    else {
        // 转发给服务器后，再转发给所有客户端
        Server_FireMontage();
    }
```

```
}

void ATutorialCharacter::Server_FireMontage_Implementation()
{
    Multicast_FireMontage();
}

void ATutorialCharacter::Multicast_FireMontage_Implementation()
{
    if (MM_Pistol_Fire_Montage) {
        GetMesh()->GetAnimInstance()->Montage_Play(MM_Pistol_Fire_Montage, 1.0f,
EMontagePlayReturnType::MontageLength, 0.0f, false);
        if(Weap_Pistol_Fire)
        WeaponMesh->PlayAnimation(Weap_Pistol_Fire, false);
    }
}
```

在UE的编辑器中，我们绑定对应的动画蒙太奇和动画，然后当开火的时候调用即可。

（6）添加装弹动作

我们需要用类似的方法，当玩家按下R键的时候，去调用对应的装弹函数，并且当判断当前不处于正在装弹的时候，我们就会利用RPC转发给服务器并转发到每一个用户上，装弹结束后我们修改状态为不正在装弹。因此，有如下的定义内容。

TutorialCharacter.h

```
UPROPERTY()
bool bIsReloading = false;

void GunReloading();
//需填写，在编辑器中
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = GunInput, meta =
(AllowPrivateAccess = "true"))
UInputAction* GunReloadAction;

void Play_ReloadMontage();

UFUNCTION(Server, Reliable, Category = "Lobby")
void Server_ReloadMontage();
void Server_ReloadMontage_Implementation();

UFUNCTION(NetMulticast, Reliable, Category = "Lobby")
void Multicast_ReloadMontage();
void Multicast_ReloadMontage_Implementation();

//需填写，在编辑器中
UPROPERTY(EditAnywhere, BlueprintReadWrite)
UAnimMontage* MM_Pistol_Reload_Montage;
//需填写，在编辑器中
UPROPERTY(EditAnywhere, BlueprintReadWrite)
UAnimationAsset* Weap_Pistol_Reload;

void Reset_IsReloading(float Diration);

UPROPERTY()
FTimerHandle Reset_IsReloadingTimerHandle;
```

```
void Set_IsReloading_false();
```

TutorialCharacter.cpp

```
void ATutorialCharacter::BindGunInputMappingContext()
{
    ...
            EnhancedInputComponent->BindAction(GunReloadAction,
ETriggerEvent::Triggered, this, &ATutorialCharacter::GunReloading);
    ...
}
void ATutorialCharacter::GunReloading()
{
    if (bIsReloading) {

    }
    else {
        bIsReloading = true;
        Play_ReloadMontage();
    }
}

void ATutorialCharacter::Play_ReloadMontage()
{
    if (HasAuthority()) {
        Multicast_ReloadMontage();
    }
    else {
        Server_ReloadMontage();
    }
}

void ATutorialCharacter::Server_ReloadMontage_Implementation()
{
    Multicast_ReloadMontage();
}

void ATutorialCharacter::Multicast_ReloadMontage_Implementation()
{
    if (MM_Pistol_Reload_Montage) {
        float Diration = GetMesh()->GetAnimInstance()-
>Montage_Play(MM_Pistol_Reload_Montage, 1.0f,
EMontagePlayReturnType::MontageLength, 0.0f, false);
        Reset_IsReloading(Diration);
        if (Weap_Pistol_Reload) {
            WeaponMesh->PlayAnimation(Weap_Pistol_Reload, false);
        }
    }
}

void ATutorialCharacter::Reset_IsReloading(float Diration)
{
    GetWorld()->GetTimerManager().SetTimer(Reset_IsReloadingTimerHandle, this,
&ATutorialCharacter::Set_IsReloading_false, Diration, false);
}

void ATutorialCharacter::Set_IsReloading_false()
```

```
{
    bIsReloading = false;
}
```

首先，我们创建对应的bIsReloading判断当前是否处于正在装弹。如果正在处于装弹的话，我们就不会进行换弹的操作。否则，我们则进行换弹，并判断直接转发到每一个玩家，或者转发给服务器后转发给每一个玩家。并在播放后设置一个计时器，当播放完动画后，设置为非正在装弹。大致内容和（5）中描述类似。

另外，我们还可以设置，正在换弹的时候不能进行射击动作，这里就让读者们自行练习了。

2.3.2 角色冲刺

待续。。不好意思。。见Readme