第2章 动画设置与武器系统

2.1 旋转位置动画

2.1.1 简化测试流程

在之前的测试中，因为我们需要客户端的相互连接，我们使用的是Standalone的方式开启测试的，耗时较长。因此，我们修改一些内容，以方便我们如图2.1.1.1所示点击New Editor Windows进行测试。

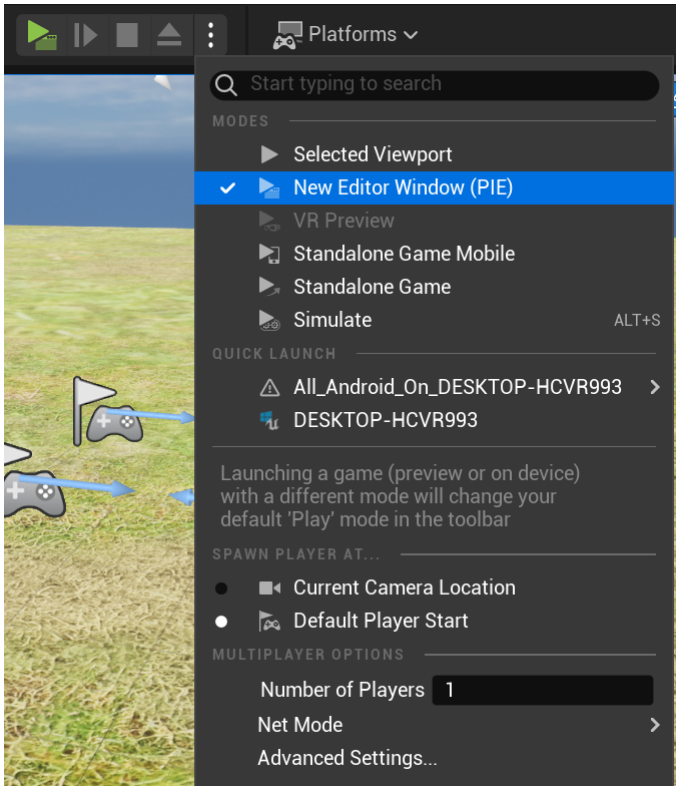

图2.1.1.1 点击New Editor Windows进行测试

现在，我们点击这个进行测试会发现人物实际上是不能移动的。我们并没有进行MyGameInstance->MP_NumConnectedPlayers这个变量的设置，因此在HandleStartingNewPlayer_Implementation中的Num_ExpectedPlayer == All_ConnectedControllers.Num()逻辑永远为假，我们调用不了HandleStartingNewPlayerDelay1这个函数。因此我们只需要添加一个额外的else逻辑，让我们在测试的时候能够调用这个函数即可。

TutorialGameMode.cpp

```
void ATutorialGameMode::HandleStartingNewPlayer_Implementation(APlayerController*
NewPlayer)
{
    Super::HandleStartingNewPlayer_Implementation(NewPlayer);
    ...
    if (MyGameInstance) {
        if (MyGameInstance->IsSoloGame) {...}
        else {
            Num_ExpectedPlayer = MyGameInstance->MP_NumConnectedPlayers;
            if (Num_ExpectedPlayer == All_ConnectedControllers.Num()) {
```

```
                GetWorld()->GetTimerManager().SetTimer(LevelStartTimerHandle,
this, &ATutorialGameMode::HandleStartingNewPlayerDelay1, 0.5f, false);
            }
            else {
                //Test，正常运行时请删除这里的逻辑，测试时必要
                GEngine->AddOnScreenDebugMessage(-1, 5.0f, FColor::Green,
*FString::Printf(TEXT("Test Using HandleStartingNewPlayer_Implementation，正常运行
时请删除这里的逻辑")));
                HandleStartingNewPlayerDelay1();
            }
        }
    }
}
```

那么现在，我们就能够在编译器中，选中Level01的地图进行快速的测试了。

2.1.2 制作旋转位置动画

（1）修改BS_Movement_Quin混合动画

记得我们在1.1中简单的绑定了角色的一些动画。现在，我们来进一步改进角色动画。进入BS_Movement_Quin，到AssetDetails中找到Blend Samples，点击垃圾桶，将原先绑定的进行删除，如图2.1.2.1所示。

图2.1.2.1 删除原先绑定的内容

最终，我们混合的动画效果如图2.1.2.2所示。首先，在Asset Browser栏下我们找到MF_Pistol_Idle_Hipfire资产，然后像1.1中一样，拖拽到我们的动画中，放在(0,0),(-180,0),(180,0)的位置，进行混合。找MF_Pistol_Walk_Bwd放在(-180,150),(180,150)的位置。找到MF_Pistol_Walk_Left放在(-90,150)的位置，找到MF_Pistol_Walk_Right放在(90,150)的位置。同理，查找MF_Pistol_Jog的动画将对应动画放在对应位置，下图2.1.2.2所示即可。

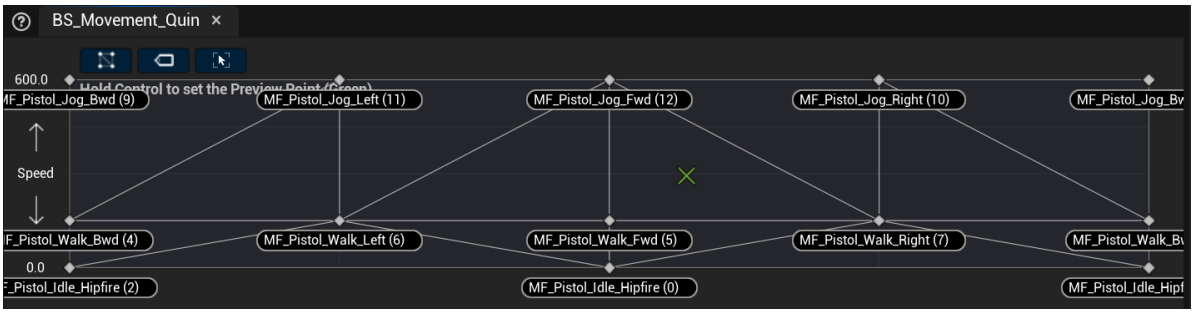

图2.1.2.2 最后混合的动画

（2）思路

我们将使用瞄准偏移来使我们进行转身。因此，我们将bUseControllerRotationYaw这个设置为false。

TutorialCharacter.cpp

```
ATutorialCharacter::ATutorialCharacter(){
    ...
    bUseControllerRotationYaw = false;  //这个变量控制角色是否跟随控制器的Yaw旋转
}
```

我们在构造函数中，将这个变量设置为false，那么角色将不会跟随控制器的Yaw进行旋转。在游戏中的表现就是，进入游戏后，我们将无法通过旋转视角方向旋转角色。

因此，接下来我们要做的就是上半身与相机旋转对齐。

我们需要在每一帧的执行中对玩家视角，也就是相机旋转方向进行检测。我们在每个本地客户端进行逐帧的检测视角偏移，然后我们在本地实现转身动画，然后利用RPC在服务端也进行转身，并复制到每一个客户端上，那么我们就完成了从本地客户端到服务端，再到每一个客户端的转身动画。

（3）添加Tick

首先，我们需要在本地从之前启用Tick，而其他情况不启用Tick。我们在BeginPlayDelay1中完成这个逻辑。

TutorialCharacter.cpp

```
void ATutorialCharacter::BeginPlayDelay1()
{
    if (IsLocallyControlled()) {
        SetActorTickEnabled(true);
        AShooterPlayerController* PlayerController =
Cast<AShooterPlayerController>(GetController());
        if (PlayerController) {
            InGameHUD = PlayerController->InGameHUD;
        }
    }
    else {
        //2.1
        SetActorTickEnabled(false);
    }
}
```

这样，我们就完成了Tick的启用。接下来，我们就要进行Tick的实现。

TutorialCharacter.h

```
UPROPERTY()
float New_YawOffset;

UPROPERTY(Replicated)
float YawOffset;
```

TutorialCharacter.cpp

```
void ATutorialCharacter::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
    if (IsLocallyControlled()) {
        FRotator DeltaRotator =
UKismetMathLibrary::NormalizedDeltaRotator(GetBaseAimRotation(),
GetActorRotation());
        New_YawOffset = DeltaRotator;
        if (HasAuthority()) {
            YawOffset = New_YawOffset;
        }
        else {
            Server_SetYawOffset(New_YawOffset);
            YawOffset = New_YawOffset;
        }
    }
}

void ATutorialCharacter::GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);
    DOREPLIFETIME(ATutorialCharacter, bAiming);

    DOREPLIFETIME(ATutorialCharacter, YawOffset);
}
```

在这段代码中，我们将获取GetBaseAimRotation和GetActorRotation这两个变量值，并计算二者的差值进行归一化。然后保存到本地变量New_YawOffset中。然后判断是否是服务器，如果是服务器我们就设置YawOffset，并将YawOffset复制到其他客户端上。如果是客户端，我们需要用RPC修改服务端的YawOffset，才能复制到其他客户端。因此，编写一个在Server运行的函数。

TutorialCharacter.h

```
UFUNCTION(Server, Reliable, Category = "Lobby")
void Server_SetYawOffset(float ParamYawOffset);
void Server_SetYawOffset_Implementation(float ParamYawOffset);
```

TutorialCharacter.cpp

```
void ATutorialCharacter::Server_SetYawOffset_Implementation(float
ParamYawOffset)
{
    YawOffset = ParamYawOffset;
}
```

这样，我们就修改了服务端上的YawOffset值。在服务器复制回来之前，为了避免明显的延迟，我们在本地也修改YawOffset的值。特别对于一些情况来说，我们常常这样做。

（4）设置AnimInstance

然后，我们打开TutorialCharacterAnimInstance，设置我们刚刚的YawOffset，将其作为蓝图中可获取的变量。

TutorialCharacterAnimInstance.h

```
UPROPERTY(EditAnywhere, BlueprintReadWrite)
float YawOffset;
```

TutorialCharacterAnimInstance.cpp

```
void UTutorialCharacterAnimInstance::NativeUpdateAnimation(float DeltaSeconds)
{
    Super::NativeUpdateAnimation(DeltaSeconds);

    if (MyCharacter) {
        Velocity = MyCharacter->GetVelocity();
        Speed = Velocity.Size();
        Direction = UKismetAnimationLibrary::CalculateDirection(Velocity,
MyCharacter->GetBaseAimRotation());
        bAiming = MyCharacter->bAiming;
        GunPitch = MyCharacter->GetBaseAimRotation().Pitch;
        if (GunPitch > 90) {
            GunPitch -= 360;
        }
        YawOffset = MyCharacter->YawOffset;//设置YawOffset的值
    }
}
```

那么，我们就能够在动画蓝图中获取这个值了。我们打开动画蓝图MyTutorialCharacterAnimInstance，在这个蓝图中的AO_MF_Pistol_Idle_ADS这个节点的输入引脚Yaw插入刚才设置的YawOffset，如图2.1.2.3所示。

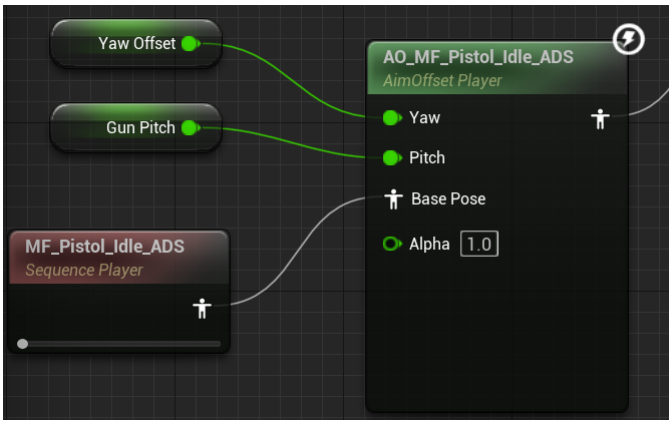

图2.1.2.3 设置YawOffset

这个运行游戏你，当我们点击右键的时候就会是实现镜头跟随着角色进行移动，并且角色的上半身在进行旋转，如图2.1.2.4所示。

图2.1.2.4 上半身旋转

我们修改一下当我们没有点击右键的时候，我们非瞄准状态也应该进行这个动画，最终我们的动画蓝图结构如图2.1.2.5所示。我们将上面的动画部分复制下来，插入到BlendPosesbybool引脚处。为了显示点击右键时的区别，我们将原本的BasePose从MF_Pistol_idle_ADS改成了MF_Pistol_Hipfire_OverridePose。

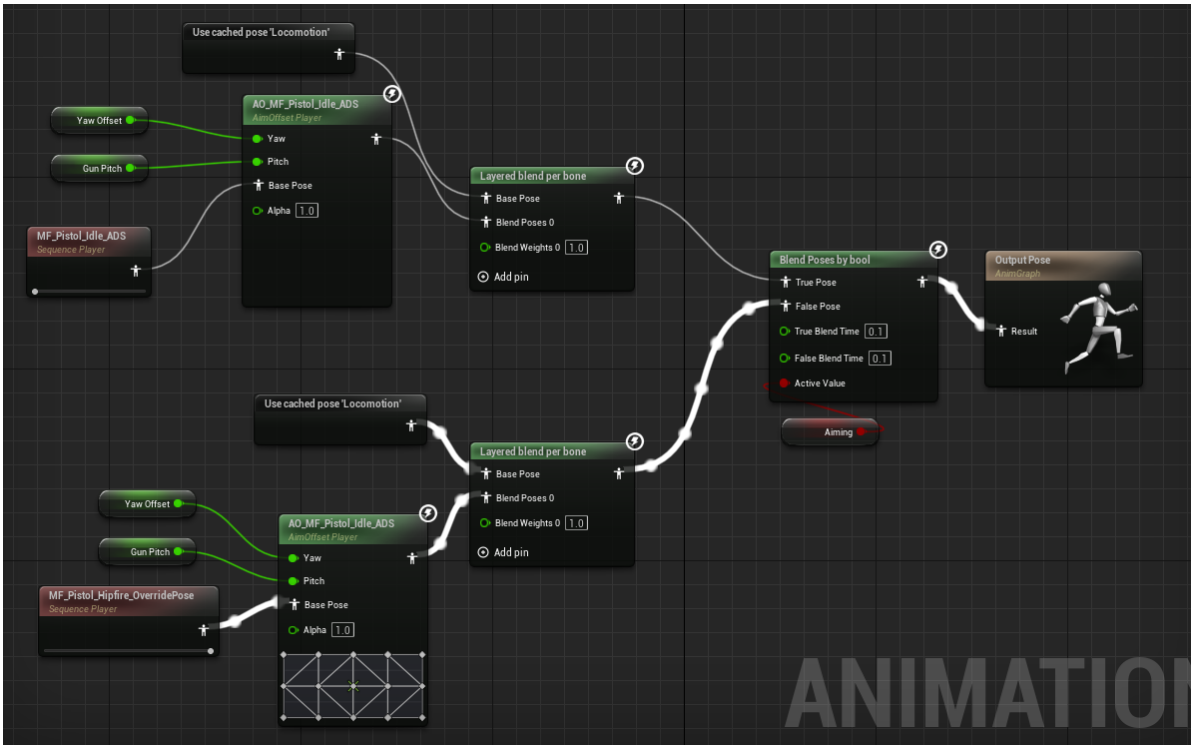

图2.1.2.5 蓝图结构

这个时候，我们没有瞄准的话，上半身也会随之旋转。

（5）设置转身动画

我们需要从Lyra中添加再迁移一个MF_Pistol_TurnRight_180资产过去。这个动画将在角色需要转身180°的时候使用。这个资产应该在Lyra项目中的/All/Game/Characters/Heroes/Mannequin/Animations/Locomotion/Pistol目录下。

双击点开MF_Pistol_TurnLeft_90这个资产，在下方移动帧的位置到大约第26帧。如果我们播放完毕这个动画的话，那么我们会发现这个动画它从26帧之后大多都是静止不动，因此我们删除26帧之后的内容。右键点击26帧的位置，点击Remove from 26 frame to 59 frame。然后，我们对于MF_Pistol_TurnRight_90和MF_Pistol_TurnRight_180也进行类似的操作，我们删去后面静止不动的帧。然后，我们对这三个资产右键点击Create/Create AnimMontage。创建对应的Montage资产。

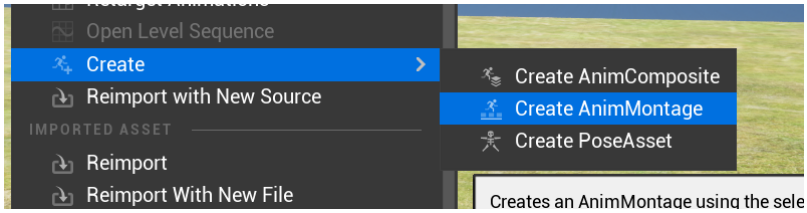

图2.1.2.6 创建对应的Montage资产

我们接着打开C++中，创建一个PlayTurnMonatage函数用于播放转身Montage。

TutorialCharacterAnimInstance.h

```
UFUNCTION()
void PlayTurnMonatage(float ParmsYawOffset);
```

TutorialCharacterAnimInstance.cpp

```
void UTutorialCharacterAnimInstance::PlayTurnMonatage(float ParmsYawOffset)
{
    UAnimMontage* LocalTurnMontageToPlay = nullptr;
    if (abs(ParmsYawOffset) < 60.0f) {
        return;
    }
    else {
        if (abs(ParmsYawOffset) >= 135.0f) {
            LocalTurnMontageToPlay = LoadObject<UAnimMontage>
(nullptr,TEXT("AnimMontage'/Game/Characters/Heroes/Mannequin/Animations/Locomoti
on/Pistol/MF_Pistol_TurnRight_180_Montage.MF_Pistol_TurnRight_180_Montage'"));
            if (LocalTurnMontageToPlay) {
                IsPlayingTurnMontage = true;
                float Diration = Montage_Play(LocalTurnMontageToPlay, 1.0f,
EMontagePlayReturnType::MontageLength, 0.0f, false);
                Reset_TurnMontage(Diration);
            }
        }
        else if (ParmsYawOffset >=60.0f) {
            LocalTurnMontageToPlay = LoadObject<UAnimMontage>(nullptr,
TEXT("AnimMontage'/Game/Characters/Heroes/Mannequin/Animations/Locomotion/Pistol
/MF_Pistol_TurnRight_90_Montage.MF_Pistol_TurnRight_90_Montage'"));
            if (LocalTurnMontageToPlay) {
                IsPlayingTurnMontage = true;
                float Diration = Montage_Play(LocalTurnMontageToPlay, 1.0f,
EMontagePlayReturnType::MontageLength, 0.0f, false);
                Reset_TurnMontage(Diration);
            }
        }
        else if (ParmsYawOffset <=-60.0f) {
            LocalTurnMontageToPlay = LoadObject<UAnimMontage>(nullptr,
TEXT("AnimMontage'/Game/Characters/Heroes/Mannequin/Animations/Locomotion/Pistol
/MF_Pistol_TurnLeft_90_Montage.MF_Pistol_TurnLeft_90_Montage'"));
            if (LocalTurnMontageToPlay) {
```

```
                IsPlayingTurnMontage = true;
                float Diration = Montage_Play(LocalTurnMontageToPlay, 1.0f,
EMontagePlayReturnType::MontageLength, 0.0f, false);
                Reset_TurnMontage(Diration);
            }
        }
    }
}
```

在将来，我们会用对这个传入YawOffset来调用。如果YawOffset的绝对值小于60，那么角色转向的角度不大，我们不进行蒙太奇的播放，直接return。如果大于135，我们就分配一个180转弯的蒙太奇动画。如果大于60，那么播放一个右转的蒙太奇动画。左转也是类似的逻辑。我们会调用LoadObject加载Montage资产，调用Montage_Play进行播放。

这个Montage_Play的第一个参数要播放的动画蒙太奇资产，第二个参数表示播放速率1.0f 表示正常速度，数值大于1会加快播放速度，小于1则减慢。第三个参数表示返回值意义，使用 MontageLength 时，函数会返回整个蒙太奇的时长（总播放时间）。第四个参数指定从蒙太奇的哪个时间点开始播放，0.0f 意味着从动画的起始位置开始播放。第五个参数false 表示不停止其他蒙太奇，如果设为 true，则会停止其他正在播放的蒙太奇。

我们再设置一个bool值IsPlayingTurnMontage，防止连续的播放这个Montage动画。

TutorialCharacterAnimInstance.h

```
UPROPERTY()
bool IsPlayingTurnMontage;

UFUNCTION()
void Reset_TurnMontage(float Diration);

UPROPERTY()
FTimerHandle Reset_TurnMontageTimerHandle;

UFUNCTION()
void Reset_TurnMontageDelay();
```

TutorialCharacterAnimInstance.cpp

```
void UTutorialCharacterAnimInstance::Reset_TurnMontage(float Diration)
{
    GetWorld()->GetTimerManager().SetTimer(Reset_TurnMontageTimerHandle, this,
&UTutorialCharacterAnimInstance::Reset_TurnMontageDelay, Diration, false);
}
void UTutorialCharacterAnimInstance::Reset_TurnMontageDelay()
{
    IsPlayingTurnMontage = false;
}
```

我们同时创建一个函数Reset_TurnMontage用于设置一段时间后，设置IsPlayingTurnMontage值为false。当我们在调用Montage_Play的时候，会返回这个Montage播放的时长，在这个时长之后设置IsPlayingTurnMontage为false即可。

然后，我们修改NativeUpdateAnimation这个函数，添加一段当速度为0的时候才调用PlayTurnMonatage播放这个Montage。

```
void UTutorialCharacterAnimInstance::NativeUpdateAnimation(float DeltaSeconds)
```

```
{
    Super::NativeUpdateAnimation(DeltaSeconds);
    if (MyCharacter) {
        Velocity = MyCharacter->GetVelocity();
        Speed = Velocity.Size();
        Direction = UKismetAnimationLibrary::CalculateDirection(Velocity,
MyCharacter->GetBaseAimRotation());
        bAiming = MyCharacter->bAiming;
        GunPitch = MyCharacter->GetBaseAimRotation().Pitch;
        if (GunPitch > 90) {
            GunPitch -= 360;
        }
        YawOffset = MyCharacter->YawOffset;
        if (Speed == 0.0f && IsPlayingTurnMontage == false) {
            PlayTurnMonatage(MyCharacter->YawOffset);//当停下的时候才会调用这个
        }
    }
}
```

然后，我们打开蓝图界面修改MyTutorialCharacterAnimInstance。我们添加一个Slot'DefaultSlot'节点，如图2.1.2.7所示。这个节点会把当前基础动画与该插槽中正在播放的蒙太奇动画进行混合。

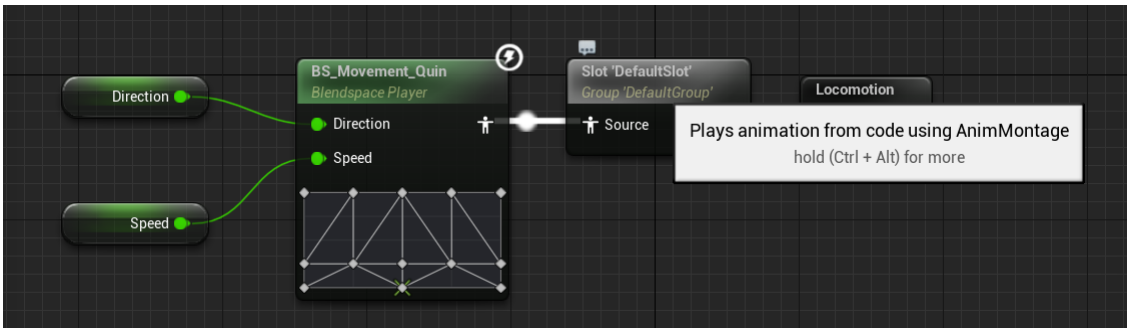

图2.1.2.6 添加一个Slot'DefaultSlot'节点

那么我们此时就有了一个会转身的角色动画了。

（6）优化旋转动画

假如角色旋转角度过快，那么就会出现一个十分不流畅的动画。我们如果能够修改我们传入动画蓝图节点的YawOffset，将它设置在一定的范围内即可。因此我们创建一个新的参数用于起到之前YawOffset的作用，命名为TurnAnimYaw，对于这个参数，我们也同样地启动复制并修改之前的Server_SetYawOffset函数定义进行服务器转发。

TutorialCharacter.h

```
UFUNCTION(Server, Reliable, Category = "Lobby")
void Server_SetYawOffset(float ParamYawOffset, float ParamTurnAnimYaw);
void Server_SetYawOffset_Implementation(float ParamYawOffset, float
ParamTurnAnimYaw);

UPROPERTY()
float New_TurnAnimYaw;

UPROPERTY(Replicated)
float TurnAnimYaw;
```

TutorialCharacter.cpp

```
void ATutorialCharacter::GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);
    DOREPLIFETIME(ATutorialCharacter, bAiming);
    DOREPLIFETIME(ATutorialCharacter, YawOffset);

    DOREPLIFETIME(ATutorialCharacter, TurnAnimYaw);
}
void ATutorialCharacter::Server_SetYawOffset_Implementation(float ParamYawOffset,
float ParamTurnAnimYaw)
{
    YawOffset = ParamYawOffset;
    TurnAnimYaw = ParamTurnAnimYaw;
}
```

然后，我们修改我们的Tick函数，在每一帧的时候设置对应的TurnAnimYaw。

```
void ATutorialCharacter::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
    if (IsLocallyControlled()) {
        FRotator DeltaRotator =
UKismetMathLibrary::NormalizedDeltaRotator(GetBaseAimRotation(),
GetActorRotation());
        New_YawOffset = FMath::Clamp(DeltaRotator.Yaw, -60.0f, 60.0f);
        New_TurnAnimYaw = DeltaRotator.Yaw;
        if (HasAuthority()) {
            YawOffset = New_YawOffset;
            TurnAnimYaw = New_TurnAnimYaw;
        }
        else {
            Server_SetYawOffset(New_YawOffset, New_YawOffset);
            TurnAnimYaw = New_TurnAnimYaw;
            YawOffset = New_YawOffset;
        }
    }
}
```

我们将New_YawOffset利用Clamp函数设置在（-60，60）之间。然后客户端和服务器设置TurnAnimYaw。打开TutorialCharacterAnimInstance.cpp，我们修改其中PlayTurnMonatage调用时的传参。

TutorialCharacterAnimInstance.cpp

```
void UTutorialCharacterAnimInstance::NativeUpdateAnimation(float DeltaSeconds)
{
    Super::NativeUpdateAnimation(DeltaSeconds);
    if (MyCharacter) {
        ...
        YawOffset = MyCharacter->YawOffset;
        if (Speed == 0.0f && IsPlayingTurnMontage == false) {
            PlayTurnMonatage(MyCharacter->TurnAnimYaw);
        }
    }
}
```

这里，我们将传入YawOffset参数改为TurnAnimYaw参数。现在，当我们旋转视角的时候，角色将在60°时停止旋转，如果我们旋转幅度很大的时候，角色将播放180°的Montage，再次对准自己。

现在，我们需要确保的是移动的时候保持正常。

(7) bUseControllerDesiredRotation

我们通过设置CharacterMovement的bUseControllerDesiredRotation参数，当这个参数为true的时候，角色会以控制器的旋转为基准来调整自己的旋转，当这个参数为false的时候，角色的朝向通常由移动方向决定。那么我们可以当，角色在移动的时候，设置对应的参数值为true，让角色以控制器的旋转为基准来调整自己的旋转。而当角色不移动的时候，让角色用我们之前的那一套原地的旋转系统即可。

TutorialCharacter.h

```
UFUNCTION(Server, Reliable, Category = "Lobby")
void Server_SetUsedesiredRot(bool UseControllerDesiredrotation);
void Server_SetUsedesiredRot_Implementation(bool UseControllerDesiredrotation);
```

TutorialCharacter.cpp

```
void ATutorialCharacter::Server_SetUsedesiredRot_Implementation(bool
UseControllerDesiredrotation)
{
    GetCharacterMovement()->bUseControllerDesiredRotation =
UseControllerDesiredrotation;
}
```

由于这属性需要复制到客户端，因此我们需要在服务器修改这个值，才能复制到每个客户端。因此我们定义Server_SetUsedesiredR ot函数进行设置。然后，我们修改我们Tick中的逻辑即可。

```
void ATutorialCharacter::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
    if (IsLocallyControlled()) {
        FRotator DeltaRotator =
UKismetMathLibrary::NormalizedDeltaRotator(GetBaseAimRotation(),
GetActorRotation());
        New_YawOffset = FMath::Clamp(DeltaRotator.Yaw, -60.0f, 60.0f);
        FVector Velocity = GetVelocity();
        float XYLength = FVector(Velocity.X, Velocity.Y, 0.0f).Size();
        if (XYLength > 0) {
            if (HasAuthority()) {
                YawOffset = New_YawOffset;
```

```
                if (!GetCharacterMovement()->bUseControllerDesiredRotation) {
                    GetCharacterMovement()->bUseControllerDesiredRotation =
true;
                }
            }
            else {
                Server_SetYawOffset(New_YawOffset, 0.0f);
                YawOffset = New_YawOffset;
                if (!GetCharacterMovement()->bUseControllerDesiredRotation) {
                    Server_SetUsedesiredRot(true);
                    GetCharacterMovement()->bUseControllerDesiredRotation =
true;
                }
            }
        }
        else {
            New_TurnAnimYaw = DeltaRotator.Yaw;
            if (HasAuthority()) {
                YawOffset = New_YawOffset;
                TurnAnimYaw = New_TurnAnimYaw;
                if (GetCharacterMovement()->bUseControllerDesiredRotation) {
                    GetCharacterMovement()->bUseControllerDesiredRotation =
false;
                }
            }
            else {
                Server_SetYawOffset(New_YawOffset, New_YawOffset);
                TurnAnimYaw = New_TurnAnimYaw;
                YawOffset = New_YawOffset;
                if (GetCharacterMovement()->bUseControllerDesiredRotation) {
                    Server_SetUsedesiredRot(false);
                    GetCharacterMovement()->bUseControllerDesiredRotation =
false;
                }
            }
        }
    }
}
```

当我们速度大于0的时候，我们就设置bUseControllerDesiredRotation为true，反之则设置为false。

我们在Character构造函数中设置GetCharacterMovement()->RotationRate.Yaw = 360.0f;当我们修改这个值，会影响从静止到移动的过渡速度，可以使得整体更加平滑。

2.1.3 总结

我们在这一小节之中，我们简化了我们的测试流程，制作了对应的旋转位置动画。同时，我们也可以试试其他的动画是否生效，当点击某个输入的时候，应该进行什么操作可实现其他的角色动作。