```cpp
//KeepMowingALawnCharacter.cpp

// Copyright 1998-2018 Epic Games, Inc. All Rights Reserved.

#include "KeepMowingALawnCharacter.h"
#include "HeadMountedDisplayFunctionLibrary.h"
#include "Camera/CameraComponent.h"
#include "Components/CapsuleComponent.h"
#include "Components/InputComponent.h"
#include "Components/BoxComponent.h"
#include "Components/AudioComponent.h"
#include "Engine.h"
#include "TimerManager.h"
#include "GameFramework/CharacterMovementComponent.h"
#include "GameFramework/Controller.h"
#include "GameFramework/SpringArmComponent.h"

//////////////////////////////////////////////////////////////////////////
// AKeepMowingALawnCharacter

AKeepMowingALawnCharacter::AKeepMowingALawnCharacter()
{
    // Set size for collision capsule
    GetCapsuleComponent()->InitCapsuleSize(42.f, 96.0f);


    // Don't rotate when the controller rotates. Let that just affect the camera.
    bUseControllerRotationPitch = false;
    bUseControllerRotationYaw = false;
    bUseControllerRotationRoll = false;

    // Create a follow camera
    FollowCamera = CreateDefaultSubobject<UCameraComponent>(TEXT("FollowCamera"));
    FollowCamera->SetupAttachment(RootComponent, USpringArmComponent::SocketName);


    /*/Code I Made/Edited /*/

    /** Below are camera settings **/
    FVector NewCameraPosition(-27.8, 0, -30.7886);
    FRotator NewCameraRotation(-10, 0, 0);
    FVector NewCameraScale(0, 0, 0);

    FTransform NewCameraTransform(NewCameraRotation, NewCameraPosition, NewCameraScale);
    FollowCamera->SetRelativeTransform(NewCameraTransform);


    SkyViewCamera = CreateDefaultSubobject<UCameraComponent>(TEXT("SkyViewCamera"));
    SkyViewCamera->SetupAttachment(RootComponent, USpringArmComponent::SocketName);

    NewCameraPosition.Set(-12.5545, 0, 30);
    SkyViewCamera->SetRelativeTransform(FTransform(FRotator(-90, 0,
    0),FVector(-12.5445,0,1346),FVector(0,0,0)));
    SkyViewCamera->Deactivate();


    /** Below are box collision objects which handle tire tracking **/
    LeftTirePos = CreateDefaultSubobject<UBoxComponent>(TEXT("LeftTirePos"));
    RightTirePos = CreateDefaultSubobject<UBoxComponent>(TEXT("RightTirePos"));

    LeftTirePos->SetupAttachment(RootComponent);
    RightTirePos->SetupAttachment(RootComponent);

    LeftTirePos->SetRelativeLocation(FVector(-50, -40, -80));
    RightTirePos->SetRelativeLocation(FVector(-50, 40, -80));

    LeftTirePos->SetRelativeScale3D(FVector(.5, .5, .5));
    RightTirePos->SetRelativeScale3D(FVector(.5, .5, .5));

    /** below are components for the sound of the mower **/
    MowerSounds = CreateDefaultSubobject<UAudioComponent>(TEXT("SoundEffects"));

    MowerSounds->SetupAttachment(RootComponent);


    // Note: The skeletal mesh and anim blueprint references on the Mesh component (inherited from Character)
    // are set in the derived blueprint asset named MyCharacter (to avoid direct content references in C++)
}

//////////////////////////////////////////////////////////////////////////
// Input

void AKeepMowingALawnCharacter::SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent)
{
    // Set up gameplay key bindings
    check(PlayerInputComponent);

    PlayerInputComponent->BindAxis("MoveLeft", this, &AKeepMowingALawnCharacter::MoveLeft);
    PlayerInputComponent->BindAxis("MoveRight", this, &AKeepMowingALawnCharacter::MoveRight);

    //Control Mower Speeds
    PlayerInputComponent->BindAction("ShiftUp",IE_Pressed, this, &AKeepMowingALawnCharacter::ShiftUp);
    PlayerInputComponent->BindAction("ShiftDown", IE_Pressed, this, &AKeepMowingALawnCharacter::ShiftDown);

    //Allow Mower Sound to be turnd on/off
    PlayerInputComponent->BindAction("Enable/Disable Mower Sound", IE_Pressed, this, &AKeepMowingALawnCharacter::ChangeMowerSoundState);

    //Allows for the camera to be moved when the mower is stopped
    PlayerInputComponent->BindAxis("LookRight", this, &AKeepMowingALawnCharacter::LookRight);
    PlayerInputComponent->BindAxis("LookLeft", this, &AKeepMowingALawnCharacter::LookLeft);

    //Allows for the Camera to toggle between Skyview and First Person
    PlayerInputComponent->BindAction("ChangeCameraView", IE_Pressed, this, &AKeepMowingALawnCharacter::ChangeCameraView);

    // VR headset functionality
    PlayerInputComponent->BindAction("ResetVR", IE_Pressed, this, &AKeepMowingALawnCharacter::OnResetVR);
}


void AKeepMowingALawnCharacter::OnResetVR()
{
    UHeadMountedDisplayFunctionLibrary::ResetOrientationAndPosition();
}


/** Returns the value of the left joystick **/
void AKeepMowingALawnCharacter::MoveLeft(float Value)
{
    LeftJoystickValue = Value;
}

/** Returns the value of the right joystick **/
void AKeepMowingALawnCharacter::MoveRight(float Value)
{
    RightJoystickValue = Value;
}
```

```cpp


void AKeepMowingALawnCharacter::Tick(float Delta)
{
    Super::Tick(Delta);

    FRotator NewCharacterRotation = GetActorRotation();

    //Left Joystick Press
    float leftangle = 360 + (SpeedMultiplier * LeftJoystickValue);
    float rightangle = 360 + (SpeedMultiplier * RightJoystickValue);
    float distancechange = (SpeedMultiplier * LeftJoystickValue) + (SpeedMultiplier * RightJoystickValue);


    if (LeftJoystickValue != 0 || RightJoystickValue != 0)
    {

        /** this block of if statements forces the driver to not be able to overturn at higher speeds **/
        if (distancechange > 3 && RightJoystickValue < -.4)
            distancechange = 3;
        else if (distancechange < -3 && LeftJoystickValue > .4)
            distancechange = -3;
        else if (distancechange < -3 && RightJoystickValue > .4)
            distancechange = -3;
        else if (distancechange > 3 && LeftJoystickValue < -.4)
            distancechange = 3;

        NewCharacterRotation.Yaw += distancechange;

        FVector NewLocation = CalculatePosition(rightangle, leftangle);

        /** This checks if the object hit something while moving and tracks distance accordingly**/
        FHitResult hitresult;
        if (!SetActorLocation(NewLocation, true, &hitresult))
        {   UE_LOG(LogClass, Log, TEXT("Blocked"));}
        else if(LeftJoystickValue != 0 && RightJoystickValue != 0)
        {   DistanceTravelled += PotentialDistanceTravelled;}

        SetActorRotation(NewCharacterRotation);
    }

        FRotator RotateCamera = GetActorRotation();

        FHitResult hitresult;
        RotateCamera.Add(0, LookLeftValue + LookRightValue, 0);
        FollowCamera->SetWorldRotation(RotateCamera,true, &hitresult);

        //UE_LOG(LogClass, Log, TEXT("PositionTesting %f, %f"), GetActorLocation().X, GetActorLocation().Y);

}


/* CalculateRotation
*This function shall take in two FVectors containing location points. One to be rotated (currentworldloc) and one to be the pivot (PivotPoint) about an angle (AngleofChangeDegree)
*/
FVector AKeepMowingALawnCharacter::CalculateRotation(FVector CurrentWorldLoc, FVector PivotPoint, float AngleofChangeDegree)
{

    float ShiftCoordinatesX = CurrentWorldLoc.X - PivotPoint.X;
    float ShiftCoordinatesY = CurrentWorldLoc.Y - PivotPoint.Y;

    /*Mat is short for Matrix, as these are the values for the matrix rotation*/
    float MatSinDeg = FMath::Sin(FMath::DegreesToRadians(AngleofChangeDegree));
    float MatCosDeg = FMath::Cos(FMath::DegreesToRadians(AngleofChangeDegree));

    float RotatedCoordinatesX = (ShiftCoordinatesX * MatCosDeg) - (ShiftCoordinatesY * MatSinDeg);
    float RotatedCoordinatesY = (ShiftCoordinatesX * MatSinDeg) + (ShiftCoordinatesY * MatCosDeg);

    FVector UpdatedCoordinatesVector(RotatedCoordinatesX + PivotPoint.X, RotatedCoordinatesY + PivotPoint.Y, CurrentWorldLoc.Z);

    //UE_LOG(LogClass, Log, TEXT("SinFloat: %f"), UpdatedCoordinatesVector.X);
    //UE_LOG(LogClass, Log, TEXT("CosFloat: %f"), UpdatedCoordinatesVector.Y);

    return UpdatedCoordinatesVector;

}

/**
* Calculates the position of a point between two vectors about a given world coordinant (originpoint)
* Note: Uses the (LocationTwo) variable for the magnitude of the new point
**/
FVector AKeepMowingALawnCharacter::LocationCombiner(FVector LocationOne, FVector LocationTwo, FVector OriginPoint, float multiplier)
{

    //Calculate new true vector direction
    FVector leftlocalvector = FVector(LocationOne.X - OriginPoint.X, LocationOne.Y - OriginPoint.Y, 0);
    FVector rightlocalvector = FVector(LocationTwo.X - OriginPoint.X, LocationTwo.Y - OriginPoint.Y, 0);

    float distancetotravel = rightlocalvector.Size() * multiplier;
    PotentialDistanceTravelled = distancetotravel;

    FVector newplacement = leftlocalvector + rightlocalvector;
    newplacement.Normalize(20);

    //Updates the normalized newplacement with the needed magnitude and put it in world coordiantes.
    newplacement.Set((newplacement.X * distancetotravel) + OriginPoint.X, (newplacement.Y * distancetotravel) + OriginPoint.Y, OriginPoint.Z);

    return newplacement;

}

/*
* This will calculate the new position for the lawnmower based upon the given angles.
* Rightangle is for the rightjoysticks value yeild.
* Leftangle is for the leftjoysticks value yeild.
*
* Note: This code is wasteful in order to increase readability, as it is already difficult to read
*/

FVector AKeepMowingALawnCharacter::CalculatePosition(float rightangle, float leftangle)
{

    FVector currentlocation = GetActorLocation();
    FVector currentlefttire = LeftTirePos->GetComponentLocation();
    FVector currentrighttire = RightTirePos->GetComponentLocation();
    //Calculate Right First Rotation

    FVector newlefttire = CalculateRotation(currentlefttire, currentrighttire, leftangle);
    FVector newrightfirstcenter = CalculateRotation(currentlocation, currentrighttire, leftangle);

    newrightfirstcenter = CalculateRotation(newrightfirstcenter, newlefttire,
    rightangle);

    //Calculate Left First Rotation
    FVector newrighttire = CalculateRotation(currentrighttire, currentlefttire,
    rightangle);
    FVector newleftfirstcenter = CalculateRotation(currentlocation, currentlefttire,
    rightangle);

    newleftfirstcenter = CalculateRotation(newleftfirstcenter, newrighttire, leftangle);

    //Calculate new true vector direction
    return LocationCombiner(newleftfirstcenter, newrightfirstcenter, currentlocation, 1);


}

void AKeepMowingALawnCharacter::ShiftUp()
{
    if (SpeedMultiplier < 10)
        SpeedMultiplier++;
}

void AKeepMowingALawnCharacter::ShiftDown()
{
    if (SpeedMultiplier > 3)
        SpeedMultiplier--;
}

void AKeepMowingALawnCharacter::LookRight(float value)
{
    LookRightValue = value * 90;
}


void AKeepMowingALawnCharacter::LookLeft(float value)
{
    LookLeftValue = value * 90;
}

void AKeepMowingALawnCharacter::BeginPlay()
{
    Super::BeginPlay();
    SpeedMultiplier = 5;
    FRotator NewCameraRotation = GetActorRotation();
    FollowCamera->SetWorldRotation(NewCameraRotation, true);
    TimeElapsed = 0;
    DistanceTravelled = 0;
    GetWorldTimerManager().SetTimer(TimeMangementHandler, this,
    &AKeepMowingALawnCharacter::IncreaseTime, 1.0f, true, 0.0f);

}

void AKeepMowingALawnCharacter::ChangeMowerSoundState()
{
    if (MowerSounds->IsPlaying())
    {
        MowerSounds->Stop();
    }
    else
    {
        MowerSounds->Play();
    }
}


void AKeepMowingALawnCharacter::StopMowerSound()
{
    if (MowerSounds->IsPlaying())
    {
        MowerSounds->Stop();
    }

    return;
}

float AKeepMowingALawnCharacter::GetGearStat()
{

    return SpeedMultiplier;
}


void AKeepMowingALawnCharacter::ChangeCameraView()
{
    if (FollowCamera->IsActive())
    {
        FollowCamera->Deactivate();
        SkyViewCamera->Activate();
    }
    else
    {
        SkyViewCamera->Deactivate();
        FollowCamera->Activate();
    }
}

void AKeepMowingALawnCharacter::IncreaseTime()
{
    TimeElapsed += 1;
}

float AKeepMowingALawnCharacter::GetElapsedTime()
{
    //if (GEngine)
    //  GEngine->AddOnScreenDebugMessage(-1, 15.0f, FColor::Yellow,
    FString::Printf(TEXT("Distance: %f"), TimeElapsed));

    return TimeElapsed;
}

float AKeepMowingALawnCharacter::GetAbsoluteTime()
{
    return AbsoluteTime;
}


float AKeepMowingALawnCharacter::GetTravelledDistance()
{
    return DistanceTravelled;
}

void AKeepMowingALawnCharacter::SetAbsoluteTime()
{
    AbsoluteTime = TimeElapsed;
    return;
}
```