Game Engines in AI Exercise 01

Waldemar Zeitler

November 13, 2018

Info

The code of the tasks is in addition to the following descriptions documented.

Task 1

For the task a pawn was created in C++ with added input components and a camera (PlayerPawn.cpp and PlayerPawn.h). The moving of the pawn is done by W,A,S,D and the rotation of the mouse. Except for the pitch rotation everything was done pretty basic, like most tutorials show. If the pitch rotation is handled with SetActorRotation() a gimbald lock occurs. The following code shows one way around it:

```
FQuat QuatRotation = FQuat(FRotator(RotationValue, 0, 0));

AddActorLocalRotation(QuatRotation, false, 0, ETeleportType::None
    ↪ );
```

The pawn received a sphere as body, by adding the component in the editor.

Task 2

In this task the player pawn received a blueprint function, which creates a sphere in front of the pawn and print to the outputlog which actors where hit by the sphere. The function is called by the button E, which was bind to call the function FindActors. FindActors receives a list of hit actors by the blueprint function GetActroList, which also draws a debug sphere in front of the character.

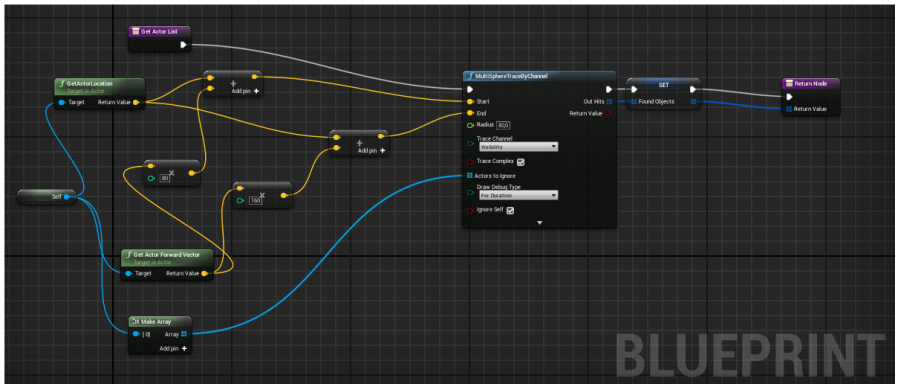

Figure 1: Blueprint Function

FindActors iterates than over the list of actors and prints the found actors to the outputlog.

Task 3

To calculate a speed up in rotation and movement velocity the pawn received an ActorComponent. The component is called SpeedComponent and constantly adds speed values for rotation and movment. These values can be changed in the editor. The rotation of the character becomes very fast out of control with this implementation and should be used sensitively.

Task 4

I solved this task slightly different, because I couldn't add the UBoxComponent, which I created, to the scale class. Probably because I forgot to add the header of the box component.
My solution is done with an actual ATriggerBox class, which has and overlap begin and overlap end function and can access the scale object. The scale object is found with in iterator over all objects in the world.

```
for (TActorIterator<AScaleActor> ActorItr(GetWorld());
    ↪ ActorItr; ++ActorItr) {
    // Same as with the Object Iterator, access the subclass
        ↪ instance with the * or -> operators.
    ScaleActor = *ActorItr;
}
```

When the overlap begin event is triggered a bool variable in the scale actor is activated, which than calls the ScaleRotated function. The rotation and scaling is done by some parameters for the strength of the effects and by a bool which says if the object should be down of up scaled. Those parameters can be set in the BeginPlay function of the blueprint.

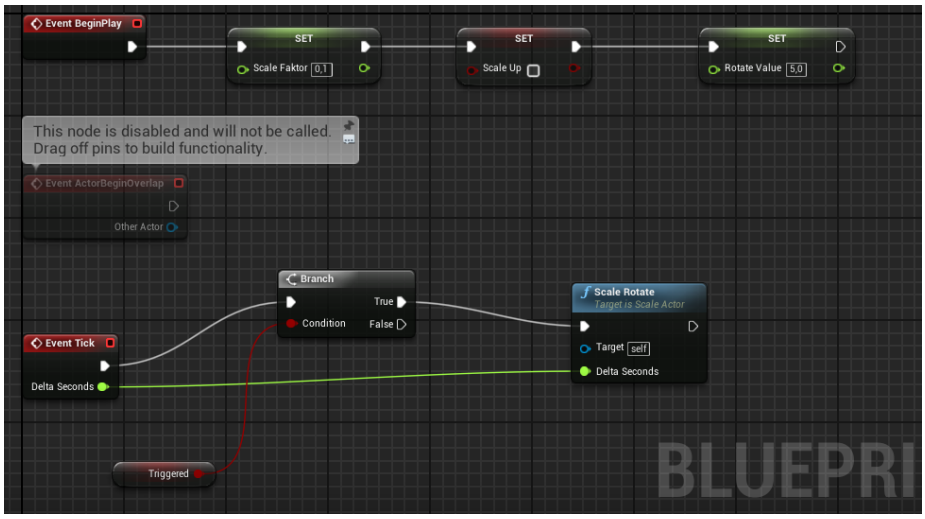

Figure 2: Blueprint Function

The function for rotating and scaling looks as follows:

```
void AScaleActor::ScaleRotate(float DeltaSeconds)
{
    // Rise the min time value of the curve, until the max time is
        ↪  reached, to stop the "animation"
    if (MinCurveTime < MaxCurveTime)
    {
        // Calculate the new rotation
        float Rotation = RotateValue * Curve->GetFloatValue(
            ↪ MinCurveTime);
        FQuat NewRotation = FQuat(FRotator(Rotation, 0.f, 0.f));
        AddActorWorldRotation(NewRotation);

        // Calculate the scaling, depending if the object should
            ↪ get smaller or bigger
        float Scaling = ScaleFaktor * Curve->GetFloatValue(
            ↪ MinCurveTime);
        if (!bScaleUp && Scaling > 0)
        {
            Scaling = Scaling * -1;
        }

        FVector RisingScale = GetActorScale();
        RisingScale += FVector(Scaling, Scaling, Scaling);

        SetActorScale3D(RisingScale);

        MinCurveTime += DeltaSeconds;
    }
    else
```

```
    {
        // Rest the time values and deactived the trigger
        Curve->GetTimeRange(MinCurveTime, MaxCurveTime);
        bTriggered = false;
    }
}
```

The function receives the seconds from the tick function. The seconds are than used to get the float point on the curve and to rise the current time. The new rotation and scale can be calculated with the parameters for scaling, rotation and the curve. If the current time is over the max time of the curve the "animation" gets stopped and the min and max values of the curve rested. The "animation" is always triggered when the pawn or an actors enters the trigger box.
The curve has to be set in the blueprint and also the static mesh which should be used for scaling.

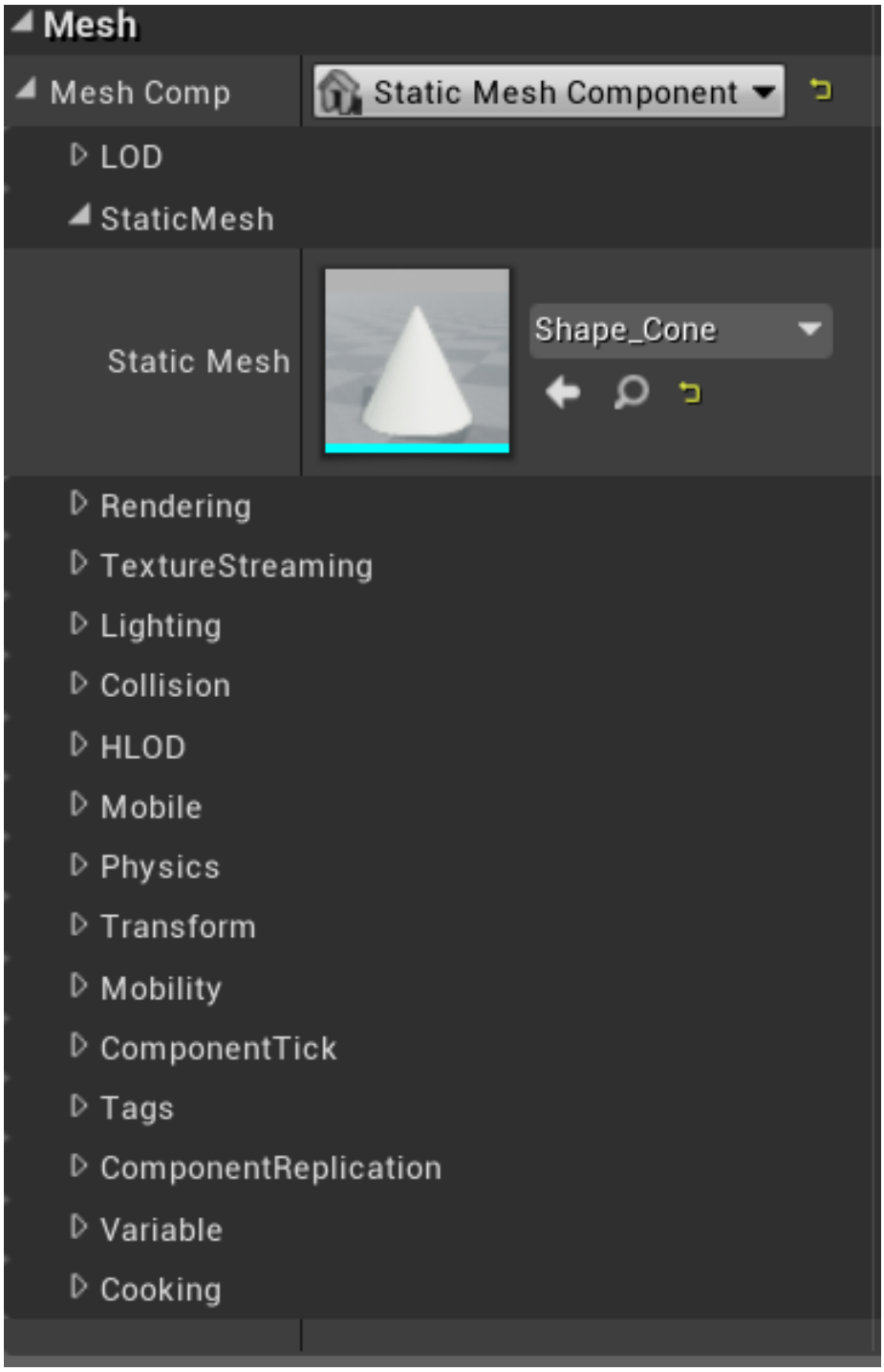

Figure 3: Blueprint Function

Annotation

If the object gets down scaled it still gets bigger, but with negative values, which also changes the direction of the rotation. This could be stopped by limiting the down scaling but is for now left like it is.
The code for ATriggerBox is also pretty much the same as the code would be for a UComponentBox, the only difference is the access to the scale object.