ДОДАТОК А

СЛАЙДИ ПРЕЗЕНТАЦІЇ


10TP
Міністерство освіти і науки України
Харківський національний університет
радіоелектроніки
АТЕСТАЦІЙНА РОБОТА МАГІСТРА
Дослідження методів пересування танка через
місцевість в ролі механіка водія на симуляторі
віртуальної реальності.
Керівник проекту
доцент каф. ІП
Назаров О.С.
Виконав
Студент групи ІПЗм-18-3
Даниленко М.О.
1


10TP
Мета роботи
Об'єкт розробки – танковий симулятор у вигляді ігрового додатка до віртуальної реальності
Мета розробки – дослідження методу пересування танку та стабілізації танку через місцевість до симулятору віртуальної реальності
Метод рішення – Windows, HTC Vive або Oculus Rift, скриптова мова Blueprint та C++.
2

Постановка задачі

1.Проаналізувати предметну область

2.Дослідити сили що впливають на підвіску під час руху танку

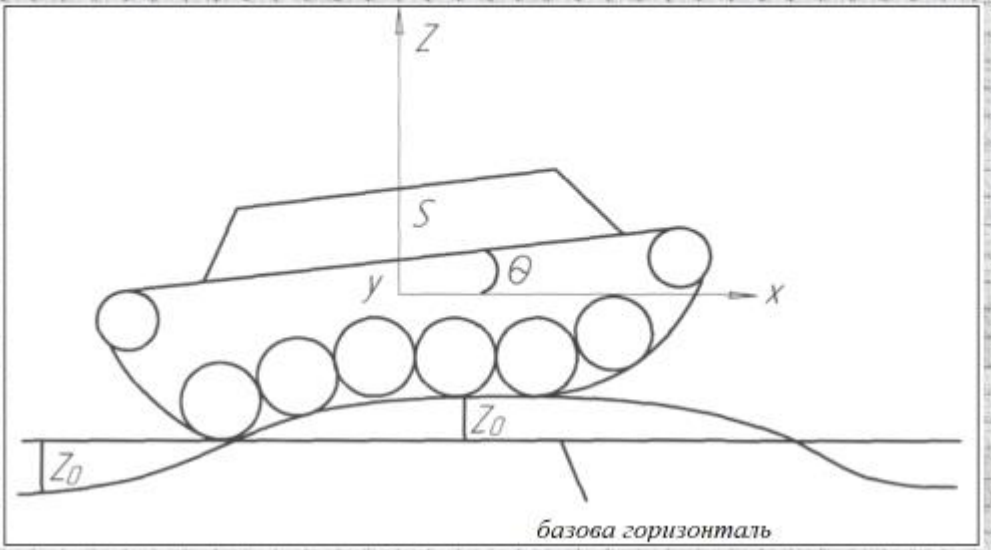
3.Розробити програмне забезпечення

4.Протестувати розроблене програмне забезпечення

3

Схема рухомого об’єкта

Координати об’єкта при пересуванні



4

Рівняння Лангранджу II рівня які описують рухомий об'єкт:

- Перша ступінь свободи: $\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{Z}}\right) - \frac{\partial T}{\partial Z} = Q_z$,

- Друга ступінь свободи: $\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\theta}}\right) - \frac{\partial T}{\partial \theta} = Q_\theta$,

- Третя ступінь свободи: $\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\psi}}\right) - \frac{\partial T}{\partial \psi} = Q_\psi$.

5

Граничні режими руху багатоцільовий гусеничної платформи

Рівняння кінетичної енергії машини $\frac{m\left[V_S^{(в)}\right]^2}{2} = \frac{c^{(в)}\delta^2}{2}$,

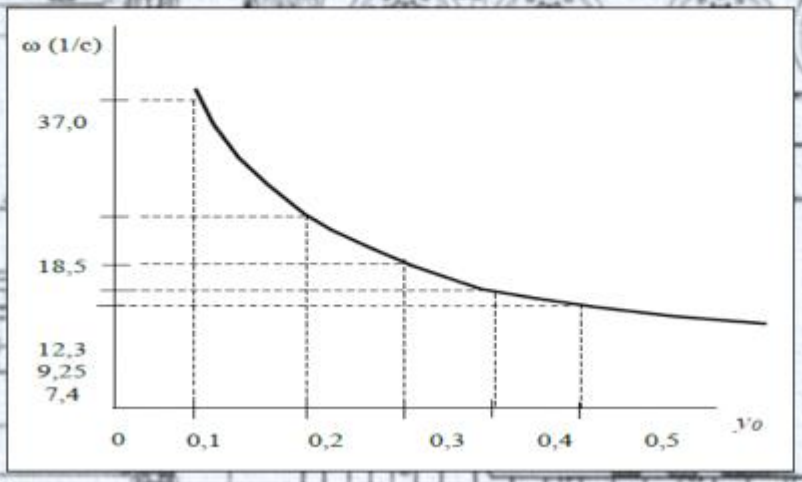
Підресорена маса рухомого об'єкта $V_S^{(в)}$,

Вертикальна жорсткість всієї підвіски $C^{(в)}$

Якщо вирішити рівняння кінетичної енергії то отримаємо $\omega = \frac{V_S^{(в)}}{y_0}$,

6

Залежність амплітудного відхилення профілю дороги та кругової частоти



7

Масив пов'язаних значень

$y_0 = 0.5$ м.	$\omega = 7.4$ 1/с ,	$\tau = 0.84$ с;
$y_0 = 0.4$ м.	$\omega = 9.25$ 1/с ,	$\tau = 0.67$ с;
$y_0 = 0.3$ м.	$\omega = 12.3$ 1/с ,	$\tau = 0.51$ с;
$y_0 = 0.2$ м.	$\omega = 18.5$ 1/с ,	$\tau = 0.34$ с;
$y_0 = 0.1$ м.	$\omega = 37$ 1/с ,	$\tau = 0.17$ с.

8

Граничні значення швидкості руху об'єкту

y_0=0.5м	V	70	60	50	40	30	20*	10
	λ	16,8	13,9	11,5	9,32	6,99	4,66	1,7
y_0=0.4м	V	70	60	50	40	30	20	
	λ	12,9	11,1	9,24	7,4	5,58	3,7	
y_0=0.3м	V	70	60	50	40*	30		
	λ	9,8	8,4	7,03	5,66	4,2		
y_0=0.2м	V	70	60	50*				
	λ	6,59	5,66	4,6				
y_0=0.1м	V	70	60					
	λ	3,33	2,82					

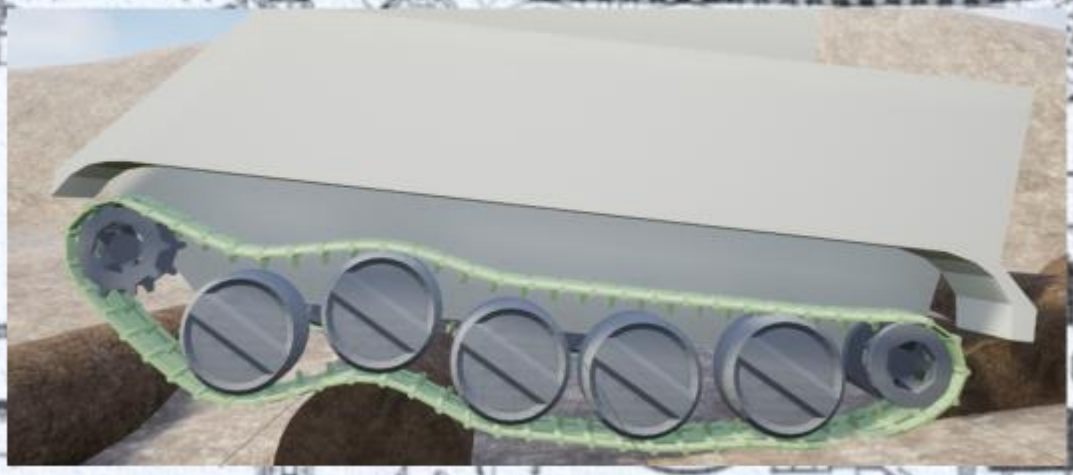
де V – швидкість

де y_0 – *амплітудне відхилення профілю від базової горизонталі*

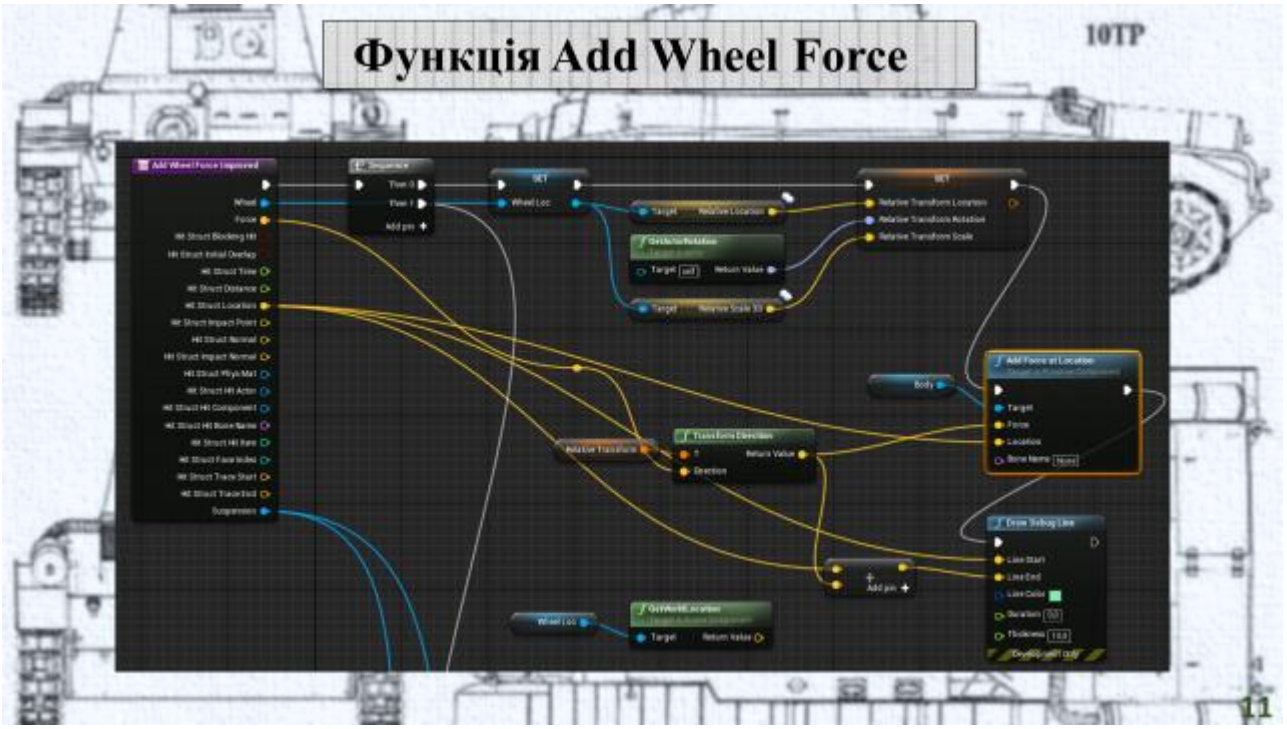
де λ – *довжина хвилі*

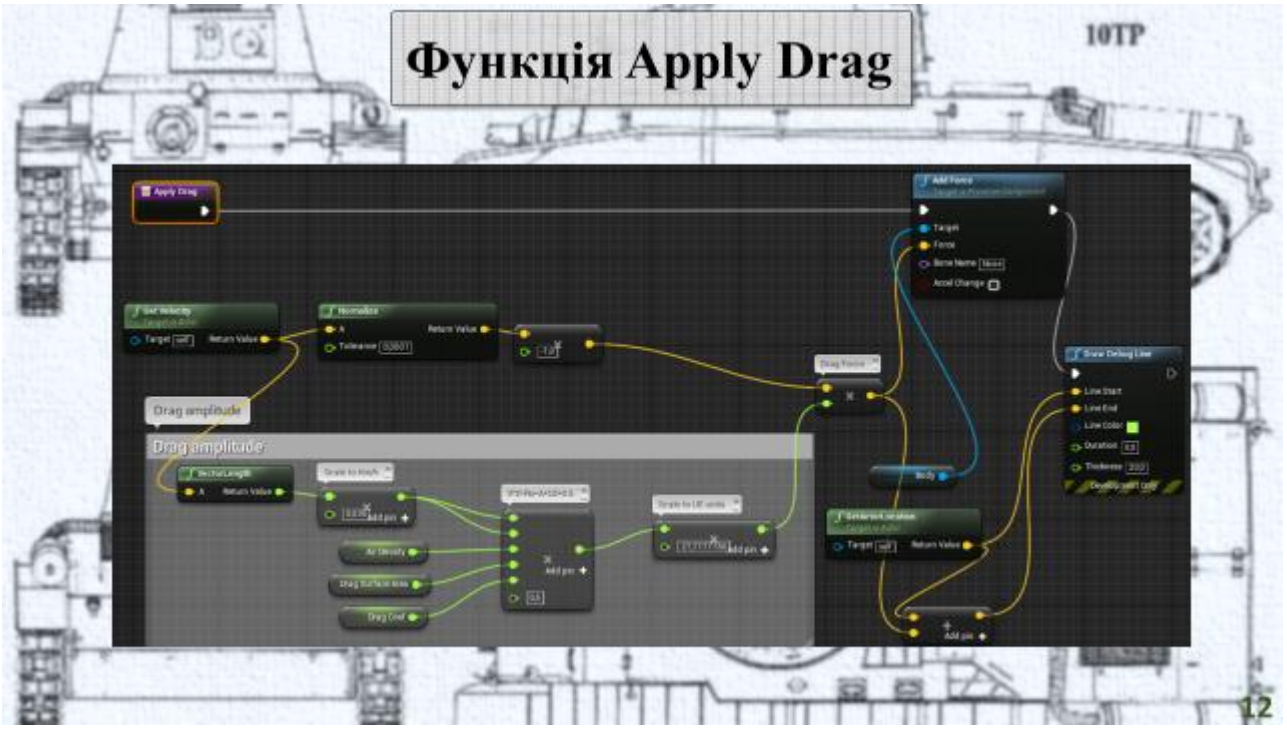
9

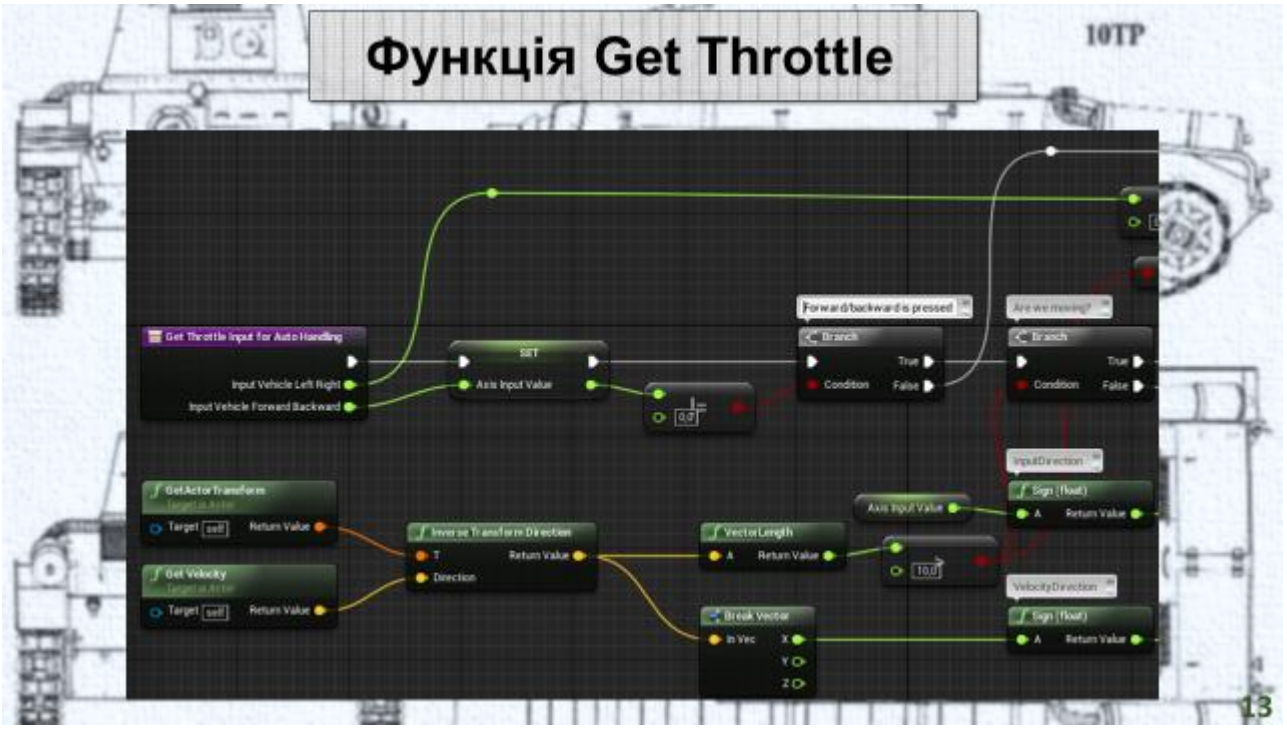
Приклад роботи підвіски у програмному додатку

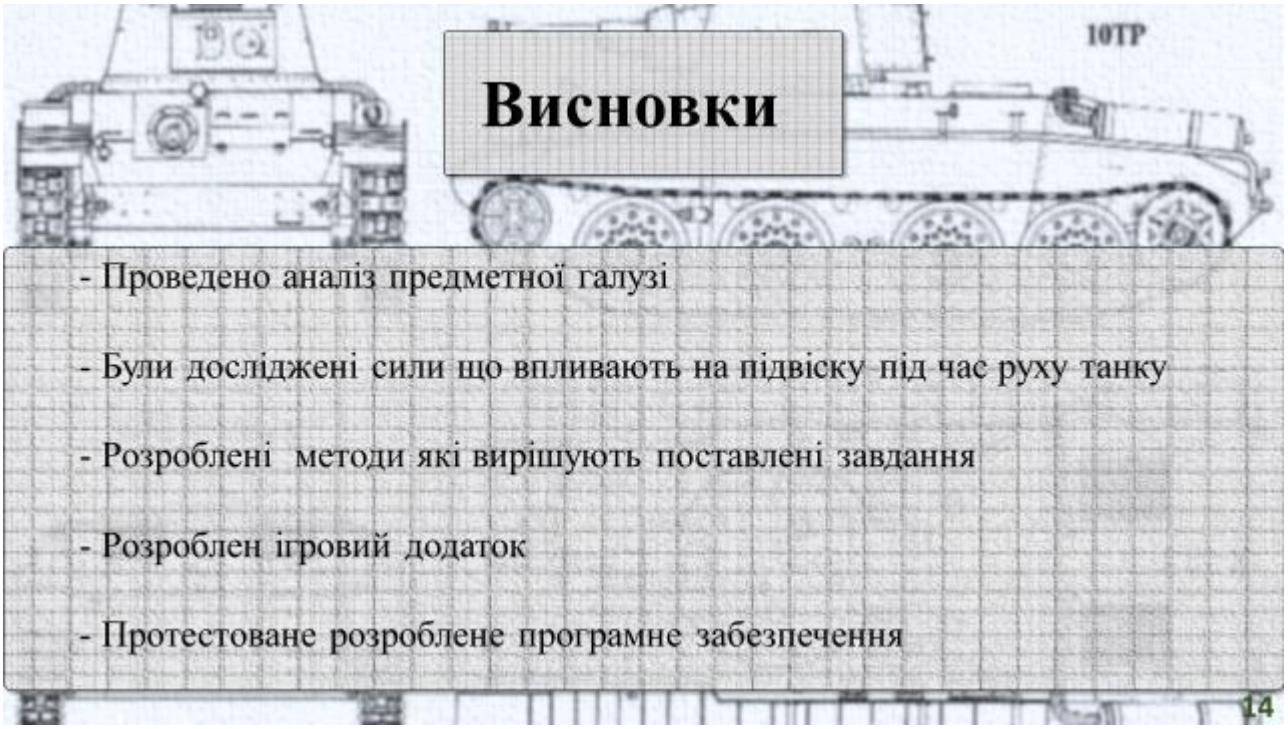


10


Функція Add Wheel Force
10TP
11


Функція Apply Drag
10TP
Drag amplitude
12


Функція Get Throttle
10TP
Get Throttle Input for Auto Handling
Input Vehicle Left Right
Input Vehicle Forward Backward
SET
Axis Input Value
Forward/backward is pressed!
Branch
True
False
Condition
Are we moving?
Get Actor Transform
Target
self
Return Value
Get Velocity
Inverse Transform Direction
T
Direction
Vector Length
A
Axis Input Value
Break Vector
In Vec
X
Y
Z
Sign (float)
13


Висновки
10TP
- Проведено аналіз предметної галузі
- Були досліджені сили що впливають на підвіску під час руху танку
- Розроблені методи які вирішують поставлені завдання
- Розроблен ігровий додаток
- Протестоване розроблене програмне забезпечення
14


10TP
Дякую за увагу!
15

ДОДАТОК Б

КОД ІГРОВОГО ДОДАТКУ

```
UMMTSuspensionStack::UMMTSuspensionStack()
{
	//Bind async trace delegate
	//TraceDelegate.BindUObject(this,
&UMMTSuspensionStack::AsyncTraceDone);
	SprungComponentName = FString("Root");
}

void UMMTSuspensionStack::Initialize()
{
	//Initialize variables again, just in case!
	bContactPointActive = false;
	ContactInducedVelocity = FVector::ZeroVector;
	ContactForceAtPoint = FVector::ZeroVector;
	ContactPointLocation = FVector::ZeroVector;
	ContactPointNormal = FVector::UpVector;
	bSprungMeshComponentSetManually = false;
	bSweepShapeMeshComponentSetManually = false;
	ComponentName = FString("ComponentRefereneFailed");
	ComponentsParentName = FString("ParentRefereneFailed");
	bWarningMessageDisplayed = false;
	WheelHubPositionLS = FVector::ZeroVector;
	PreviousSpringLenght = 1.0f;
	SuspensionForceMagnitude = 0.0f;
	SuspensionForceLS = FVector::ZeroVector;
	SuspensionForceWS = FVector::ZeroVector;
	SuspensionForceScale = 1.0f;


	USceneComponent* TryParent =
CastChecked<USceneComponent>(GetOuter());
	if (IsValid(TryParent))
	{
		ParentComponentRef = TryParent;
```

```
            GetNamesForComponentAndParent();

            GetNamedComponentsReference();

            PreCalculateParameters();

            GetDefaultWheelPosition();


            //Line Trace default query parameters, called from here to have
valid reference to parent

            //LineTraceQueryParameters.bTraceAsyncScene = false;

            LineTraceQueryParameters.bTraceComplex = false;

            LineTraceQueryParameters.bReturnFaceIndex = false;

            LineTraceQueryParameters.bReturnPhysicalMaterial = true;

            LineTraceQueryParameters.AddIgnoredActor(ParentComponentRef-
>GetOwner());


            //Sphere Trace default query parameters, called from here to
have valid reference to parent

            //SphereTraceQueryParameters.bTraceAsyncScene = false;

            SphereTraceQueryParameters.bTraceComplex = false;

            SphereTraceQueryParameters.bReturnFaceIndex = false;

            SphereTraceQueryParameters.bReturnPhysicalMaterial = true;

            SphereTraceQueryParameters.AddIgnoredActor(ParentComponentRef-
>GetOwner());


            //Shape Sweep default query parameters, called from here to have
valid reference to parent

            //ShapeSweepQueryParameters.bTraceAsyncScene = false;

            ShapeSweepQueryParameters.bTraceComplex = false;

            ShapeSweepQueryParameters.bReturnFaceIndex = false;

            ShapeSweepQueryParameters.bReturnPhysicalMaterial = true;

            ShapeSweepQueryParameters.AddIgnoredActor(ParentComponentRef-
>GetOwner());


      }

      else

      {

            //Disable component to avoid potential null point reference
```

```
            SuspensionSettings.bDisabled = true;


            GEngine->AddOnScreenDebugMessage(-1, 15.0f, FColor::Red,
FString::Printf(TEXT("%Inner Suspension Stack object failed to receive
correct parent reference")));
            UE_LOG(LogTemp, Warning, TEXT("Inner Suspension Stack object
failed to receive correct parent reference"));
      }


      bContactPointActive = false;
}



void UMMTSuspensionStack::GetNamesForComponentAndParent()
{
            //Get names of component and its parent
            ComponentName = ParentComponentRef->GetName();
            ComponentsParentName = ParentComponentRef->GetOwner()-
>GetName();
}


//Find and store reference to named components
void UMMTSuspensionStack::GetNamedComponentsReference()
{
      //Sprung mesh reference
      if (!bSprungMeshComponentSetManually)
      {
            if (SprungComponentName != FString("none"))
            {
                  SprungMeshComponent =
UMMTBPFunctionLibrary::GetMeshComponentReferenceByName(ParentComponentRef,
SprungComponentName);
                  if (!IsValid(SprungMeshComponent))
                  {
                        GEngine->AddOnScreenDebugMessage(-1, 15.0f,
FColor::Red, FString::Printf(TEXT("%s->%s component failed to find
```

```
component named '%s' or it's not derived from MeshComponent class"),
*ComponentsParentName, *ComponentName, *SprungComponentName));

                    UE_LOG(LogTemp, Warning, TEXT("%s->%s component
failed to find component named '%s' or it's not derived from MeshComponent
class"), *ComponentsParentName, *ComponentName, *SprungComponentName);

               }

          }

          else

          {

               SprungMeshComponent = NULL;

               GEngine->AddOnScreenDebugMessage(-1, 15.0f, FColor::Red,
FString::Printf(TEXT("%s->%s component's SprungComponentName property
shouldn't be 'none', set proper name for effected component"),
*ComponentsParentName, *ComponentName));

               UE_LOG(LogTemp, Warning, TEXT("%s->%s component's
EffectedComponentName property shouldn't be 'none', set proper name for
effected component"), *ComponentsParentName, *ComponentName);

          }

     }


     //ShapeSweep mesh reference

     if (SuspensionSettings.bCanShapeSweep &
!bSweepShapeMeshComponentSetManually)

     {

          if (SuspensionSettings.SweepShapeComponentName !=
FString("none"))

          {

               SweepShapeMeshComponent =
UMMTBPFunctionLibrary::GetMeshComponentReferenceByName(ParentComponentRef,
SuspensionSettings.SweepShapeComponentName);

               if (!IsValid(SweepShapeMeshComponent))

               {

                    GEngine->AddOnScreenDebugMessage(-1, 15.0f,
FColor::Red, FString::Printf(TEXT("%s->%s component failed to find
component named '%s' or it's not derived from MeshComponent class"),
*ComponentsParentName, *ComponentName,
*SuspensionSettings.SweepShapeComponentName));

                    UE_LOG(LogTemp, Warning, TEXT("%s->%s component
failed to find component named '%s' or it's not derived from MeshComponent
class"), *ComponentsParentName, *ComponentName,
*SuspensionSettings.SweepShapeComponentName);
```

```
                }
            }
            else
            {
                SweepShapeMeshComponent = NULL;
                GEngine->AddOnScreenDebugMessage(-1, 15.0f, FColor::Red,
FString::Printf(TEXT("%s->%s component's SweepShapeComponentName property
shouldn't be 'none', set proper name for effected component"),
*ComponentsParentName, *ComponentName));
                UE_LOG(LogTemp, Warning, TEXT("%s->%s component's
EffectedComponentName property shouldn't be 'none', set proper name for
effected component"), *ComponentsParentName, *ComponentName);
            }
        }
}


//Recalculate parameters to save performance
void UMMTSuspensionStack::PreCalculateParameters()
{
    //Shift spring offsets if custom position of the stack is used
    SpringOffsetTopAdjusted = SuspensionSettings.bUseCustomPosition ?
SuspensionSettings.StackLocalPosition + SuspensionSettings.SpringTopOffset
: SuspensionSettings.SpringTopOffset;
    SpringOffsetBottomAdjusted = SuspensionSettings.bUseCustomPosition ?
SuspensionSettings.StackLocalPosition +
SuspensionSettings.SpringBottomOffset :
SuspensionSettings.SpringBottomOffset;


    //Calculate spring direction in local coordinate system
    SpringDirectionLocal = SpringOffsetBottomAdjusted -
SpringOffsetTopAdjusted;
    SpringMaxLenght = SpringDirectionLocal.Size();


    //Normalize vector and check if normalization was successful
    if (!SpringDirectionLocal.Normalize())
    {
        SpringDirectionLocal = FVector::UpVector;
```

```
            GEngine->AddOnScreenDebugMessage(-1, 15.0f, FColor::Red,
FString::Printf(TEXT("%s->%s distance between Top and Bottom offsets of the
spring shouldn't be zero"), *ComponentsParentName, *ComponentName));

            UE_LOG(LogTemp, Warning, TEXT("%s->%s distance between Top and
Bottom offsets of the spring shouldn't be zero"), *ComponentsParentName,
*ComponentName);

      }


      //Calculate line trace points in local space, taking into account
road wheel radius and tread thickness

      LineTraceOffsetTopLS = SpringOffsetTopAdjusted + SpringDirectionLocal
* (SuspensionSettings.RoadWheelRadius + SuspensionSettings.TrackThickness);

      LineTraceOffsetBottomLS = SpringOffsetBottomAdjusted +
SpringDirectionLocal * (SuspensionSettings.RoadWheelRadius +
SuspensionSettings.TrackThickness);


      SphereCheckShape =
FCollisionShape::MakeSphere(SuspensionSettings.RoadWheelRadius +
SuspensionSettings.TrackThickness);


}



//Updates position of the road-wheel, calculates and applies spring force
to sprung component

void UMMTSuspensionStack::PhysicsUpdate(const float& DeltaTime)

{

      if (!SuspensionSettings.bDisabled)

      {

            //Update world space transform of parent component

            ReferenceFrameTransform =
UMMTBPFunctionLibrary::MMTGetTransformComponent(ParentComponentRef,
NAME_None);


            UpdateWheelHubPosition();


            CalculateAndApplySuspensionForce(DeltaTime);

      }

}
```

```
void UMMTSuspensionStack::LineTraceForContact()
{
	//Clean results
	FHitResult LineTraceOutHit = FHitResult(ForceInit);

	//Transform points into world space using component's transform
	FVector LineTraceStart = 
ReferenceFrameTransform.TransformPosition(LineTraceOffsetTopLS);
	FVector LineTraceEnd = 
ReferenceFrameTransform.TransformPosition(LineTraceOffsetBottomLS);

	//Do line trace
	bContactPointActive = ParentComponentRef->GetWorld()-
>LineTraceSingleByChannel(LineTraceOutHit, LineTraceStart, LineTraceEnd, 
SuspensionSettings.RayCheckTraceChannel,
		LineTraceQueryParameters, LineTraceResponseParameters);
		//ParentComponentRef->GetWorld()-
>LineTraceSingleByChannel(LineTraceOutHit, LineTraceStart, LineTraceEnd, 
SuspensionSettings.RayCheckTraceChannel,
			//			LineTraceQueryParameters, 
LineTraceResponseParameters);

	//Dirty but assumption is that contact information is never used if 
bContactPointActive is false.
	if (bContactPointActive)
	{
		ContactPointLocation = LineTraceOutHit.ImpactPoint;
		ContactPointNormal = LineTraceOutHit.ImpactNormal;
		ContactPhysicalMaterial = LineTraceOutHit.PhysMaterial.Get();

		if (SuspensionSettings.bGetContactBodyVelocity)
		{
			if (LineTraceOutHit.Component->IsSimulatingPhysics())
			{
```

```
                        ContactInducedVelocity = LineTraceOutHit.Component-
>GetPhysicsLinearVelocityAtPoint(ContactPointLocation);
                }
                else
                {
                        ContactInducedVelocity = LineTraceOutHit.Component-
>ComponentVelocity;
                }
            }
      }

      //Draw debug
      if (SuspensionSettings.bEnableDebugMode)
      {
            DrawDebugLineTrace(bContactPointActive, LineTraceStart,
LineTraceEnd, LineTraceOutHit.ImpactPoint, ParentComponentRef->GetWorld());
      }
}
```