Математичка гимназија

МАТУРСКИ РАД

- из рачунарства и информатике -

Тркачка игра покренута Unreal енџином

Ученик:
Марко Радосављевић
IVц

Ментор:
Филип Хаџић

Београд, јун 2022.

Садржај

1

Увод

Unreal engine је пакет алата за креирање и развој игара, архитектуру и креирање симулација који развија компанија Epic Games . Први пут је приказан 1998. године у игри Unreal по којој је и добио име. Написан је у C++ програмском језику што му омогућава велику флексибилност и могућност за креирање апликација за скоро сваку платформу. Поред C++а могуће је и прављење графичких скрипти Blueprints , које је често брже, али креатору даје мање контроле и често има лош утицај на перформансе.

Љубав према видо играма привукла ме је да кренем да се бавим креирањем игрица и већ неко време користим Unreal за своје пројекте. Пројекат који сам представио у овом раду је тркачка игрица коју сам ја назвао Заклопача мотоспорт . Игра има два мода тркачки и дрифт.

Од помоћних програма користио сам Blender за израду 3D модела и Visual Studio за куцање кода и компајловање.

Игра има једноставне контроле, W, A, S, D за кретање лево, десно, напред и назад, Shift за ручну кочницу и Space bar за Nitro . Такође постоје и многа графичка подешавања која се могу активирати путем конзоле. Конзола се активира "∼" дугметом , а ово су неки примери функција:

r.ScreenPercentage <X> - AMD FSR технологија која рендерује слику на нижој резолуцији а затим је повећава на резолуцију монитора

stat fps - Исписује број фрејмова по секунди

t.MaxFPS <X> - Ограничава FPS

r.SetRES <A>x<B> - Мења резолуцију

2

Уводни појмови

2.1 PhysX

PhysX је open-source физички енџин који је у почетку развијала Ageia , а касније Nvidia. Првобитна замисао била је да поред графичке картице постоји још једна картица која ће помагати процесору да врши физичке прорачуне, али данас се тај чип налази у графичким картицама и служи за хардверску акселерацију у многим другим програминам. PhysX је написан у C++у и данас се налази у свим модерним енџинима.

2.2 Графички API

Графички API је је скуп команди и библиотека које комуницирају са графичком картицом при креирању 2D и 3D апликација. Unreal engine подржава DirectX (или Direct3D) и Vulkan. DirectX је специјализован само за Windows и не може се користити на другим платформама за разлику од Vulkan-а. Ја сам за свој пројекат ипак изабрао DirectX 12 јер сам са њим најбоље упознат и на Windows рачунарима даје мало боље перформансе.

3

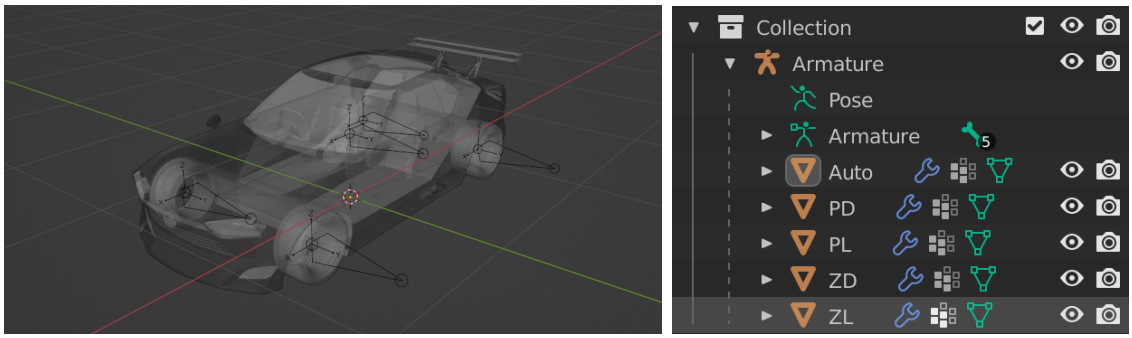
Графика

3.1 Блендер

Блендер је софтверски сет алата за 3D компјутерску графику који се користи за креирање визуелних ефеката, уметничких дела, 3D модела и видео игрица. Карактеристике Блендера укључују 3D моделовање, UV мапирање, текстурирање, дигитално цртање, уређивање графике, симулацију флуида и дима и рендеровање. Изабрао сам да радим у њему јер је израда модела релативно лака и зато што могу да експортујем FBX фајлове јер Unreal једино њих подрзава.

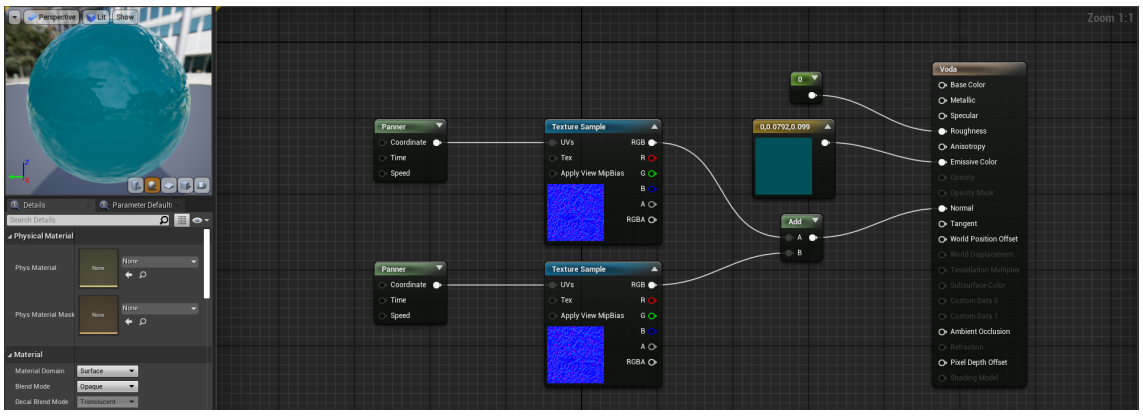
3.2 3D модели

За статичке моделе процес израде је једноставан тј. потребно је само направити фигуру онога што желимо. Међутим за неке покретне моделе које имају више различитих покретних делова потребно је направити посебан облик за сваки тај део. Затим је потребно направити скелет модела, за свако засебно тело потребна је барем једна коска. У случају аутомобила потребне су коске за сваки точак и тело аутомобила. Коска за тело аутомобила треба бити постављена на позитивном делу Z осе (изнад земље) јер ће она касније одређивати координате аутомобила, а коске точкова треба поставити у центру точкова како би се точкови ротирали око њих. Да бих себи касније олакшао програмирање аутомобила све коске окренуо сам према Y оси. На крају потребно је направити vertex групе тј. упарити одговатајуће делове аутомобила са одговарајуим делом скелета као што је приказано на сликама испод. Пре импортовања у енџин потребно само још одредити орјентацију модела (дакле у мом сличају X оса је напред, а Z оса је горе).



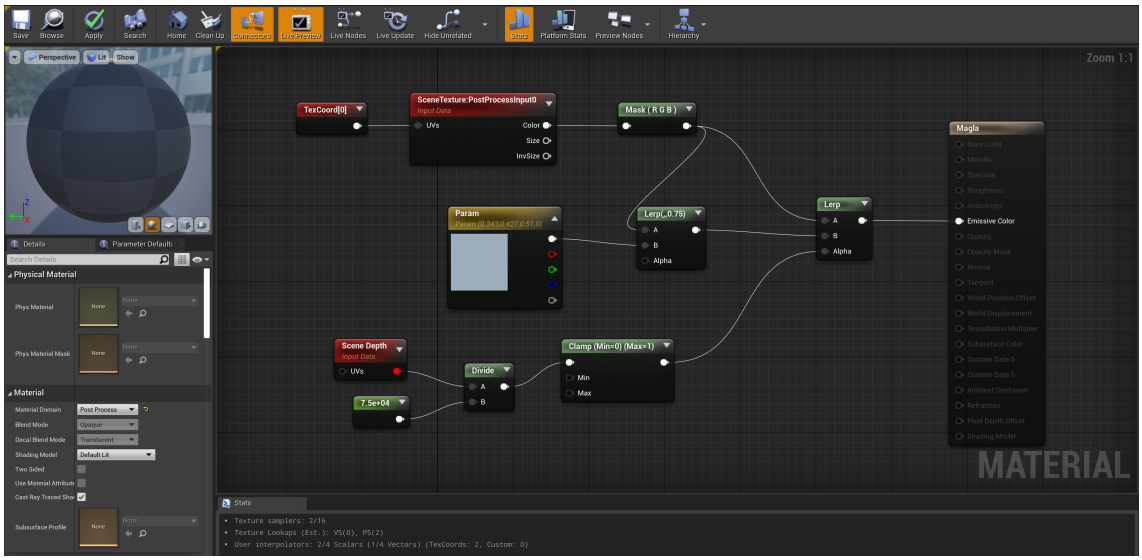
3.3 Текстуре

Текстуре је могуће правити на два начина. Први је директно у Блендеру, али његова мана је то што Unreal често не препозна текстуре из Блендера. Зато сам ја своје текстуре правио у енџину, а предности су такође што се могу правити интерактивне или тексуре које реагују на неки догађај који се дешава у игри. Ово је један од делова који може веома утицати на захтевнос игре тј. количине видео меморије коју игра заузима и зато је потребно обратити пажњу на резолуцију слика које се користе и број инструкција за стварање текстуре. На слици испод налази се пример текстуре воде из игре.



3.4 Сенчење

Неке ефекте попут магле је немогуће представити обичним текстурама и зато постоји shader. Глобални shader се приказују помоћу статичких модела (само у неком одређеном простору) и не мора да се повезује са материјалима. У свакој тачки могуће је приказивати само један shader. Примери би били филтрирање сенки или post processing.



4

Механика

4.1 Аутомобили

У енџину постоји више дефинисаних профила за различите акторе нпр. за човека, возило... Они помажу у повезивању нашег скелета са облицима које праве сударе, а ја сам наравно користио онај за возила и зато је пре свега потребно убацити библиотеку "Actor.h" . Затим је потребно направити нове библиотеке које сам ја назвао "Auto.h", "prednjiTocak.h", "zadnjiTocak.h"и "Tocak.cpp".

"Auto.cpp" ће одређивати брзину, скретање, кочење... На почетку повезао сам точкове и тело аутомобила са скелетом на следећи начин:

```
auto->WheelSetups[0].WheelClass = prednjiTochak::Tocak();
auto->WheelSetups[0].BoneName = FName("PL");
auto->WheelSetups[0].AdditionalOffset = FVector(0.f, -8.f, 0.f);
```

Za tochkove sam koristio PhysX jer су они најбитнији физички делови аутомобила. Ја сам точкове поделио у две библиотеке иако сам могао да ставим све у једну јер ми то даје већу флексибилност да у наставку мењам својства предњих и задњих точкова што је било кључно за дрифт. У наставку сам дао неке примере функција где сам користио PhysX за физику точкова:

```
        //Uzima trenutne podatke o automobilu
#if WITH_PHYSX
FPhysXVehicleManager* prednjiTocak::Manager()
{
        UWorld* World = GEngine->GetWorldFromContextObject(auto,
        EGetWorldErrorMode::LogAndReturnNull);
```

```
        return World ? FPhysXVehicleManager::GetVehicleManagerFromScene(
        World->GetPhysicsScene()) : nullptr;
}
#endif

        //Skretanje
float prednjiTocak::Skreni()
{
#if WITH_PHYSX
if (FPhysXVehicleManager* manager = Manager())
        {
        SCOPED_SCENE_READ_LOCK(manager->GetScene());
        return FMath::RadiansToDegrees(manager->GetWheelsStates_AssumesLocked(
        auto[brTocka].ugaoSkretanja));
        }
#endif
        return 0.0f;
}

        //Okretanje tocka
float prednjiTocak::Rotiraj()
{
#if WITH_PHYSX
if (FPhysXVehicleManager* manager = Manager())
        {
        SCOPED_SCENE_READ_LOCK(manager->GetScene());

        float ugao = -30.0f * FMath::RadiansToDegrees(
        auto->PVehicle->mWheelsDynData.getRotationAngle.pitch(brTocka));
        return ugao;
        }
#endif
        return 0.0f;
}

        //Ako auto dodiruje zemlju ne moze da ubrzava ni da skrece
bool prednjiTocak::DaLiDodirujeZemlju()
{
#if WITH_PHYSX
if (FPhysXVehicleManager* manager = Manager())
        {
```

```
        SCOPED_SCENE_READ_LOCK(manager->GetScene());

        return manager->GetWheelsStates_AssumesLocked(auto
        [brTocka].isInAir);
        }
#endif
        return false;
}

        //Brzina
void prednjiTocak::Tick(float Vreme)
{
        UWorldLocation Location1 = Location;
        Location= GetPhysicsLocation();
        float brzina = (Location - Location1) /Vreme;
}
```

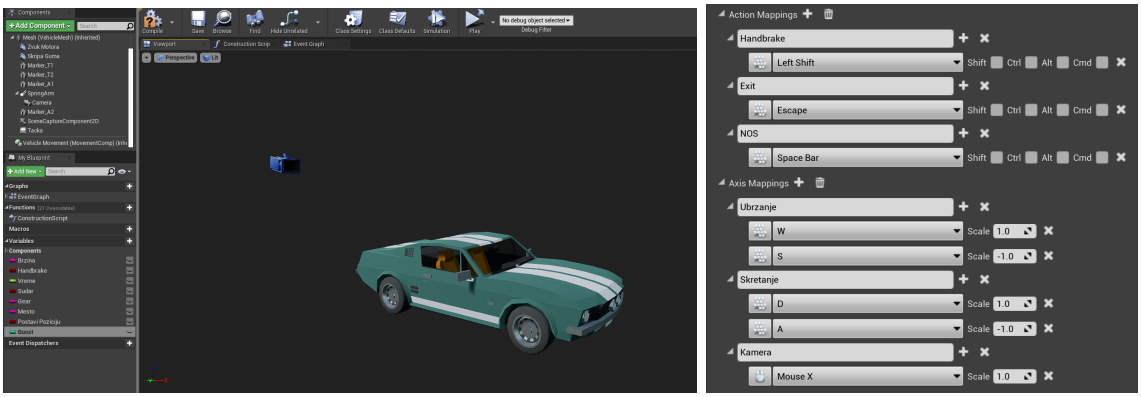
Затим треба поставити камеру и направити звук аутомобила:

```
        // Ruka oko koje se kamera okrece
SpringArm = CreateDefaultSubobject<USpringArmComponent>(TEXT("SpringArm"));
SpringArm->SetRelativeLocation(FVector(0.0f, 0.0f, 0.0f));
SpringArm->SetWorldRotation(FRotator(0.0f, 20.0f, 0.0f));
SpringArm->SetupAttachment(RootComponent);
SpringArm->TargetArmLength = 125.0f;
SpringArm->bEnableCameraLag = true;
SpringArm->bEnableCameraRotationLag = true;
SpringArm->bInheritPitch = false;
SpringArm->bInheritYaw = true;
SpringArm->bInheritRoll = false;

        // Kamera
Camera = CreateDefaultSubobject<UCameraComponent>(TEXT("Camera"));
Camera->SetupAttachment(SpringArm, USpringArmComponent::SocketName);
Camera->SetRelativeLocation(FVector(-150.0, 0.0f, 25.0f));
Camera->SetRelativeRotation(FRotator(00.0f, 0.0f, 0.0f));
Camera->bUsePawnControlRotation = true;
Camera->bUsePawnControlRotationLag = 10.f;
Camera->FieldOfView = 90.f;

        //Zvuk
```

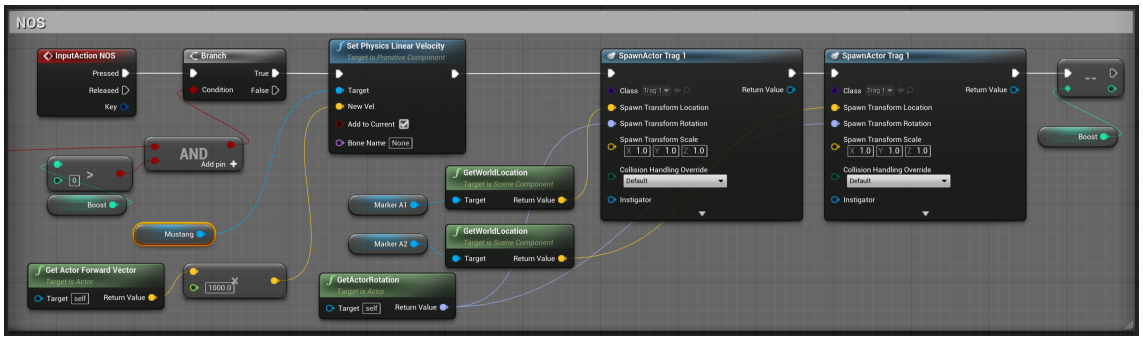
```
static ConstructorHelpers::FObjectFinder<USoundCue>
SoundCue(TEXT("/Game/Automobili/Zvuk/Motor"));
SoundComponent = CreateDefaultSubobject<UAudioComponent>
(TEXT("ZvukMotora"));
SoundComponent->SetSound(SoundCue.Object);
SoundComponent->SetupAttachment(GetMesh());
```

Да би физика аутомобила била што реалистичнија неке промењиве попут убразња (тј. времена између којег се точак окрене) или скретања морају да зависе од неких других фактора попут тренутне брзине. Што је тренутна брзина већа угао скретања ће бити мањи. То постижемо на следећи начин:

```
auto->Skretanje.GetRichCurve()->Reset();
auto->Skretanje.GetRichCurve()->AddKey(0.0f, 1.0f);
auto->Skreatnje.GetRichCurve()->AddKey(40.0f, 0.9f);
auto->Skretanje.GetRichCurve()->AddKey(120.0f, 0.8f);
```



Након компајловања у енџину ћемо добити ауто и последњи корак је креирање контрола. Пошто су оне исте за сваки ауто нећемо их дефинисати за сваки посебно већ за цео пројекат и користити Eventh Graph за њихово позивање.



4.2 Ботови

Постоји више начина да се направе ботови аутомобила. Први је са ласерском детекцијом растојања између аутомобила и других објеката, дакле аутомобил је свестан своје позиције у простору, може да детектује друге објекте и да на основу њих скреће, убрзава или кочи. Разлог због којег ја нисам одабрао овај метод је то што је потребно јако пуно времена како би се ботови истренирали. Метода коју сам ја користио је далеко једноставнија тј. аутомобили се само крећу од тачке до тачке. Да се аутомобили не би сваки пут кретали истом путањом ауто бира једну од три различите тачке на путу.

```
float AI::skreci(AActor * spline)
{
        UWorldLocation L1 = GetPhysicsLocation();
        UWorldLocation L2 = spline[i].FindSpotClosestToWorldLocation(L1)*1000/0.0
        UWorldLocation L3 = FindLocationClosestToWorldLocation(L1+L2);
        UWorldRotation R1 = GetWorldRotation();
        UWorldRotation R = FindLookAtRotation(L3,L1);

        float a1 = R1.yaw * R2.yaw + R1.pitch * R2.pitch + R1.roll * R2.roll;
        float a2 = sqrt(R1.yaw*R1.yaw + R1.pitch*R1.pitch + R1.roll*R1.roll);
        float a3 = sqrt(R2.yaw*R2.yaw + R2.pitch*R2.pitch + R2.roll*R2.roll);

        float ugao= acos( a1 / (a2 * a3));

        if (ugao>AI->Skretanje.GetRichCurev()->Get(Brzina))
        AI.Vreme = AI.Vreme + 0.01f; \\vreme obrtanja tochka se povecava

        spine.reset();
        return ugao;
}

void AI::Vozi()
{
        srand (time(NULL));
        int x;
        GetAllActorsOfClass(put.spline[x]);
        for(int i = 0, i<100, i++)
        {
                x = rand() % 3 + 1;
                ugao = skreci(&put.spline[x][i]);
                Vreme = 1*cos(ugao);
        }
}
```

4.3 Тригери

Тригери су актери који се користе да изазову догађај када су у интеракцији са неким другим објектом на нивоу. Другим речима, користе се за покретање догађаја као одговор на неку другу акцију на нивоу. Сви подразумевани тригери

су генерално исти, разликују се само у облику области утицаја (кутије, капсуле и сфере...) које користи да открије да ли га је други објекат активирао. Примери тригера у мојој игри су вода која у контакту са аутомобилом завршава трку и дисквалификује играча или циљ који броји позицију и завршава трку.

```
void KrajTriggerBox::OnOverlapBegin(class AActor* OvActor, class AActor* Actor)
{
    if (OvActor == Actor)
        {
                Sleep(1000);
                SetPause(Trka);
                int pozicija = Actor.Mesto;
                UUserWidget* Kraj = Cast<KrajWidget>;
                Kraj->AddToViewport();
                Sleep(2000);
                UGameplayStatics::OpenLevel(GetWorld(), FName('menu'));
        }
}
```

4.4 Дрифт

Аутомобил за дрифт по физици се највише разликује од свих осталих. Првобитна идеја била је да симулирам ауто који има погон на задњим точковима, задњи точкови се окрећу, а предњи само клизају, али тако нисам успео да направим ауто којим се може лако направити дрифт. Зато сам морао да симулирам суспензију дрифт аутомобила из правог живота. Дакле кад ауто не скреће оба точка ће равномерно додиривати подлогу, али када се точкови окрену на пример на леву страну десни точак ће се подићи и уз помоћ функције коју сам раније дефинисао за точкове DaLiDodirujeZemlju мој ауто ће се окретати на страну.

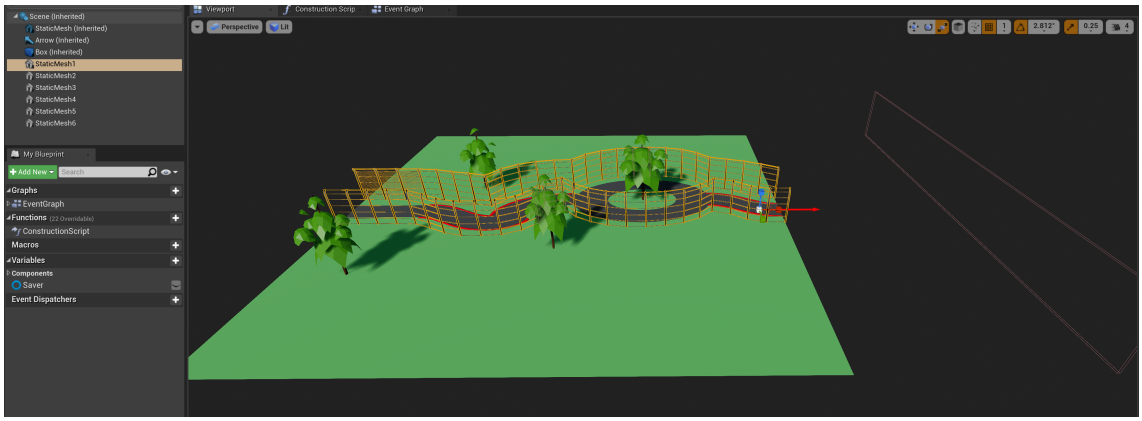
Друга занимљива ствар је како рачунам број бодова и како уопште детектујем дрифт. За аутомобил имам два везана вектора, први је увек усмерен у правцу у којем је усмерен и аутомобил, а други је усмерен у правцу кретања аутомобила. Дрифт настаје када та два вектора немају исти правац. У правом животу постоје три параметра која одређују успешан дрифт брзина, угао и линија по којој се ауто креће. Број бодаова рачунао сам по формули:

$Brzina/100 * sin(\alpha)$

И сабирао сам га са претходним бројем бодова на сваких 0.01 секунд.

4.5 Мапа за дрифт

Мапа за дрифт разликује се од обичне мапе јер се сама генерише од неколико различитих делова и нема крај. То се постиже уз помоћ тригера који се налази на крају сваког сегмента и са пролазом кроз тригер додаје се нови део. Локацију новог дела одређује вектор који се налази на на крају претходног сегмента. Такође је битно да ограда и пут буду два различита модела да би ограду касније користио као тригер тј. када се аутомобил судари са оградом игра се завршава.



Дакле на почетку треба дефинисати тригер за постављење следећег сегмента:

```
void Seg::OnEndTriggerOverlap(AActor* OtherActor)
{
        if (OtherActor)
        {
        AActor Player = Cast<IDX>(OtherActor);
        if (Player)
        {
        AGameModeBase* GameMode = GetWorld()->GetAuthGameMode();
        if (GameMode)
        {
        UGameMode* GameModeDrift = Cast<DriftGameMode>(GameMode);
        if (RunnerGameMode)
        {
        GameModeDrift->DodajSeg();
```

```
        FTimerHandle Delay;
        \\Dodajem sledeci i trenutni brisem za dve sekunde
        GetWorldTimerManager().SetTimer(Delay, this, Destroy(), 2.0f);
        }
        }
        }
        }
}
```

Затим користимо GameMode да на почетку игре поставимо неколико делова без активирања тригера јер се тригер налази испред сегмента, а такође постижемо ефекат који заварава играча да је пут заиста бескрајан.

```
void DriftGameMode::BeginPlay()
{
        Drift::BeginPlay();

        for (int i = 0; i <= 2; i++)
        {
                DodajSeg();
        }
}

void DriftGameMode::DodajSeg()
{
        srand (time(NULL));
        FActorSpawnParameters Spawn = Sledeci;
        AActor* Seg;

        int i = rand() % 3 + 1;

        Seg = GetWorld()->SpawnActor(Seg[i]);

        Sledeci = Seg->GetAttachTransform(Strelica);

}
```

5

Закључак

За крај желео бих да кажем да сам уживао радећи на овом пројекту. Програм има неколико ситних багова али надам се да ће и играчи ове игре уживати у њој. У овом тексту нису приказане баш све функције које сам исписао, јер сам се више пажње посветио изради саме игре. Надам се да сам још некога заинтересовао за прављење оваквих програмчића јер временом како технологија напредује модерни програми чине стварање игара све лакшим и доступнијим свим људима.

Литература

[1] Unreal Engine Documentation https://docs.unrealengine.com/5.0/en-US/
[2] Blender 3.1 Reference Manual https://docs.blender.org/manual/en/latest/