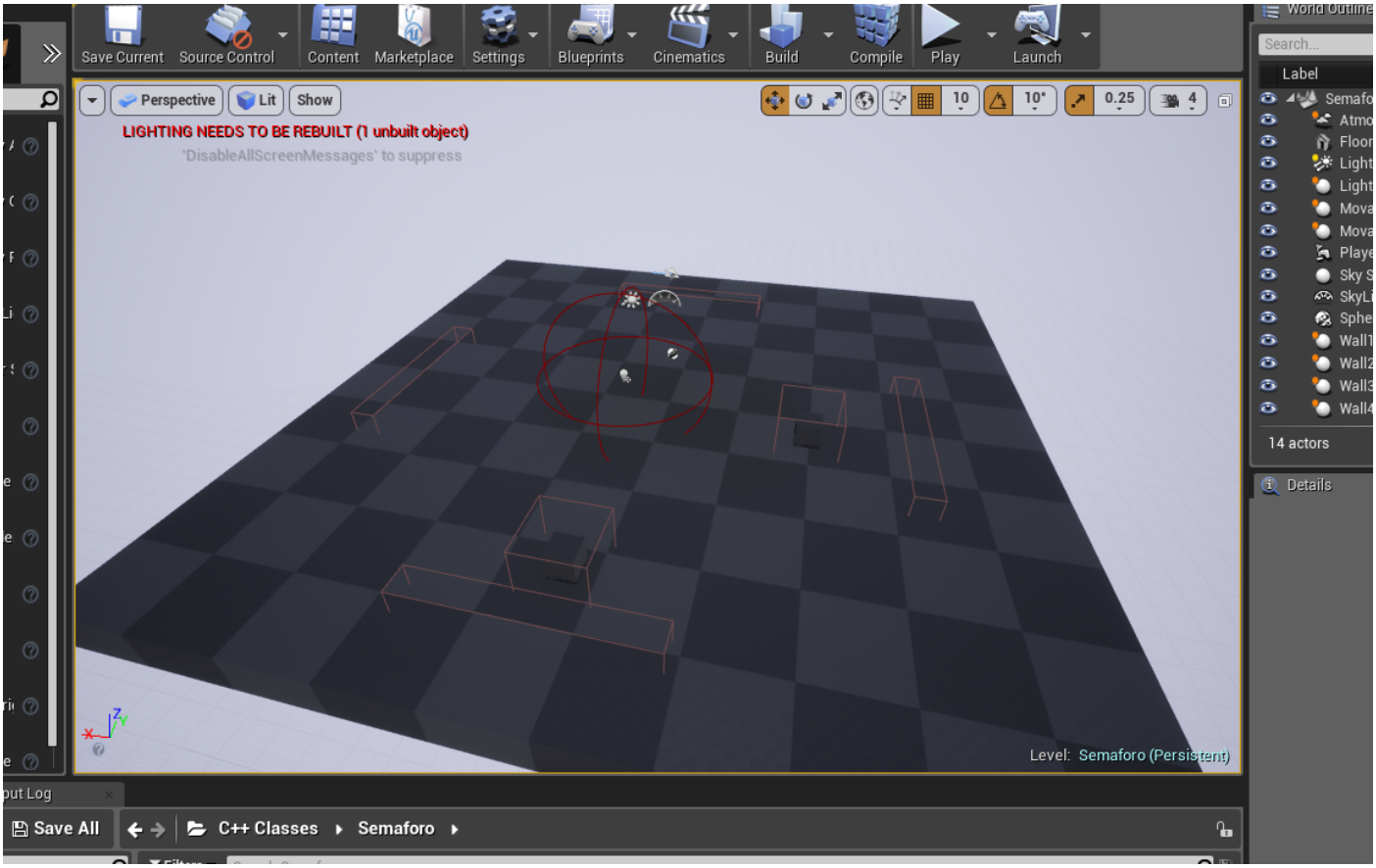
Práctica Semáforos - UE4

Diseño de Pre-Producción y Concept Art.

Víctor García Cortés. Grupo 3.3 de videojuegos. ESNE.



Práctica Semáforos - UE4 1
INTRODUCCIÓN. 3
OBJETIVO Y MOTIVACIÓN. 3
DESCRIPCIÓN TÉCNICA. 3
DIARIO DE DESARROLLO. 10

INTRODUCCIÓN.

Tercera práctica de Diseño de pre-producción y concept art, Videojuegos, desarrollada con Unreal Engine 4.

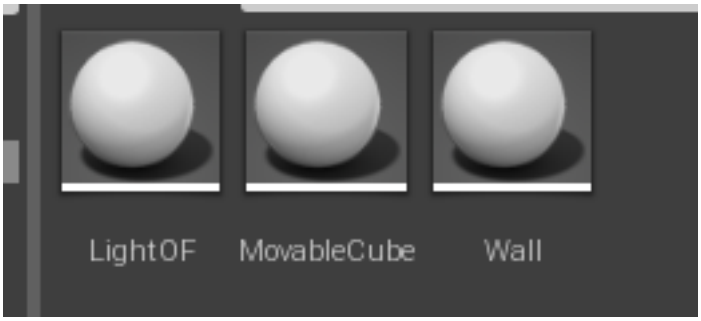
OBJETIVO Y MOTIVACIÓN.

El objetivo de la práctica era la creación de un proyecto en Unreal utilizado el sistema de programación por código (c++). La práctica consiste en realizar un proyecto en cual habrá en escena varias objetos con un movimiento constante de ida y vuelta entre dos puntos y un semáforo en el centro representado por una luz de colores verde y rojo.
El funcionamiento consiste en que cuando los cubos vayan a pasar por el semáforo este controlará el flujo, impidiendo que pasen más de uno a la vez. Si no hay nadie en transición el semáforo emitirá un color verde indicando que se puede pasar, cuando un objeto pasa a través de él, el semáforo cambiará a color rojo, indicando que los demás cubos tendrán que esperar. Cuando el primero cubo haya terminado de pasar, la luz del semáforo volverá a verde y pasará el siguiente cubo.

DESCRIPCIÓN TÉCNICA.

A continuación paso a explicar los componentes del proyecto y su funcionamiento.

Los componentes del del proyecto son 3 C++ Classes: LightOF, MovableCube,



El primer elemento que creé fue el MovableCube. Este tiene la funcionalidad de poder moverse y detectar colisiones ya que le hemos incluido un collider, en forma de UPROPERTY. A parte de esto, MovableCube tiene 3 funciones Displace(), con la cual ponemos a true un booleano mediante el cual indicaremos en AMovableCube::Tick() si se puede mover o no. La siguiente función es Stop() donde pondremos ese mismo boleado a false. Y por último está Reverse() mediante la cual cambiamos el sentido de la dirección del

AMovableCube.h

```cpp
#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Components/CapsuleComponent.h"
#include "LightOF.h"
#include "MovableCube.generated.h"

UCLASS()
class SEMAFORO_API AMovableCube : public AActor
{
    GENERATED_BODY()

public:
/// ALightOF* lightInstance;

    UPROPERTY(EditAnywhere, Category = "velocidad")
        float speed;
    UPROPERTY(EditAnywhere, Category = "Malla")
        class UStaticMeshComponent* mesh;
    UPROPERTY(EditAnywhere, Category = "Colsionador")
        class UBoxComponent* collider;
    UPROPERTY(EditAnywhere, Category = "Trigger")
        class UCapsuleComponent* triggerCapsule;

    // Sets default values for this actor's properties
    AMovableCube();

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

    UFUNCTION()
        void Displace();

    UFUNCTION()
        void Stop();

    UFUNCTION()
        void Reverse();

    /* ... */
private:
    bool ableToMove;

};
```

AMovableCube.cpp

```cpp
#include "MovableCube.h"
#include "Engine.h"
#include "EngineUtils.h"
#include "DrawDebugHelpers.h" //Help us see the colliders

// Sets default values
AMovableCube::AMovableCube()
{
    // Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
    PrimaryActorTick.bCanEverTick = true;

    /// Creamos la malla del objeto y se la asignamos a RootComponent
    mesh = CreateDefaultSubobject<UStaticMeshComponent>("Malla del Objecto");
    RootComponent = mesh;

    /// Creamos el colisionador, le asignamos un tamaño y lo attachamos al RootComponent
    collider = CreateDefaultSubobject<UBoxComponent>("Colisionador");
    collider->SetBoxExtent(FVector(120, 120, 120));
    collider->AttachTo(RootComponent);

    triggerCapsule = CreateDefaultSubobject<UCapsuleComponent>(TEXT("Trigger Capsule"));
    triggerCapsule->InitCapsuleSize(20, 20);
    triggerCapsule->SetupAttachment(RootComponent);

    /*triggerCapsule->OnComponentBeginOverlap.AddDynamic(this, &AMovableCube::OnOverlapBegin);
    triggerCapsule->OnComponentEndOverlap.AddDynamic  (this, &AMovableCube::OnOverlapEnd  );
    */
}
```

```cpp
// Called when the game starts or when spawned
void AMovableCube::BeginPlay()
{
    Super::BeginPlay();
    Displace();
}

// Called every frame
void AMovableCube::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    if (ableToMove)
    {
        SetActorLocation(GetActorLocation() + GetActorForwardVector() * DeltaTime * speed);
    }

}

void AMovableCube::Displace()
{
    ableToMove = true;
}

void AMovableCube::Stop()
{
    ableToMove = false;
}

void AMovableCube::Reverse()
{
    speed = -speed;
}
```

El siguiente elemento es Wall. Como MovableCube también dotamos a este por medio de UPROPERTY de un mes y un collider, también he decidido crear una instancia del objeto MovableCube para detectar cuándo colisiona con este objeto. En Las funciones implementadas hay dos métodos OnOverlapBegin() y OnOverlapEnd() mediante las cuales detectamos si hay algo colisionando y cuándo ha dejado de colisionar. Mediante las UPROPERTY y los UFUNTION detectaremos las colisiones. En BeginPlay() hago que se suscriba al evento de OnOverlapBegin() y OnOverlapEnd(). A continuación en OnOverlapBegin() voy a comprobar aquello con lo que ha comisionado, si el objeto con el que ha comisionado es una instancia del MovableCube, llamaremos a su función Reverse() para que cambie el sentido con el que se mueve, creando el efecto de ida y venida.

AWall.h

```
UCLASS()
class SEMAFORO_API AWall : public AActor
{
    GENERATED_BODY()

public:

    AMovableCube* cubeInstance;

    //Colisionador del objeto
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Collider")
        class UBoxComponent* boxCollider;

    //Malla del objeto
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Mesh")
        class UStaticMeshComponent* mesh;

    // Sets default values for this actor's properties
    AWall();

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

    UFUNCTION()
        void OnOverlapBegin(UPrimitiveComponent* OverlappedComponent,
            AActor* OtherActor,
            UPrimitiveComponent* OtherComponent,
            int32 OterBodyIndex,
            bool bFromSweep,
            const FHitResult &SweepResult);

    UFUNCTION()
        void OnOverlapEnd(class UPrimitiveComponent* OverlappedComp,
            class AActor* OtherActor,
            class UPrimitiveComponent* OtherComp,
            int32 OtherBodyIndex);



};
```

```cpp
// Sets default values
AWall::AWall()
{
    // Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
    PrimaryActorTick.bCanEverTick = true;

    //Crea el componente collider
    boxCollider = CreateDefaultSubobject<UBoxComponent>("Collider");

    //Modifica el tamaño del collider
    boxCollider->SetBoxExtent(FVector(50, 50, 50));

    //Establece la raíz de la jerarquía de componentes
    SetRootComponent(boxCollider);

    //Crea el componente collider
    mesh = CreateDefaultSubobject<UStaticMeshComponent>("Mesh");

    //Modifica la jerarquía de componentes
    mesh->AttachToComponent(boxCollider, FAttachmentTransformRules::SnapToTargetIncludingScale);

}

// Called when the game starts or when spawned
void AWall::BeginPlay()
{
    Super::BeginPlay();
    //Suscripción a los eventos trigger enter
    boxCollider->OnComponentBeginOverlap.AddDynamic(this, &AWall::OnOverlapBegin);

    //Suscripción a los eventos trigger exit
    boxCollider->OnComponentEndOverlap.AddDynamic(this, &AWall::OnOverlapEnd);

}

// Called every frame
void AWall::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
}
```

```cpp
// Called every frame
void AWall::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
}

void AWall::OnOverlapBegin(UPrimitiveComponent * OverlappedComponent, AActor * OtherActor,
                           UPrimitiveComponent * OtherComponent, int32 OterBodyIndex, bool bFromSweep, const FHitResult & SweepResult)
{
    if (OtherActor != nullptr && (OtherActor != this) && OtherComponent != nullptr)
    {
        cubeInstance = Cast<AMovableCube>(OtherActor);

        if (cubeInstance != nullptr)
            cubeInstance->Reverse();
    }
}

void AWall::OnOverlapEnd(UPrimitiveComponent * OverlappedComp, AActor * OtherActor, UPrimitiveComponent * OtherComp, int32 OtherBodyIndex)
{
}
```

El último elemento creado a sido LightOF. Para darle las propiedades de una luz, en la cabecera le añadimos mediante UPROPERTY un UPointLightComponent, USphereComponent y una intensidad. Mediante estas UPROPERTY le damos el componente de luz, un colider de forma esférica y una intensidad de luminiscencia.
Como quiero que emita colores rojo y verde he creado dos struct FLiniearColor para asignarle los colores de verde y rojo. Por último he creado una referencia a MovableCube para comprobar como en wall si ha entrado en colisión la la luz y el cubo.
Dentro de las UFUNCTION, he creado ToggleLight(), ChangeColor(), OnOverlapBegin() y OnOverlapEnd().
El comportamiento de la bombilla es el siguiente, en el constructor le añado todas las propiedades que he creado en la cabecera, haciéndolas visibles en el inspector. Como en Wall hago que LightOF se suscriba a los eventos de OnOverlapBegin() y OnOverlapEnd(), le asigno los colores de creados por medio de parámetro RGB el color rojo y verde a los FLinearColor.
Llamo a la función ToggleVisibility() onBeginPlay() para que luzca siempre y por defecto llamo a la función ChangeColor() pasándole el FColor verde, para que comience en verde.

ALightOF.h

```
UCLASS()
class SEMAFORO_API ALightOF : public AActor
{
    GENERATED_BODY()

public:

    class AMovableCube* AcubeInstance;

    TArray<AMovableCube*> cubos;

    UPROPERTY(EditAnywhere, Category = "Colisionador")
        class UBoxComponent* collider;

    /// Definición de los parámetros de la bombilla
    UPROPERTY(VisibleAnywhere, Category = "Light Switch")
        class UPointLightComponent* pointLight;
    UPROPERTY(VisibleAnywhere, Category = "Light Sphere")
        class USphereComponent* lightSphere;
    UPROPERTY(VisibleAnywhere, Category = "Light Intensity")
        float lightIntensity;


    struct FLinearColor reddd;
    struct FLinearColor greeen;

    bool light_status;
    bool isCollisioning;

    // Sets default values for this actor's properties
    ALightOF();

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;
```

```
            if (i == 0)
                cubos[i]->Displace();

            if(i != 0)
            {
protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

    UFUNCTION()
        void ToggleLight(bool status);

    UFUNCTION()
        void ChangeColor(struct FLinearColor color);

    UFUNCTION()
        void addToList();

    UFUNCTION()
        void elimFromList();


    UFUNCTION()
        void OnOverlapBegin(UPrimitiveComponent* OverlappedComponent,
            AActor* OtherActor,
            UPrimitiveComponent* OtherComponent,
            int32 OterBodyIndex,
            bool bFromSweep,
            const FHitResult &SweepResult);
    UFUNCTION()
        void OnOverlapEnd(class UPrimitiveComponent* OverlappedComp,
            class AActor* OtherActor,
            class UPrimitiveComponent* OtherComp,
            int32 OtherBodyIndex);

};
```

ALightOF.cpp

```
// Sets default values
ALightOF::ALightOF()
{
    // Set this actor to call Tick() every frame.  You can turn this off to improve performance if you don't need it.
    PrimaryActorTick.bCanEverTick = true;

    /// Intendidad de luz
    lightIntensity = 3000.f;

    pointLight = CreateDefaultSubobject<UPointLightComponent>(TEXT("pointLight"));
    pointLight->Intensity = lightIntensity;
    pointLight->bVisible = true;
    RootComponent = pointLight;

    /// Damos valor a las propiedades que va a tener la bombilla
    lightSphere = CreateDefaultSubobject<USphereComponent>(TEXT("Light Sphere Component"));
    lightSphere->InitSphereRadius(300.0f);
    lightSphere->SetCollisionProfileName(TEXT("Trigger"));
    lightSphere->SetupAttachment(RootComponent);

    /// Suscribción al evento
    lightSphere->OnComponentBeginOverlap.AddDynamic(this, &ALightOF::OnOverlapBegin);
    lightSphere->OnComponentEndOverlap.AddDynamic(this, &ALightOF::OnOverlapEnd);
}

// Called when the game starts or when spawned
void ALightOF::BeginPlay()
{
    Super::BeginPlay();
    //ToggleLight(true);

    //isCollisioning = false;

    ///Colores del semáforo
    reddd = FLinearColor(255.f, 0.f, 0.f);
    greeen = FLinearColor(0.f, 255.f, 0.f);
}
```

```cpp
void ALightOF::OnOverlapBegin(UPrimitiveComponent * OverlappedComponent, AActor * OtherActor,
                              UPrimitiveComponent * OtherComponent, int32 OterBodyIndex, bool bFromSweep, const FHitResult & SweepResult)
{
    isCollisioning = true;

    ///Compruebo si con aquello con lo que ha colisionado es una instancia del cubo
    if (OtherActor != nullptr && (OtherActor != this) && OtherComponent != nullptr)
    {
        AcubeInstance = Cast<AMovableCube>(OtherActor);

        if (AcubeInstance != nullptr)
        {
            /// Si esta instancia del cubo no está dentro del Array la añado
            if (!cubos.Contains(AcubeInstance))
            {
                cubos.Add(AcubeInstance);
                UE_LOG(LogTemp, Warning, TEXT("Adding"));
            }
            else
            {
                UE_LOG(LogTemp, Warning, TEXT("Removing"));
            }
        }
    }
}
```

```cpp
void ALightOF::OnOverlapEnd(UPrimitiveComponent * OverlappedComp, AActor * OtherActor, UPrimitiveComponent * OtherComp, int32 OtherBodyIndex)
{

    if (OtherActor != nullptr && (OtherActor != this) && OtherComp != nullptr)
    {
        AcubeInstance = Cast<AMovableCube>(OtherActor);

        if (AcubeInstance != nullptr)
        {
        if(cubos.Num() > 0)
            cubos.Remove(AcubeInstance);
        }
    }
}
```

DIARIO DE DESARROLLO.

Durante el desarrollo de la práctica he tenido algún problema. Al añadir la funcionalidad del cubo que se mueve y de pared para realizar el efecto de rebote no tuve ningún problema, me resulto sencillo. Los problemas llegaron al realizar la funcionalidad del propio semáforo. El primero fue al cambiar de color cuando detectara una colisión, el cual parpadea no realizando el comportamiento deseado. Es algo que no entiendo ya que lo único que hice fue mediante el boleado que comprueba si un elemento esta colisionando con el semáforo pase a verde o rojo. No hago ninguna otra comprobación.
Otras dificultades que he tenido han sido al comprobar cual es objeto que esta colisionando. Lo cual me dificulto mucho el realizar la práctica ya que dediqué demasiado tiempo a ello sin conseguir solución. Este problema me impedía dar el comportamiento deseado a los cubos ya que a veces se paraban otras seguían, todo tipo de comportamientos extraños.