Práctica Semáforos C ++

Ignacio Tapia Marfil

Diseño y desarrollo de videojuegos y entornos virtuales.

09/05/2024

Índice.-

Ignacio Tapia Marfil

Diseño y Desarrollo de Videojuegos
Tecnologías de Desarrollo de Videojuegos y Entornos Virtuales II

Descripción General.-

Este proyecto está enfocado en aprender cómo incorporar la programación en c ++ en nuestros proyectos de Unreal Engine 5.

Para esto vamos a tener que crear dos objetos distintos. Un objeto móvil o coche y un semáforo.

- Objeto móvil o coche: Un actor que deberá moverse a una velocidad x (Editable desde motor) entre dos puntos fijos.
- Semáforo: Un objeto estático que contendrá. Un collider y dos luces, roja y verde. Este objeto se encargará de detectar cuando un coche entre o salga dentro de su colider y permitir o no el paso a los coches a través de él. Las luces se usan para representar esta posibilidad de paso.

Cuando un coche entre dentro del collider de un semáforo, se pondrá con luz roja hasta que este salga del collider que entonces volverá a la luz verde. Si no llega al colider ningún coche más durante este proceso no sucederá nada mas.

En el caso que un coche llegue al colíder del semáforo mientras ya hay un coche pasando, este primer coche se pondrá en estado de espera dentro de un array de coches independiente de cada semáforo. Cualquier otro coche que entre al semáforo mientras ya haya un coche esperando, se meterá igualmente en este array en orden de llegada.

Cuando el coche que estaba pasando sale del collider del semáforo, la luz se pone en verde y empieza un temporizador(Independiente de cada semáforo y editable desde motor). Cuando este temporizador llegue a 0, se permite el paso al primer coche en el array de espera y se saca de este array. Este proceso se repite hasta que no hay más coches dentro de el array de espera

Esto habrá que realizarlo enteramente con scripts en c ++.

Aparte de esto como trabajo extra también me propuse realizar mis propias Mesh con las herramientas que nos da Unreal Engine 5.

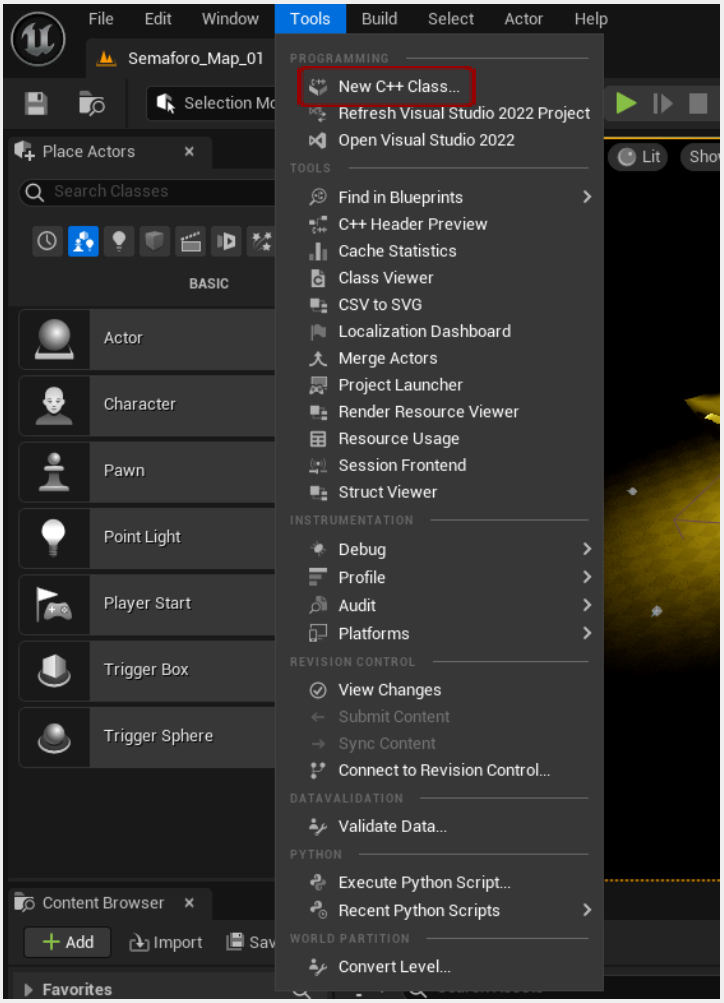
Descripción Técnica.-

Creación del proyecto.-

A la hora de crear el proyecto tuve que tener en cuenta el seleccionar la opción de C ++ para que así se me generasen los archivos necesarios para poder usar Visual Studio.

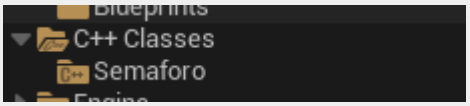
Una vez generado el proyecto, creé un nuevo mapa vacío para así poder trabajar sobre él.

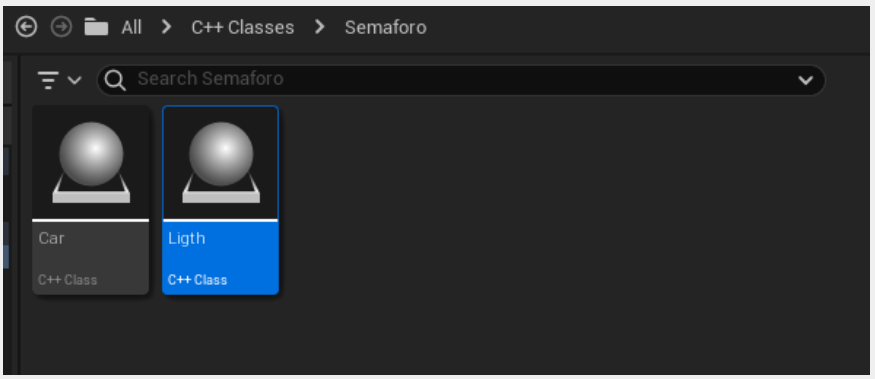
Después creé los dos archivos de C ++ que necesitaba para el proyecto, el coche y el semáforo.



Ignacio Tapia Marfil

Esto genera una carpeta donde se guardarán los scripts que se crean.





Con esto ya tenía el proyecto listo para empezar con el código.

Coche.-

Para el coche empecé creando 5 variables públicas, las cuales estarán con el EditAnywhere para así poder ser editadas también desde motor. Estas variables son:

```
class SEMAFORO_API ACar : public AActor
{
    GENERATED_BODY()

public:

    ///<summary>
    ///Velocidad
    /// </summary>
    UPROPERTY(EditAnywhere, Category = "Parametros")
    float speed = 1000;

    ///<summary>
    ///Movimiento
    /// </summary>
    UPROPERTY(EditAnywhere, Category = "Parametros")
    bool canMove = true;

    ///<summary>
    ///Mesh
    /// </summary>
    UPROPERTY(EditAnywhere, Category = "Parametros")
    class UStaticMeshComponent* mesh;

    ///<summary>
    ///TargetPoint
    /// </summary>
    UPROPERTY(EditAnywhere, Category = "TargetPoints")
    class ATargetPoint* StartTargetPoint;

    ///<summary>
    ///TargetPoint
    /// </summary>
    UPROPERTY(EditAnywhere, Category = "TargetPoints")
    class ATargetPoint* EndTargetPoint;

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

- speed: Controla la velocidad con la que se mueve el objeto.
- canMove: Controla si el objeto se puede mover o no(La ponemos aquí para poder hacer comprobaciones sin entrar a código)

- mesh: Es un objeto de tipo UStaticMeshComponent el cual seleccionaremos desde el motor. Este será el cuerpo de nuestro objeto

- StartTargetPoint, EndTargetPoint: Son objetos de tipo ATargetPoint, los cuales nos servirán como puntos de referencia por los cuales el objeto se moverá.

Después está la declaración de los métodos BeginPlay() y el Tick() que son los Start() y Update() de Unity. Estos se generan automáticamente al crear el script.

También está la declaración del método Movement(float) que es el método que vamos a usar para realizar el movimiento del coche.

Tick y Movement están en public para poder llamarlas desde cualquier parte del código.

En el apartado private he declarado unas variables que voy a utilizar en el método movement. Dos FVector que utilizaré para guardar las posiciones de los TargetPoints y un currentTime que utilizaré para controlar el movimiento entre esos dos puntos.

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

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

    void Movement(float);

private:

    FVector startLocation;
    FVector endLocation;

    float currentTime;
```

Con esto ya tenía terminada la declaración de todo lo que iba a necesitar, por lo que pase a la creación de los métodos.

Esto se hace desde el archivo .cpp que nos crea Unreal.

```cpp
#include "Car.h"

// Sets default values
ACar::ACar()
{
    // 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;

    mesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("mesh"));

    mesh->SetupAttachment(RootComponent);

}

// Called when the game starts or when spawned
void ACar::BeginPlay()
{
    if (StartTargetPoint && EndTargetPoint)
    {
        startLocation = StartTargetPoint->GetActorLocation();
        endLocation = EndTargetPoint->GetActorLocation();
    }

    currentTime = 0.0f;

    Super::BeginPlay();

}

// Called every frame
void ACar::Tick(float DeltaTime)
{
    canMove ? Movement(DeltaTime) : void();

    Super::Tick(DeltaTime);

}
```

Aquí tienes ya creadas: La implementación del constructor, a la que le añadí la generación de la mesh. La implementación del BeginPlay() a la que le añadí la localización de los TargetPoints y puse el tiempo a 0 para poder empezar con el movimiento. La implementación del Tick() al que le añadí un if para activar o desactivar el movimiento.

A parte de estas también realice la implementación del método Movement el cual controla el movimiento del objeto ayudándose del DeltaTime del proyecto y de un temporizador propio con el que podrá controlar la posición del objeto y ver si ha llegado a uno de los TargetPoints, en cuyo caso se intercambian los target points entre sí y se reanuda el movimiento.

```cpp
void ACar::Movement(float DeltaTime)
{
    if (StartTargetPoint && EndTargetPoint)
    {
        currentTime += DeltaTime;
        float Alpha = FMath::Clamp(currentTime * speed / FVector::Distance(startLocation, endLocation), 0.0f, 50.0f);

        FVector NewLocation = FMath::Lerp(startLocation, endLocation, Alpha);

        SetActorLocation(NewLocation);

        if (Alpha >= 1.0f)
        {
            FVector Temp = startLocation;
            startLocation = endLocation;
            endLocation = Temp;
            currentTime = 0.0f;
        }
    }
}
```

Con esto ya tenía toda la programación del coche, solo hace falta controlar la variable canMove para activar o desactivar su movimiento.

Semáforo.-

Al igual que con el coche, con el semáforo empecé creando 4 variables para poder ser editadas desde el motor.

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

public:

    /// <summary>
    /// Mesh
    /// </summary>
    UPROPERTY(EditAnywhere, Category = "Params")
    class UStaticMeshComponent* mesh;

    /// <summary>
    /// Temp
    /// </summary>
    UPROPERTY(EditAnywhere, Category = "Params")
    float TimeToGreen = 2.0f;

    /// <summary>
    /// Ligth
    /// </summary>
    UPROPERTY(EditAnywhere, Category = "Lights")
    class UPointLightComponent* RedLight;

    /// <summary>
    /// Ligth
    /// </summary>
    UPROPERTY(EditAnywhere, Category = "Lights")
    class UPointLightComponent* GreenLight;

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

- Mesh: Al igual que con el coche es un UStaticMEshComponent que se utiliza de cuerpo para el actor.
- TimeToGreen: Un float que se usa para ajustar el temporizador del semáforo.
- RedLigth, GreenLigth: Son dos UPointLigthComponent que serán las luces que usare para el semáforo.

En este script, además de los métodos BeginPlay() y Tick() que hay en el resto de script, hice 4 métodos más: AllowCar(ACar*), que accede a el coche que le pasamos y activa su movimiento. ChangeToAllow(), que cambia la disponibilidad del semáforo para poder pasar. ChangeLigths(), que cambia las luces entre la roja y la verde.

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

public:

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

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

    void AllowCar(ACar* Car);

    void ChangeToAllow();

    void ChangeLights(bool bIsGreen);

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

private:

    bool Allow;

    FTimerHandle TimerHandle;

    TArray<ACar*> CarsWaiting;
};
```

Además de estos métodos también cree unas variables para controlar el semáforo, Allow, TimeHandle y un array de objetos Car para saber los coches que hay esperando en el semáforo.

Con esto creado ya pase a la creación de los métodos en el archivo .cpp.

En el constructor de la clase, ALigth(), genere el collider que voy a utilizar y lo vincule a los métodos OnOverlapBegin y el OnOverlapEnd. También generé las dos luces y la malla del objeto.

```
// Sets default values
ALigth::ALigth()
{
    // 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;

    UBoxComponent* BoxComponent = CreateDefaultSubobject<UBoxComponent>(TEXT("CollisionBox"));
    RootComponent = BoxComponent;
    BoxComponent->SetCollisionProfileName(TEXT("Trigger"));
    BoxComponent->OnComponentBeginOverlap.AddDynamic(this, &ALigth::OnOverlapBegin);
    BoxComponent->OnComponentEndOverlap.AddDynamic(this, &ALigth::OnOverlapEnd);

    RedLight = CreateDefaultSubobject<UPointLightComponent>(TEXT("RedLight"));
    RedLight->SetupAttachment(RootComponent);
    GreenLight = CreateDefaultSubobject<UPointLightComponent>(TEXT("GreenLight"));
    GreenLight->SetupAttachment(RootComponent);

    mesh = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("mesh"));
    mesh->SetupAttachment(RootComponent);

}

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

    IsGreen = true;

    ChangeLights(IsGreen);
}

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

    //Si el semaforo tiene permitido el paso, mira en la lista de espera del cohe y si hay alguno le permite esperar
    if (IsGreen)
    {
        if (CarsWaiting.Num() > 0)
        {
            AllowCar(CarsWaiting[0]);
        }
    }
}
```

También tenemos la implementación del BeginPlay y el Tick. En este primero simplemente inicializo la variable IsGreen a true y llamó a la función Change color para poner las luces en verde. En el Tick compruebo si hay algún coche en la lista de espera y si hay alguno y está permitido el paso, lo dejo pasar.

En el método OnOverlapBegin(), controlo cuando un coche entra dentro del collider del semáforo. Cuando esto pasa, el movimiento del coche se detiene y luego compruebo si se puede pasar o no. Primero, miro si hay algún coche esperando ya en el colider, si es así simplemente añado este nuevo coche a la lista de espera. Si no hay ningún coche esperando, miro si el semáforo está activo o no, si está activo simplemente dejo al coche pasar y cambio las luces a rojo. Si no está permitido el paso, este coche se mete en la lista de espera.

```cpp
void ALigth::OnOverlapBegin(UPrimitiveComponent* OverlappedComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult& SweepResult)
{
    ACar* Car = Cast<ACar>(OtherActor);

    //Un coche entra en el colider del semaforo
    if (Car)
    {
        //Se para el movimiento del coche antes de hacer nada
        Car-> canMove = false;

        //Si hay ya un coche en la lista de espera, el chcohe que ha entrado se añade a la lista de espera
        if (CarsWaiting.Num() > 0)
        {
            CarsWaiting.Add(Car);
        }
        else
        {
            //si no hay ningun coche en la lista de espera se comprueba si el semaforo permite el paso o no
            if (IsGreen)
            {
                //si lo permite activa el movimiento del coche, deja de permitir pasar a mas coches y cambia su luz a rojo
                Car->canMove = true;
                IsGreen = false;
                ChangeLights(false);
            }
            else
            {
                //si no lo permite simplemente se añade el coche a la lista de espera
                CarsWaiting.Add(Car);
            }

        }
    }
}
```

En el método OnOverlapEnd(), controlo cuando un coche sale del collider del semáforo. Cuando este sale, la luz se cambia a verde. Si hay un coche esperando, se activa el temporizador para activar el paso. Si no hay ningún coche esperando simplemente se activa el paso.

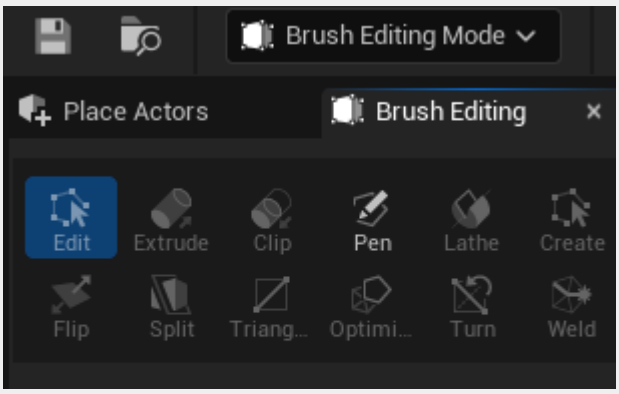
```cpp
void ALigth::OnOverlapEnd(UPrimitiveComponent* OverlappedComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex)
{
    ACar* Car = Cast<ACar>(OtherActor);
    //Un coche sale del colider del semaforo
    if (Car)
    {
        //Se cambia la luz a verde
        ChangeLights(true);
        //Si hay algun chcoche esperando
        if (CarsWaiting.Num() > 0)
        {
            //Se llama al metdodo que activa el permiso en el semaforo, pero se le llama esperando "TimeToGreen" Segundos
            GetWorldTimerManager().SetTimer(TimerHandle, this, &ALigth::ChangeToAllow, TimeToGreen, false);
        }
        //Si no hay
        else
        {
            //Se activa el permiso de pasar directamente
            ChangeToAllow();
        }
    }
}
```

```
110     }
111
112
113
114     void ALigth::ChangeToAllow()
115     {
116         Allow = true;
117     }
118
119     void ALigth::AllowCar(ACar* Car)
120     {
121         Car->canMove = true;
122         CarsWaiting.RemoveAt(0);
123         Allow = false;
124         ChangeLights(false);
125     }
126
127     void ALigth::ChangeLights(bool Green)
128     {
129         RedLight->SetVisibility(!Green);
130         GreenLight->SetVisibility(Green);
131     }
132
```

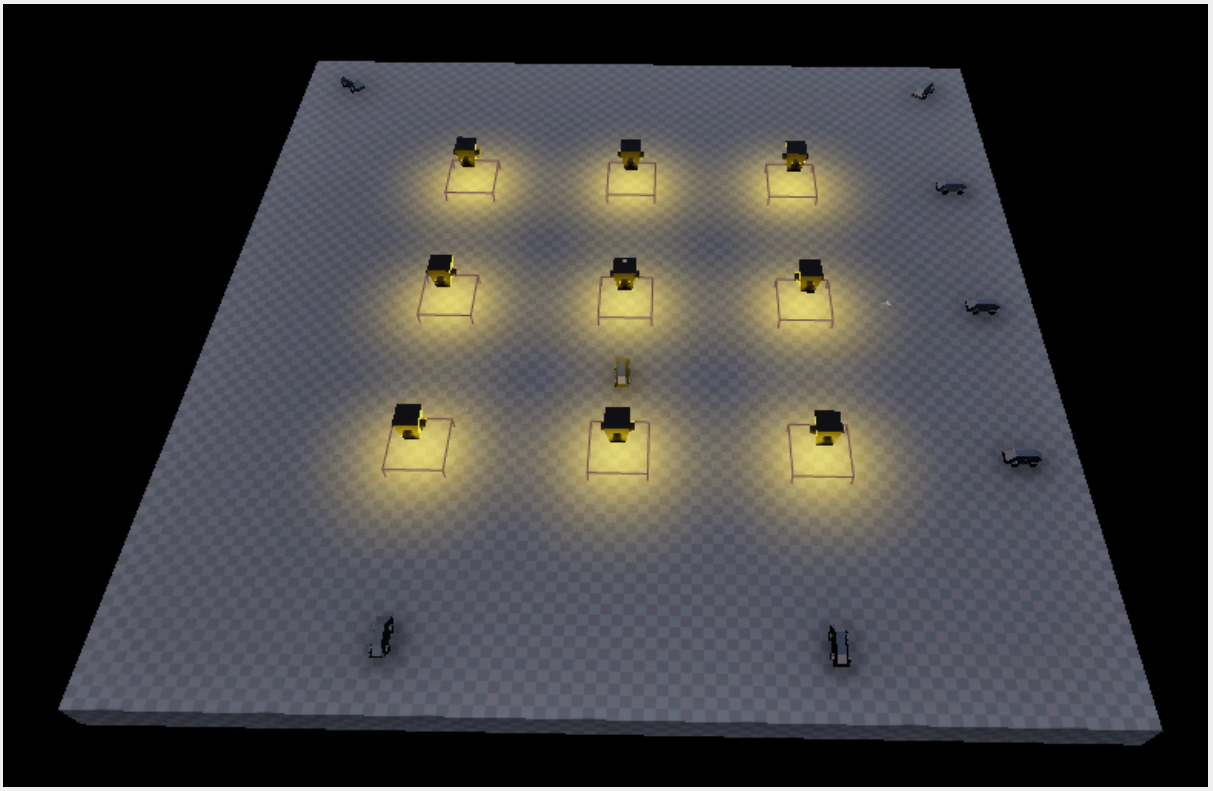
Estos son los métodos que se llaman en los otros dos: ChangeToAllow(), pongo a true el booleano Allow. AllowCar(), activo el movimiento del coche que se le pasa, lo saco de la lista de espera y desactivo el semáforo cambiando la luz. ChangeLigths() si se manda un booleano true se pone a verde si se manda un booleano false se pone en rojo.

Con esto ya está toda la programación del proyecto realizada, solo me quedaba cambiar las mesh básicas por unas complejas.

Para eso usé la herramienta de BrushEditing, que me permitió de una forma simple editar y juntar formas geométricas simples para así poder crear algo más complejo.



Con esto creé dos mallas, una para el coche y otra para el semáforo. Son mallas simples pero que añaden algo más al proyecto.



Diario de Desarrollo.-

Este proyecto realizado en UnrealEngine 5 ha supuesto un reto más alto que el resto de proyectos realizados durante esta asignatura. Me he encontrado con varios problemas, sobre todo con el cómo empezar con el C ++ en Unreal y la implementación de esto.

Empezando por enterarme de cómo era la sintaxis básica de los métodos y funciones dentro del motor, lo cual me llevó más tiempo de lo que suponía debido a que me encontraba bastantes formas de hacerlo. Al final, gracias a la ayuda de ChatGpt conseguí encontrar la forma más común y fácil de creación del archivo .h en Unreal.

El resto del proyecto, sobre todo lo que tiene que ver con la parte de la lógica, no supuso mucho problema y fue más rápido de lo que esperaba.

Otro problema que me encontré que no tiene que ver con el código fueron los errores constantes de Unreal a la hora de incorporar la programación. Cierres repentinos del programa, errores de código donde en principio esta todo bien y errores a la hora de descargar el proyecto.

También indicar que debido a que quería incorporar una malla compleja a los coches y a los semáforos, más allá de cubos. Tuve alguna complicación buscando alguna forma para hacerlo, probé con blueprints con una malla ya creada pero esto dio muchos problemas. Al final descubrí una herramienta del propio Unreal que te permite crear Mesh complejas a partir de otras más simples. Gracias a esto conseguí crear dos mallas para mis dos objetos.

Más allá de todo esto no tuve ningún problema más y el proyecto acabó siendo más sencillo de lo que parecía en un principio.

Bibliografía.-

---

https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/IntroductionToCPP/

https://www.blackbox.ai/