Práctica Redes

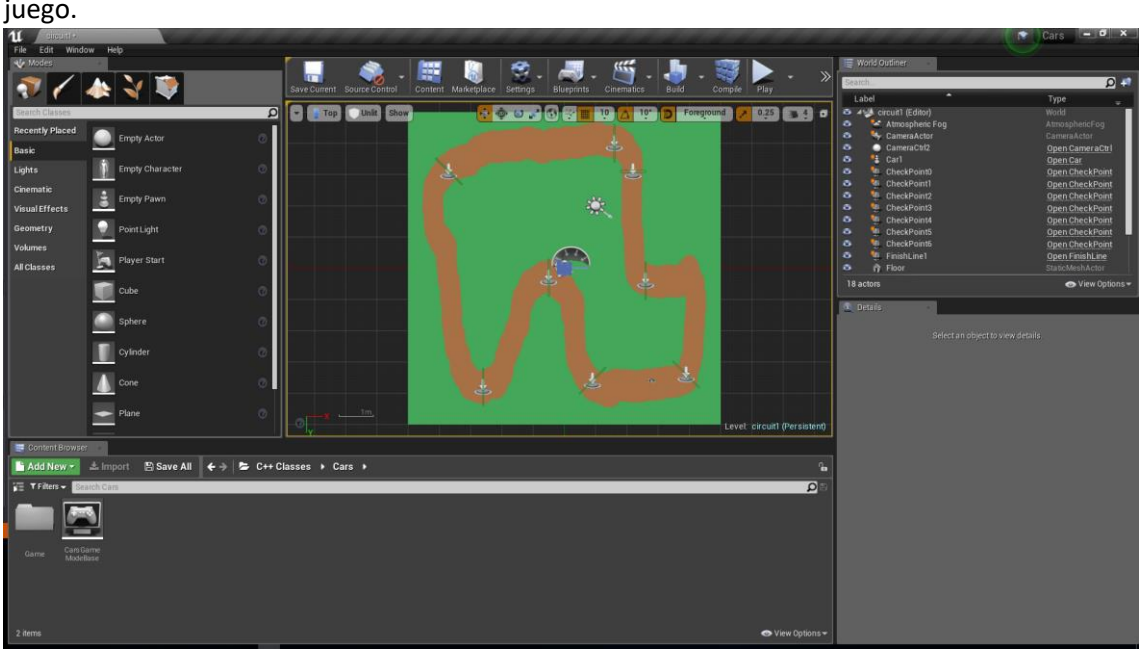
Estos días vamos a hacer diversas prácticas de red sobre un mini juego de coches muy sencillo. Todas las prácticas que hagamos serán en Unreal, sin embargo no vamos a usar las facilidades de uso en red de Unreal. Para ello nos montaremos nuestra propia infraestructura de red basada en las prácticas que hicisteis en C++ con Juanan. Durante estas prácticas vamos a promover el uso de C++, intentando huir de los blueprints en la medida de lo posible. Lo que aprendamos aquí será extrapolable a otros motores e incluso nos permitirá comunicar aplicaciones realizadas en diferentes motores, plataformas o lenguajes (por ejemplo una aplicación hecha en Unreal para PC comunicada con una aplicación hecha en Unity para android).

Juego monojugador

Antes de ponernos con la práctica en si vamos a crear un sencillo juego sobre el que luego realizaremos el resto de ejercicios. El juego será un juego de coches 2D visto desde arriba, donde controlaremos un coche con WASD al que seguirá la cámara. La única funcionalidad extra que tendremos que implementar será la acumulación vueltas al circuito, contabilizando la mejor vuelta.

Circuito

Al abrir el proyecto tendremos un cuidado y detallado circuito sobre el que se realizará el juego.



A parte del circuito ya hay colocadas unas cuantas entidades por el mapa como los check points, que nos garantizarán que hacemos una vuelta completa, la línea de meta, que comprobará que la vuelta dada es correcta y actualizará el número de vueltas y el mejor tiempo, la cámara que seguirá al coche y el propio coche que manejaremos.

Camera

Lo primero que tendremos que hacer es poner nuestra cámara como view target en el player controller, para que sea ésta la encargada de mostrar el juego. Esto lo haremos en el BeginPlay del AGameCamera.

```
#include "Kismet/GameplayStatics.h"
```

```
void AGameCamera::BeginPlay()
{
  APlayerController* OurPlayerController =
UGameplayStatics::GetPlayerController(this, 0);
  if (OurPlayerController)
  {
    OurPlayerController->SetViewTarget(this);
  }
}
```

Car

La clase ACar representa a el coche que controlaremos y será la encargada de gestionar el input que se recibe. En el proyecto se han declarado ya dos axis mappings (Move y Turn) que se controlan con WASD o con el stick izquierdo de un mando. Deberemos enlazar dos métodos de la clase ACar a dichos ejes y cuando se llamen modificaremos un vector de input que será miembro de la clase. A partir de este vector moveremos luego el coche.

El enlace de los métodos de la clase ACar con los ejes lo realizaremos en el método `SetupPlayerInputComponent`, que tendremos que sobrescribir, llamando al método `BindAxis` del `UInputComponent`.

```
class UCarMovementComponent;

class CARS_API ACar : public APawn
{
public:
  // Called to bind functionality to input
  virtual void SetupPlayerInputComponent(class UInputComponent* InputComponent)
override;

protected:
  //Input functions
  void Move(float AxisValue);
  void Turn(float AxisValue);

protected:
  //Input variables
  FVector2D m_vMovementInput = FVector2D::ZeroVector;
  //Movement
  UPROPERTY(EditAnywhere)
  UCarMovementComponent* m_pCarMovement;
};
```

```
#include "Components/InputComponent.h"

// Called when the game starts or when spawned
void ACar::BeginPlay()
{
  m_vMovementInput.Set(0.f, 0.f);
}

// Called to bind functionality to input
void ACar::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
  Super::SetupPlayerInputComponent(PlayerInputComponent);
  PlayerInputComponent->BindAxis("Move", this, &ACar::Move);
  PlayerInputComponent->BindAxis("Turn", this, &ACar::Turn);
}

//Input functions
void ACar::Move(float AxisValue)
```

```
{
  m_vMovementInput.Y = FMath::Clamp<float>(AxisValue, -1.0f, 1.0f);
}

void ACar::Turn(float AxisValue)
{
  m_vMovementInput.X = FMath::Clamp<float>(AxisValue, -1.0f, 1.0f);
}
```