Práctica Redes II

Lobby

En este apartado empezaremos a montar todo el tema de inicialización y conexión de red. Para ello nos apoyaremos en UMG para crear un sencillo sistema de GUI.

GUI

Para poder crear nuestros modos de juego cargamos el módulo UMG.

```
public Cars(ReadOnlyTargetRules Target) : base(Target)
{
  PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;
  PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject",
"Engine", "InputCore", "UMG" });
}
```

Vamos a llevar en nuestro game mode (`ACarsGameModeBase`) la gestión de cambios de estado. Para la gestión de estados vamos a tener diferentes UserWidgets en el que gestionaremos eventos de GUI y llamaremos a funciones propias que declararemos en `ACarsGameModeBase`. Lo primero será declararnos una función para cambiar de widget

```
class CARS_API ACarsGameModeBase : public AGameModeBase
{
  GENERATED_BODY()
public:
  UFUNCTION(BlueprintCallable, Category = CarsNet)
  void ChangeMenuWidget(TSubclassOf<UUserWidget> NewWidgetClass);
protected:
  /** Called when the game starts. */
  virtual void BeginPlay() override;
  UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = CarsNet)
  TSubclassOf<UUserWidget> StartingWidgetClass;
  UPROPERTY()
  UUserWidget* CurrentWidget;
};
```

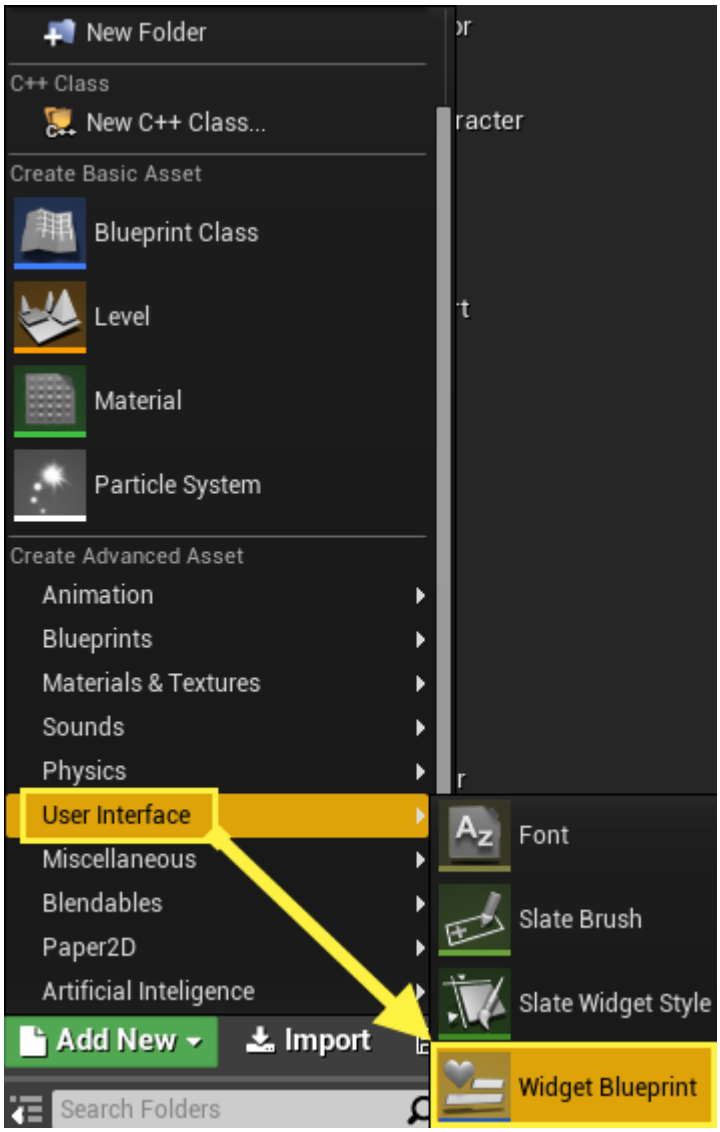
```
#include "Blueprint/UserWidget.h"

void ACarsGameModeBase::BeginPlay()
{
  Super::BeginPlay();
  ChangeMenuWidget(StartingWidgetClass);
}

void ACarsGameModeBase::ChangeMenuWidget(TSubclassOf<UUserWidget> NewWidgetClass)
{
  if (CurrentWidget != nullptr)
  {
    CurrentWidget->RemoveFromViewport();
    CurrentWidget = nullptr;
  }
  if (NewWidgetClass != nullptr)
  {
    CurrentWidget = CreateWidget<UUserWidget>(GetWorld(), NewWidgetClass);
    if (CurrentWidget != nullptr)
    {
      CurrentWidget->AddToViewport();
    }
```

```
  }
}
```

Además tenemos que cambiar el input mode del player controller para que reacciones con los elemento del GUI

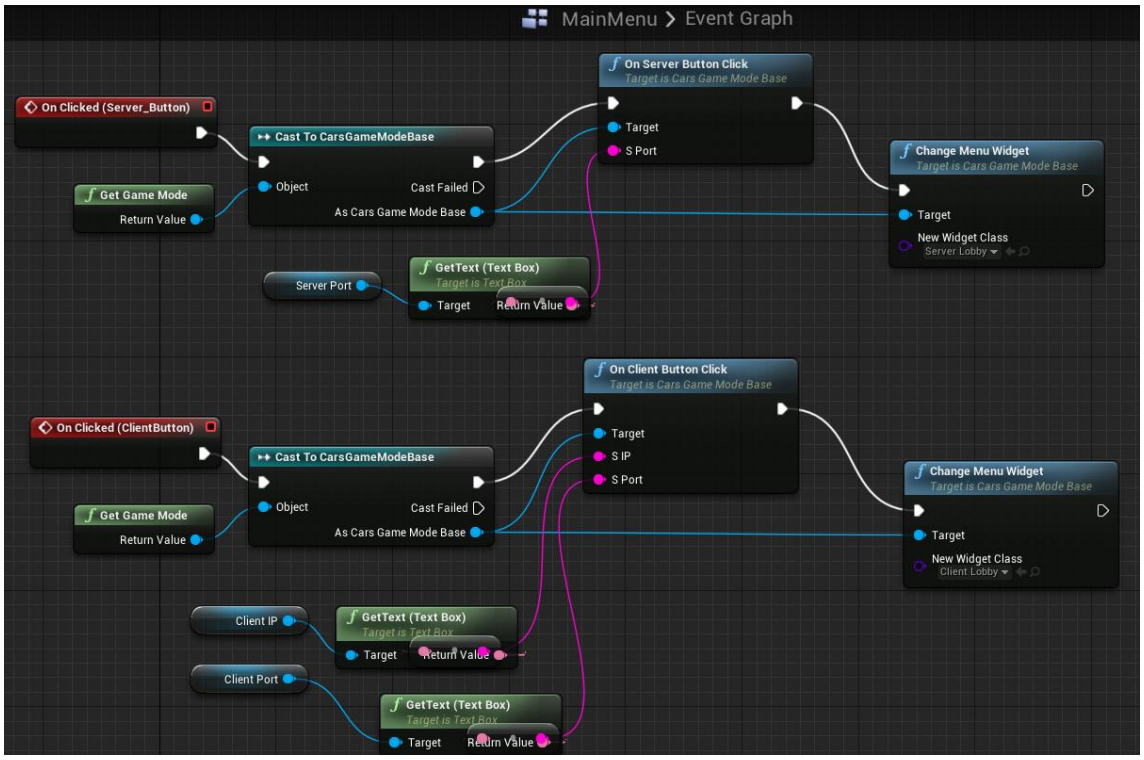
```
void ACarsPlayerController::BeginPlay()
{
  Super::BeginPlay();
  SetInputMode(FInputModeGameAndUI());
}
```

Añadimos nuestro primer modo



Y le llamamos MainMenu. Crearemos botones y campos de texto para obtener lo siguiente:

Y asignaremos a los botones eventos, creando con ellos el siguiente blueprint



Pero para poder terminarlo tendremos que publicar e implementar las funciones que se lanzarán con los eventos de pulsado de botones.

Net

Cuando se pulsen los botones del menú lo que tendremos que hacer por fin es empezar con la conexión

```
class CARS_API ACarsGameModeBase : public AGameModeBase
{
  GENERATED_BODY()
public:
  UFUNCTION(BlueprintCallable, Category = CarsNet)
  void OnServerButtonClick(FString sPort);
  UFUNCTION(BlueprintCallable, Category = CarsNet)
  void OnClientButtonClick(FString sIP, FString sPort);
};
```

```
#include "CarsGameModeBase.h"
#include "Game/CarsPlayerController.h"
#include "Blueprint/UserWidget.h"
#include "Net/buffer.h"
#include "Net/paquete.h"
#include <iostream>


ACarsGameModeBase::CServerObserver::CServerObserver() : m_pController(nullptr)
{
  if (!Net::CManager::getSingletonPtr())
  {
    Net::CManager::Init();
  }
  m_pManager = Net::CManager::getSingletonPtr();
}

ACarsGameModeBase::CServerObserver::CServerObserver(ACarsGameModeBase*
_pController) : m_pController(_pController)
{
  if (!Net::CManager::getSingletonPtr())
  {
    Net::CManager::Init();
  }
  m_pManager = Net::CManager::getSingletonPtr();
}

void ACarsGameModeBase::CServerObserver::dataPacketReceived(Net::CPaquete*
packet)
{
  if (m_pManager->getID() == Net::ID::SERVER)
  {
  }
  else
  {
    // Creamos un buffer con los datos para leer de manera más cómoda
    Net::CBuffer data;
    data.write(packet->getData(), packet->getDataLength());
    data.reset();
    char sInfo[128];
    data.read(sInfo, data.getSize());
    if (GEngine)
    {
      // Put up a debug message for five seconds. The -1 "Key" value (first
argument) indicates that we will never need to update or refresh this message.
      GEngine->AddOnScreenDebugMessage(-1, 5.0f, FColor::Red, sInfo);
    }
  }
}

//---------------------------------

void ACarsGameModeBase::CServerObserver::connexionPacketReceived(Net::CPaquete*
packet)
{
  if (m_pManager->getID() == Net::ID::SERVER)
  {
    // Creamos un buffer con los datos para leer de manera más cómoda
    Net::CBuffer data;
    const char* sHello = "Connected";
    data.write(sHello, sizeof(sHello));
    m_pManager->send(data.getbuffer(), data.getSize());
```

```
    if (GEngine)
    {
      // Put up a debug message for five seconds. The -1 "Key" value (first
argument) indicates that we will never need to update or refresh this message.
      GEngine->AddOnScreenDebugMessage(-1, 5.0f, FColor::Red, "Client connected!
");
    }
  }
  else
  {

  }
}

//---------------------------------

void
ACarsGameModeBase::CServerObserver::disconnexionPacketReceived(Net::CPaquete*
packet)
{

}

ACarsGameModeBase::ACarsGameModeBase(const class FObjectInitializer&
ObjectInitializer) : AGameModeBase(ObjectInitializer), m_oObserver(this)
{
  PrimaryActorTick.bCanEverTick = true;
  PlayerControllerClass = ACarsPlayerController::StaticClass();
  if (!Net::CManager::getSingletonPtr())
  {
    Net::CManager::Init();
  }
  m_pManager = Net::CManager::getSingletonPtr();
}

APawn* ACarsGameModeBase::SpawnDefaultPawnFor_Implementation(AController*
NewPlayer, AActor* StartSpot)
{
  return nullptr;
}

void ACarsGameModeBase::BeginPlay()
{
  Super::BeginPlay();
  ChangeMenuWidget(StartingWidgetClass);
}

void ACarsGameModeBase::Tick(float DeltaSeconds)
{
  Super::Tick(DeltaSeconds);
  m_pManager->tick();
}

void ACarsGameModeBase::ChangeMenuWidget(TSubclassOf<UUserWidget> NewWidgetClass)
{
  if (CurrentWidget != nullptr)
  {
    CurrentWidget->RemoveFromViewport();
    CurrentWidget = nullptr;
  }
  if (NewWidgetClass != nullptr)
  {
    CurrentWidget = CreateWidget<UUserWidget>(GetWorld(), NewWidgetClass);
```

```
    if (CurrentWidget != nullptr)
    {
      CurrentWidget->AddToViewport();
    }
  }
}

void ACarsGameModeBase::OnServerButtonClick(FString sPort)
{
  GEngine->AddOnScreenDebugMessage(-1, 5.0f, FColor::Green, *FString("Server"));

  m_pManager->addObserver(&m_oObserver);
  m_pManager->activateAsServer(FCString::Atoi(*sPort));
}

void ACarsGameModeBase::OnClientButtonClick(FString sIP, FString sPort)
{
  GEngine->AddOnScreenDebugMessage(-1, 5.0f, FColor::Green, *FString("Client"));

  m_pManager->activateAsClient();
  m_pManager->addObserver(&m_oObserver);
  m_pManager->connectTo(TCHAR_TO_ANSI(*sIP), FCString::Atoi(*sPort));
}

void ACarsGameModeBase::OnServerStartButtonClick()
{
  GEngine->AddOnScreenDebugMessage(-1, 5.0f, FColor::Green, *FString("Server
Start!"));
}
```