# Atajos

**Compile** 🡪 CTRL + ALT + F11 (from visual)

**Unpossess Character** 🡪 F8

# Notes

**Setup** **for Compiling All on Startup**: Editor Preferences > Loading & Saving > True

**Rotate Pawn to movement**: Character Movement Details

Graphical user interface, application

Description automatically generated

## Compilar

En el explorador de soluciones: Click derecho en MCVGame y compilar

Graphical user interface, application

Description automatically generated

# Clases Instances / Levels

## GameInstance

* **What is it?** A Global Singleton Class**:** tostoredata among levels.
* **Can store**: savegame, game config, assets load…
* To **Access**: World > GetGameInstance()
* Base Class: UGameInstance

## Gamemode

* **What is it?** The level itself (can be Lobby or Menu)
* To Access: World > GetAuthGameMode()
* In Multiplayer Gamemode **only exists in server side**.
* Base Class: AGameModeBase

## GameState

* **Store**: common global state of **all players** (as team score, list of connected players)
* By default it has a **PlayerArray** of players
* Receives events when a player enters or exists the game.
* Lifecycle: the same as Gamemode (it is created by the Gamemode)
* Base Class: AGameStateBase
* In Multiplayer: **created in Server Side and replicated to the clients**

## PlayerState

* Store: player data dependent of its Pawn/Character (score, level, XP)
  + (El Pawn/Character would be the representation, the avatar)

## PlayerController

* Has the input
* Can Own a Pawn
* The Player Controller is always the same (the Pawn can change)

## World

* Tiene **timer** para eventos
* Cada world está asociado a un nivel
* Los actores tienen un GetWorld para acceder a él.

# Delegates

## Simple

In InventoryComponent.h we **declare**



To **fire** **the** **event**:

Texto

Descripción generada automáticamente

El IfBound es por si no hay nadie escuchando no lo lanza.

In the constructor we call this function BindOnInventoryChanged() to **bind** to the event (we could call this “Register()” or “Subscribe”):

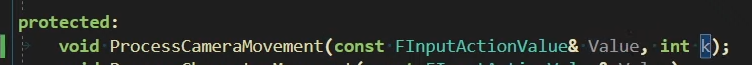
Texto

Descripción generada automáticamente

### Con párametros

Podemos pasarle parámetros



Y añadimos el parametro en el .h

## MULTICAST DELEGATE

**Multicast Delegate** (2 params: levelPrevious, newLevel)

DECLARE\_MULTICAST\_DELEGATE\_TwoParams(FOnPlayerLevelUpDelegate, int, int);

We are defining the type of the delegate, so we create a variable Delegate



We create a function to AddXP and use the delegate:

Text

Description automatically generated

We call the function with Broadcast.

Now we need to bind (or subscribe) to the delegate (we could use BeginPlay, but maybe there is no PlayerState), so we override the function PossessedBy

In **MCVGameCharacter.cpp** 🡪 #include “GameFramework/MCVGamePlayerState.h”

And we add the function. We can use:

* AddUObject
* AddLamba
* AddSP (shared pointer)

Text

Description automatically generated

## DYNAMIC MULTICAST DELEGATE (For BP)

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_TwoParams(FOnPlayerLevelUpDelegateDynamic, int, PrevLevel, int, CurrentLevel);

* Para registrar desde **Blueprints** se usan los DYNAMIC\_MULTICAST
* Necesitan el nombre de la variable tamb.

Declare variable

Text

Description automatically generated

Call Broadcast

Text

Description automatically generated

To listen to the Dynamic Delegate is a little bit different:



NOTE: we need to mark the OnLevelUp function as UFUNCTION()

## Unbind

We can unbind too, so we store a FDelegateHandle

Graphical user interface

Description automatically generated with low confidence

And we want to unbind when unpossessed so override the function



We store the Handle when we add the function



For Dynamics:

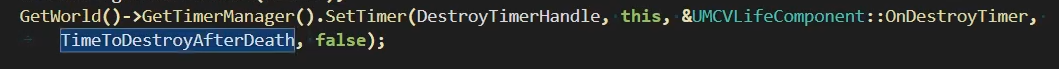


Text

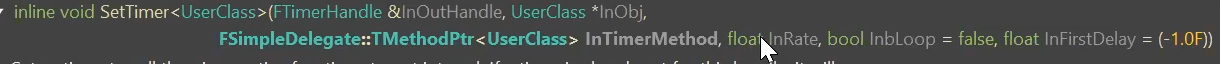
Description automatically generatedAnd we remove (unbind) like this

# Timers

 (esto lo crea en el .h, pero creo que tamb se puede encima)



It needs the following parameters:

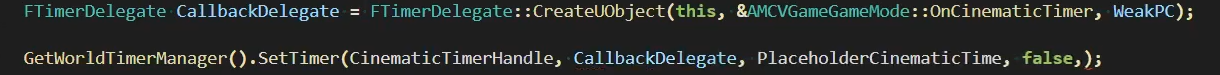


* **FTimerHandle**: handle we have already defined
* **UserClass**: this in this case (GameGameMode), Object with the function
* **FSimpleDelegate**: function of ObjClass
* **float** **InRate**: how much time it lasts to start the timer
* **InbLoop**: is this loopable
* **Float** **inFirstDelay**: a first delay

## Timer with parameters

Set the timer:





}

Usamos un TWeakObjectPtr porq llamamos después de un tiempo (puede ser que se haya destruido el player). Es como si ponemos el UPROPERTY, para que no pete.

Text

Description automatically generated

# KEYS

## TSoftObjectPtr

Al definir un puntero para el nivel **UWorld\* MainMenu** yo tengo cargado el nivel en memoria.

En la GameInstance tenemos ptrs a todos los niveles y NO interesa tenerlos todos en memoria por lo que usamos **TSoftObjectPtr<UWorld> MainMenu**;

## MCVGAME\_API

Macro para que la clase se exponga a otros módulos.

Texto

Descripción generada automáticamente

## UStaticMeshComponent

Stores the mesh, its materiales, etc. (.h)

## TSubclassof<>

Type of class that is subclass of the template type

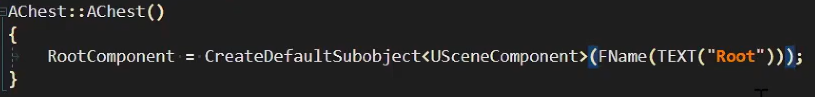
Interfaz de usuario gráfica

Descripción generada automáticamente

# Functions

## CreateDefaultSubobject

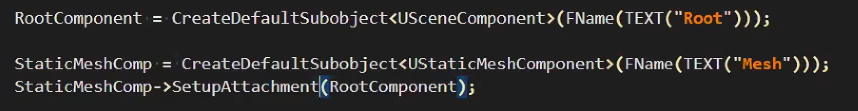
To create a Component. Must be called in the **constructor** of C++ classes (.cpp)



## SetUpAttachment

To parent a component

childComp->SetupAttachment(parentComp);



## OnComponentBeginOverlap

(In MCVGameCharacter.cpp)

// Register event to a function. To do this, the function should be marked as UFUNCTION() in .h

GetCapsuleComponent()->OnComponentBeginOverlap.AddDynamic(this,

&AMCVGameCharacter::OnCapsuleOverlaps);

void AMCVGameCharacter::OnCapsuleOverlaps(

UPrimitiveComponent\* OverlappedComponent,

AActor\* OtherActor,

UPrimitiveComponent\* OtherComp,

int OtherBodyIndex,

bool bFromSweep,

const FHitResult& SweepResult

){

if(UInventory\* OtherInventory =

OtherActor->FindComponentByClass<UInventory>()){

Inventory->TransferAllContentsFrom(OtherInventory)

}

}

En foto:

Texto

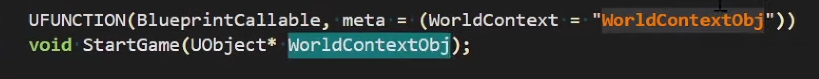
Descripción generada automáticamente



## UGameplayStatics::OpenLevelBySoftObjectPtr

UGameplayStatics::OpenLevelBySoftObjectPtr(this, GameConfig.GameLevel)

* This = const UObject\* WorldContextObject
* GameLevel = const TSoftObjectPtr<UWorld>



Se le puede pasar como meta información el WorldContextObj como parametron.

## GetWorld()->LineTrace (Clase3 Quim a las 19:57)

There are several types:

* Multi: a ray can return several collisions
* Single: only the first one
* ByChannel:
* ObjectType: types of collisions (pawn, for example)

Texto

Descripción generada automáticamenteExample:

And QueryParams can Ignore myself Actor

## UE\_LOG

* LogTemp
* Display, Warning, Error
* TEXT(“Interact Actor %s”, ActorName)

**Ej: UE\_LOG(LogTemp, Display, TEXT(“Hello”));**

**With params:**



## PlayerController->Possess(Pawn) / UnPossess

It allows to possess a Character with the input (changes the camera too).

Example: when you look at the other Character and press E, you possess the other one

Texto

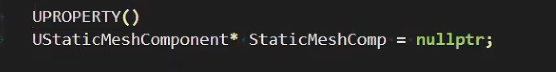
Descripción generada automáticamente

## Clamp, Min, Max(FMath)

* FMath:Min(x,y)
* FMath:Max(x,y)



# Attributes



## UPROPERTY

IMPORTANT: when having a ptr of a Unreal Class (UOBJECT), Always put UPROPERTY ()

When the **UStaticMeshComponent** above is deleted, the UPROPERTY allows the garbage collector to delete and warn us that the object is not usable.

### Flags

Can be added to the UPROPERTY



* **EditAnywhere vs EditDefaultsOnly vs EditInstanceOnly**
  + **Defaults:** desde el Blueprint Editor
  + **Instance:** desde el Viewport, con instancias ya arrastradas
* **BlueprintReadWrite**: can be modified in r untime
* **BlueprintReadOnly**: can NOT be modified in runtime

Categories para agrupar

A screen shot of a computer

Description automatically generated with low confidence

## UFUNCTION()

This attribute is necessary when we want that a function could be called from blueprints (.h)

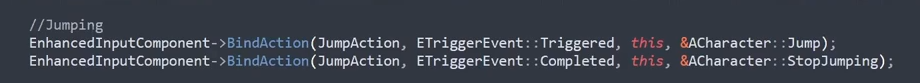
UFUNCTION(BlueprintCallable)

void TransferInventoryContents(UInventoryComponent\* OtherInventory);

# Input

In MCVGameCharacter (SetupPlayerInputComponent)

Graphical user interface, text, application

Description automatically generated

ETriggerEvent::Triggered

ETriggerEvent::Started 🡪 when it started to be pressed

ETriggerEvent::Completed 🡪 when it started to be realesed

# Data Structures

* **TArray<>** …………………… equivalent of **std::vector<>**
* **TMap<>** …………………… equivalent of **std::map<>**
* **FString** ……………………
* **FName** ……………………
* **FText** ……………………