

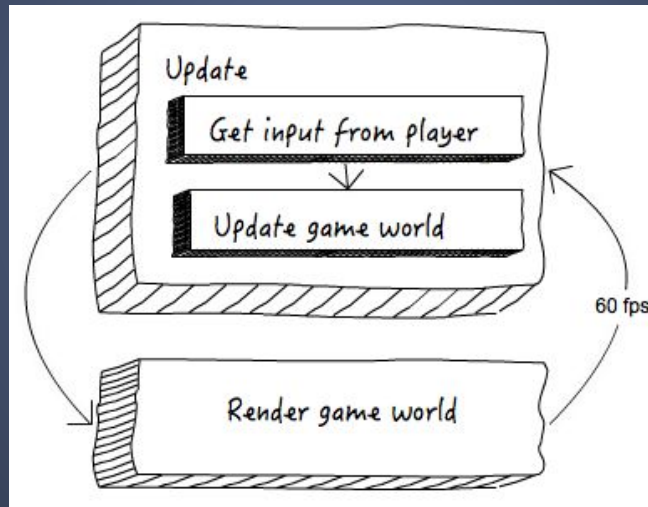


Desarrollo de Videojuegos

Jugabilidad: Arquitectura y
mecánicas principales
Arquitectura del videojuego

Motivación

- ¿Qué tipo de *software* es un videojuego?
 - *Bucle común* a un sistema multimedia interactivo...



- ...pero para un videojuego, aunque hay muchos tipos, hay más *arquitectura que puede ser común*

Armazón de jugabilidad

GAMEPLAY FRAMEWORK

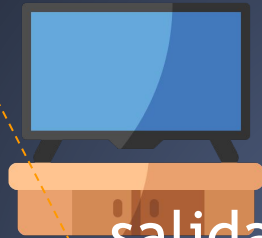


Jugadores



entradas

Inputs



salidas



HUDS y PlayerCameraManagers



Cameras con
AudioListeners

World

Actors (Pawns... y otros)

PlayerControllers

Player States

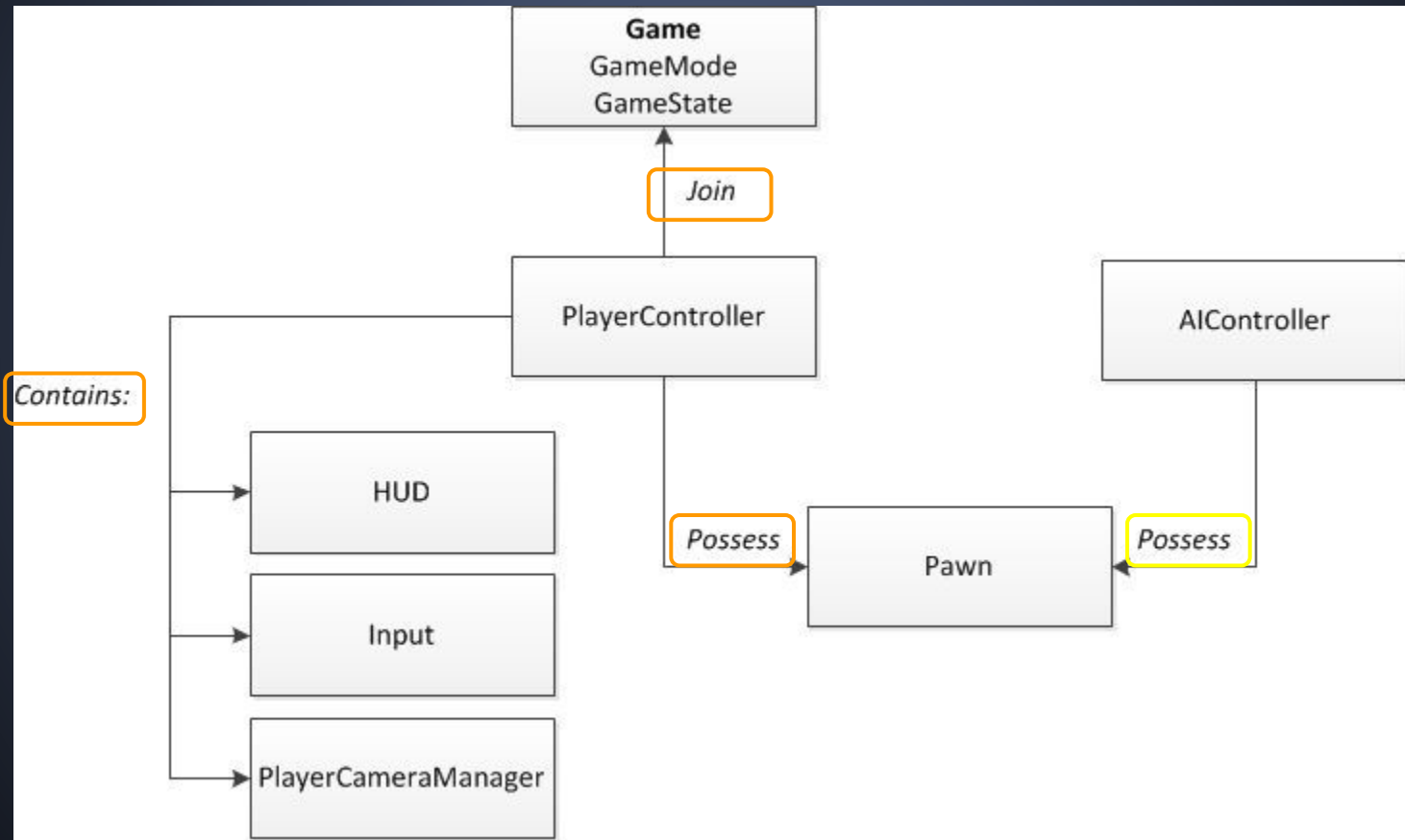
AIControllers
(bots*)

GAME Instance

Game Mode
Game State

Game

Armazón de jugabilidad



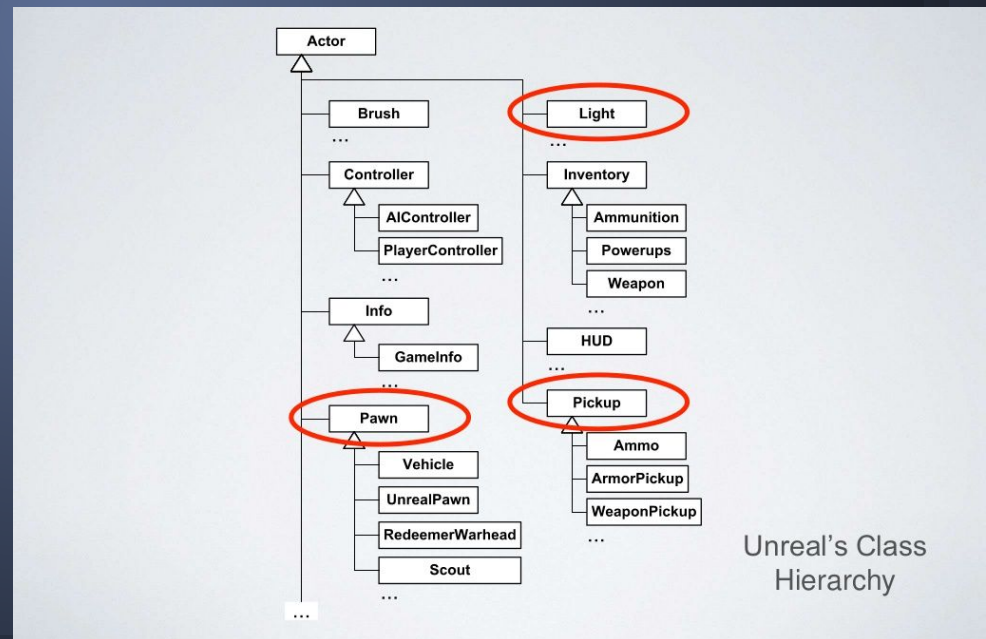
Info: <https://cedric-neukirchen.net/docs/category/multiplayer-network-compendium>

Armazón de jugabilidad

- World: Mundo virtual sobre el que trabajar, generado en base a uno o más Levels
 - Level: El espacio mínimo donde poner Actors
 - Se pueden cargar varios a la vez, o irlos cargando según se necesiten (*streaming*)
 - Si hay varios, uno debe ser el nivel principal (persistente) y el resto subniveles
 - Map: Asset (*.umap) que contiene un Level único o uno principal con referencias a sus subniveles

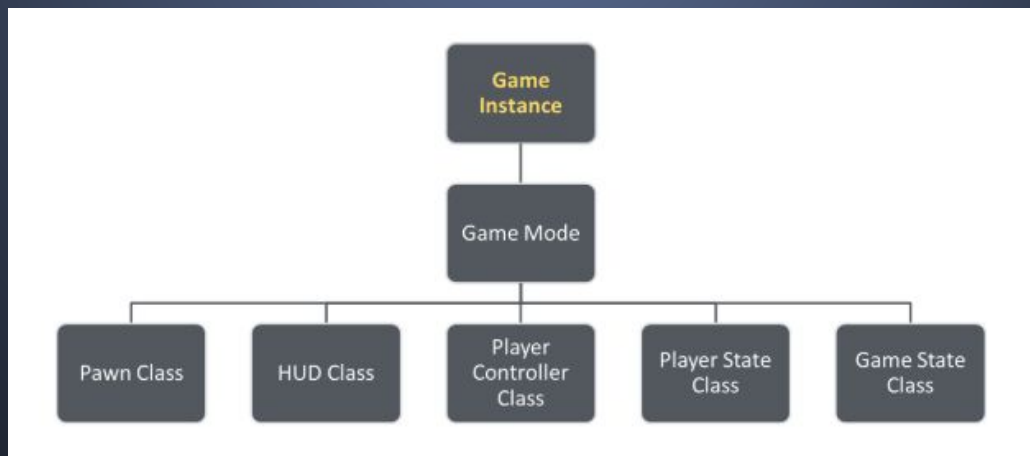
Armazón de jugabilidad

- Actor: *Objeto* que puede **estar en el mundo**
 - Pawn: Puede ser **poseído por un controlador**
 - Character: Con **movimiento humanoide**
 - WheeledVehicle: Con **movimiento sobre ruedas**
 - StaticMeshActor
 - SkeletalMeshActor
 - PointLight
 - BoxTrigger
 - AmbientSound
 - ParticleEmitter
 - ...



Armazón de jugabilidad

- GameMode: se crea al cargar cada nivel y **controla toda la jugabilidad** (típico manejado desde el “servidor”)
 - Game State Class
 - PlayerController Class y Player State Class
 - HUD Class
 - Default Pawn Class
 - Session, Spectator, Replay Spectator ... classes

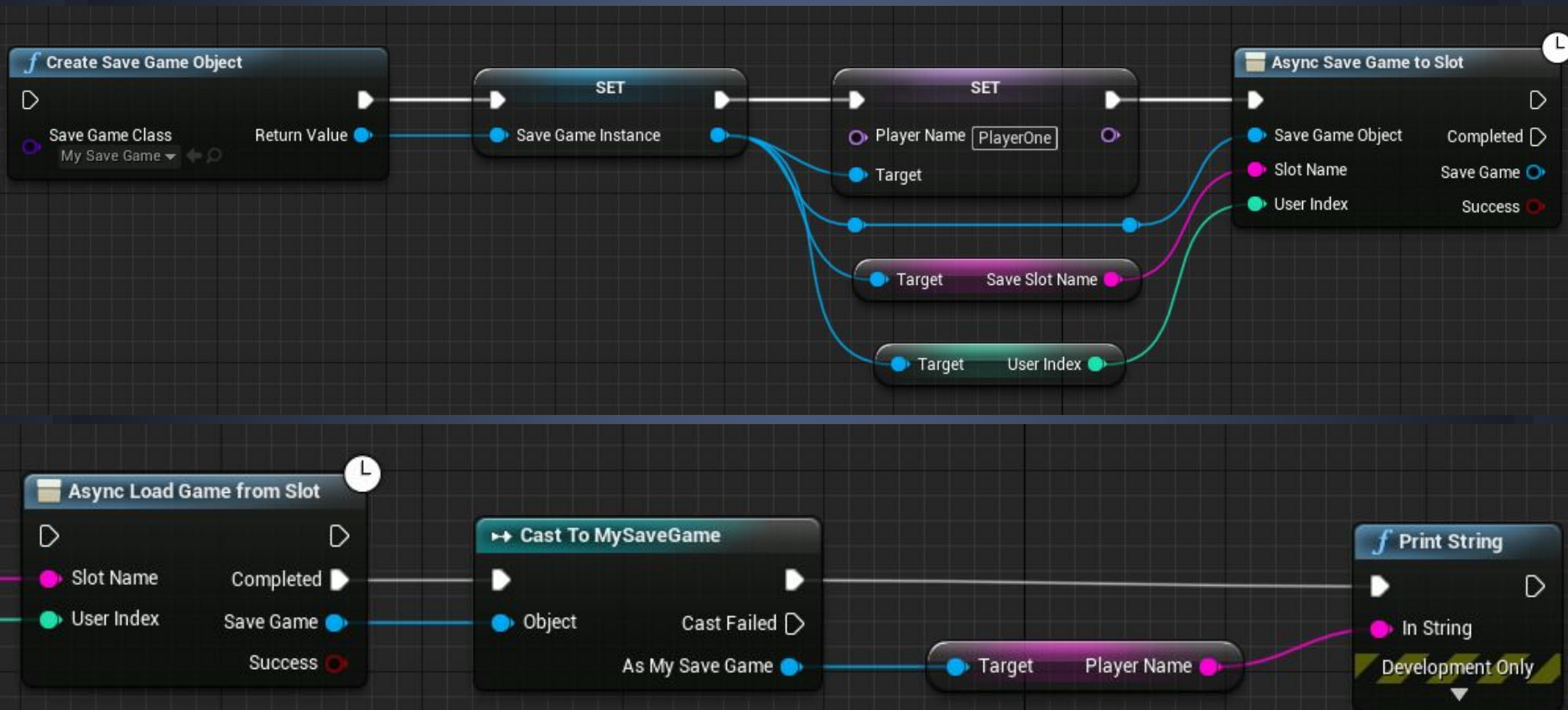


Armazón de jugabilidad

- Game Instance: **permanece vivo toda la ejecución**, y almacena los datos (no *referencias*) que persistirán entre niveles, evitando el tener que **guardar partida** al concluir un nivel
- Game State: son generados y gestionados por *Game Mode*, cada uno **lleva los datos y el control de reglas para un momento concreto del juego** que decide el *Game Mode*

Ej. Persistencia del estado del juego

- Guardar y cargar partida



Armazón de jugabilidad

- PlayerController: se genera cuando un jugador entra al juego, **gestiona toda la entrada/salida** (ratón, cámaras, HUD...) y **da órdenes a los Pawn** que posea el jugador
 - Posee uno o más Pawns (y los controla totalmente)
 - Recibe el input de un jugador
 - Accede al HUD de ese jugador
 - Accede al PlayerCameraManager de ese jugador
- Player State: se genera al generarse un PlayerController y contiene info pública que no es específica del Pawn
- AI Controller: controlador/**bot** que maneja “la máquina”

Ejemplo

- Juego de naves **Bird of Prey**
 - **Actors:** Pickups
 - **Pawns:** Naves de jugadores, naves enemigas, proyectiles, jefes y torretas
 - **Game Mode:** Maneja las decisiones de juego primarias y la lógica. Cuando las naves son destruidas, sus puntos se evalúan aquí
 - **Game State:** Guarda la puntuación más alta y otras informaciones
 - **Game Instance:** Graba la nave actual del jugador



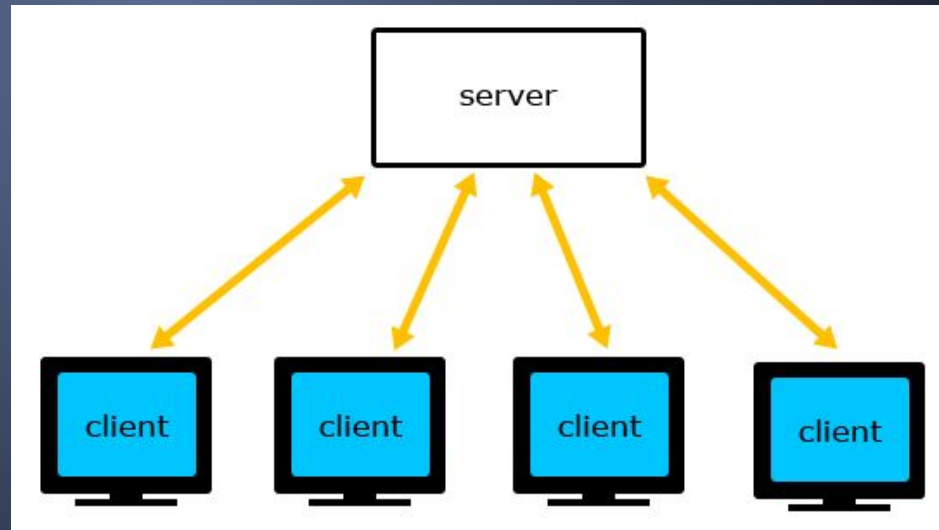
Participación

- ¿Qué componentes de Unreal Engine hacen falta para un buen **personaje protagonista**?
 - A. StaticMesh, Pawn y Character
 - B. SkeletalMesh, PlayerController y GameMode
 - C. StaticMesh, Pawn, PawnController
 - D. SkeletalMesh, Character y PlayerController
 - E. Character y PlayerMesh



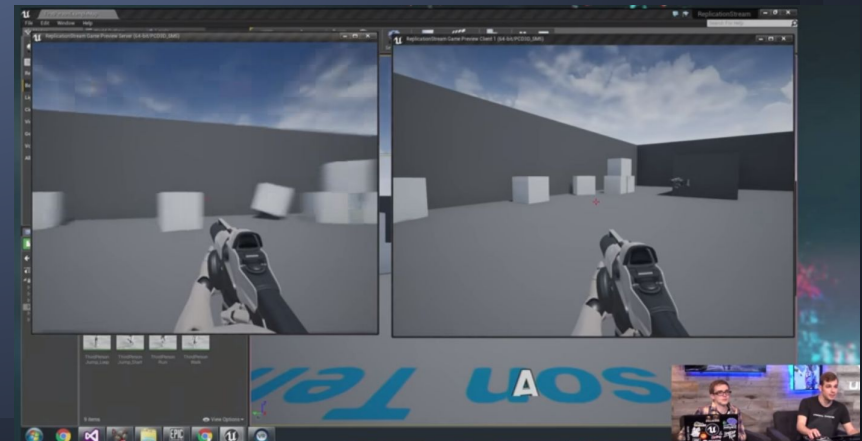
Videojuegos multijugador

- Las *experiencias multijugador en línea* actuales requieren **sincronizar muchísima información entre muchísimas máquinas** de jugadores alrededor de todo el mundo
- Sincronizar todo sería *demasiado costoso*, pero UE ofrece una solución 🧐

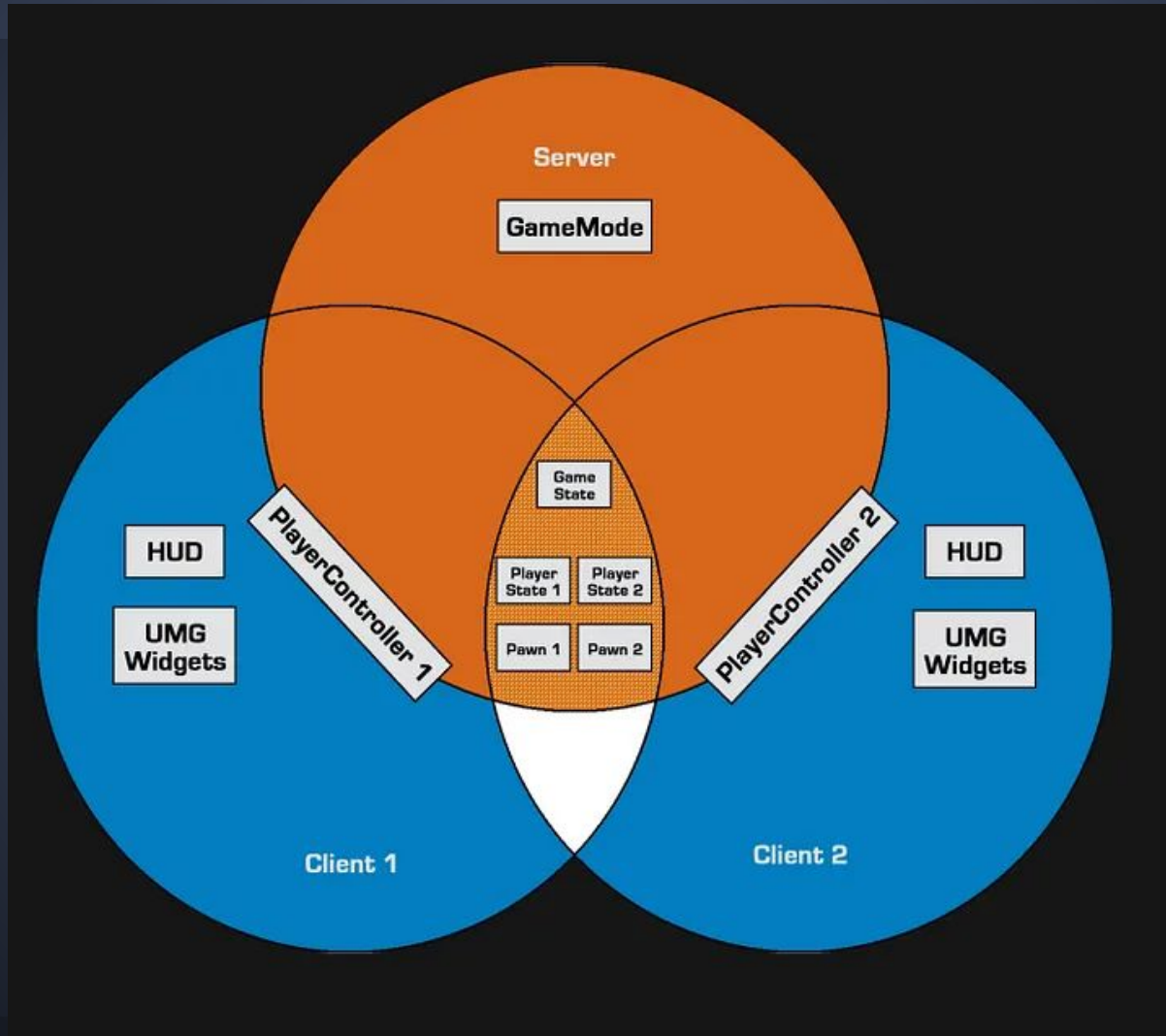


Videojuegos multijugador

- En Unreal Engine es el sistema **Replication** el que se encarga de sincronizar los **datos** y las **llamadas a procedimientos remotos** entre **clientes** y **servidores** REMOTE PROCEDURE CALLS (RPCs)
 - Esta **replicación** es clave: permite trabajar de forma *abstracta*, en alto nivel, a la vez que podemos personalizar *cómo funciona exactamente* la comunicación en red del juego, en bajo nivel



Arquitectura cliente-servidor



Arquitectura cliente-servidor

- El **servidor** es el *nodo “autoritario”* de la red, que se asegura de *replicar* lo que corresponda a todos los demás (**clientes**)
 - Existen 3 modos de red
 - **Cliente**
 - **Servidor dedicado**
 - **Servidor en escucha** (cliente y servidor en mismo proceso)
 - Incluso en **monojugador** o en **multijugador local**, ¡hace falta un cliente y un servidor... *siempre!*

Type	Command
Listen Server	UE4Editor.exe ProjectName MapName?List
Dedicated Server	UE4Editor.exe ProjectName MapName -ser
Client	UE4Editor.exe ProjectName ServerIP -ga
NOTE Dedicated servers are headless by default. If you don't use -log, you won't see any window	

Arquitectura cliente-servidor

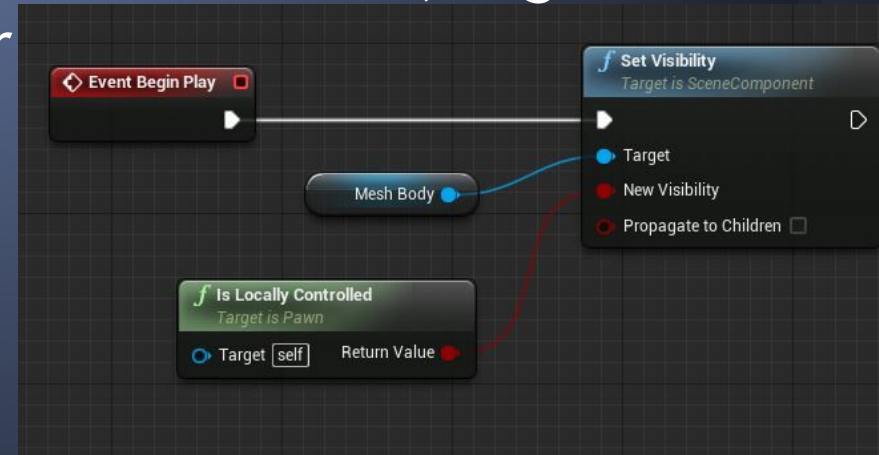
- En el servidor **se gestiona la jugabilidad**
 - Comienzo y final del juego
 - Cargas de niveles
 - Replicación en clientes (como **Game State**)
 - **Game Mode** (GM) sólo existe en el servidor
- Así funciona el sistema:
 1. El cliente solicita conectar
 2. Si el servidor acepta, le manda un mapa a cargar
 3. Tras la carga, se ejecuta el prelogin del GM
 4. Si se acepta, se ejecuta el login del GM
 - Se crea y replica el verdadero **PlayerController**
 - Se le llama a BeginPlay
 - Se ejecuta el postlogin de GM y ya podemos usar RPC

Replicación

- Así funciona la replicación
 - En la práctica el servidor informa del *estado de los actores* que han cambiado mediante **notificaciones** (“actualizaciones”) **regulares** a todos sus clientes
 - Las **propiedades** (ej. **salud**), al ser datos, se sincronizan solas, pero las **acciones** (ej. **una explosión**), al ser llamadas a procedimientos remotos, se replican sólo si expresamente así lo hacemos (recuerda que *no son funciones ni pueden devolver nada*)
 - Los **PlayerControllers**, por ejemplo, sólo se replican en un cliente: el del jugador correspondiente

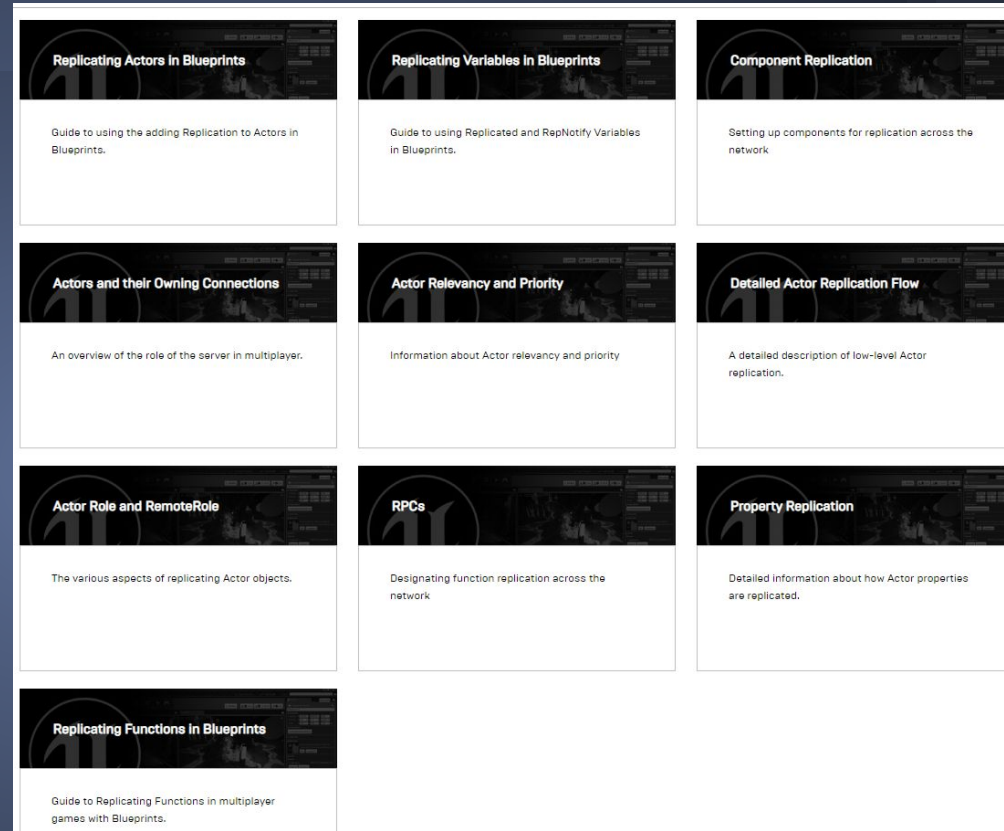
Replicación

- Hay **componentes de Unreal Engine** como **CharacterMovement** que ya realizan replicación de forma automática
 - Aún así, a un **Pawn** le podemos preguntar **IsLocallyControlled** para mostrarlo o no, según sea el que maneja el jugador o sea el de otro jugador
- Las **mallas estáticas** pueden replicar igual si marcamos **Static Mesh Replicate Movement**



Replicación

- Cómo replicar actores en general
 - Cambiando atributos de actor como *Net Load on Client* (el servidor carga niveles y se ven en los clientes) y *Replicates* (los cambios del servidor se replican en los clientes)



<https://docs.unrealengine.com/en-US/Gameplay/Networking/Actors/index.html>

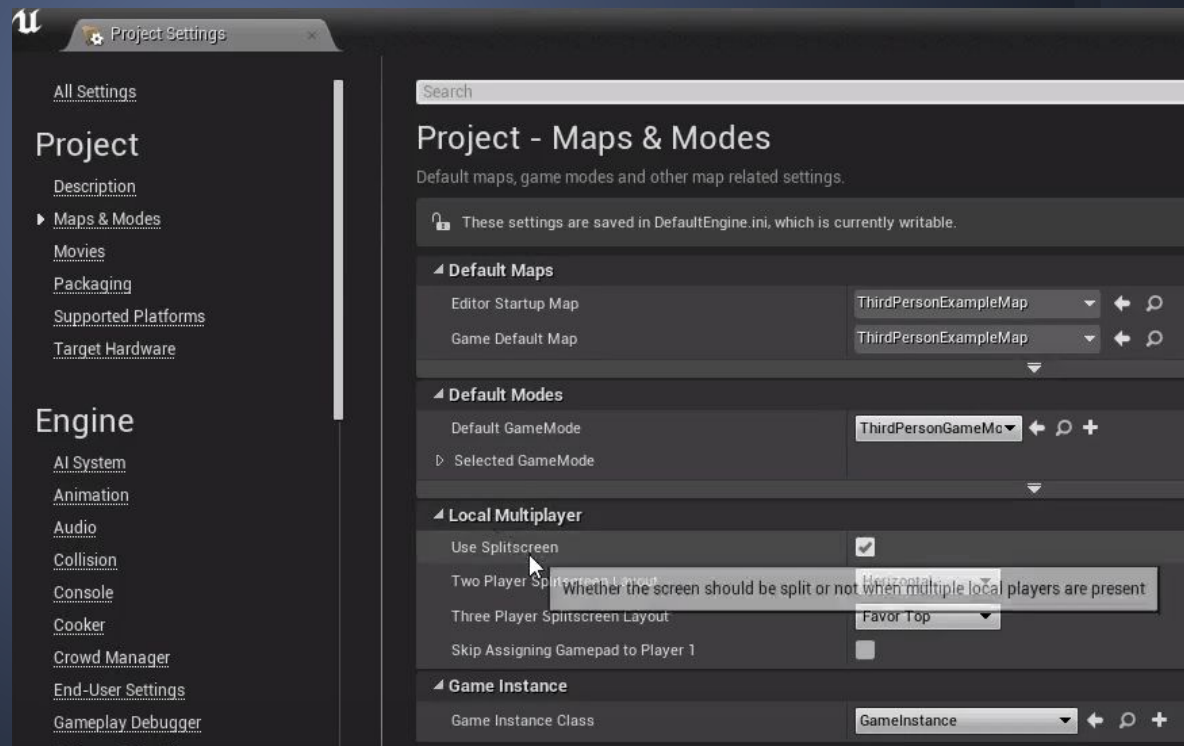
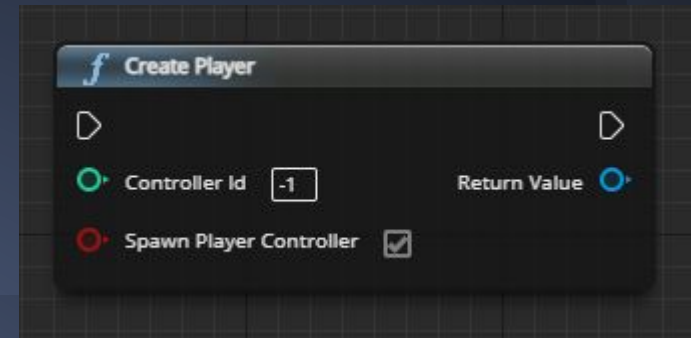


Replicación

- Como vemos, es fácil que todo lo que pasa en el servidor se replique en los clientes... pero *¿y lo que pasa en los clientes?*
 - Se notificará al servidor **todo lo importante**, para que se replique en *todos los clientes...* y sólo lo “cosmético” será exclusivo del propio cliente
 - Para eso se crea un **Custom Event** marcando **Run on Server** y **Reliable** al que se le llama siempre
 - Para saber **si algo se ejecuta en el servidor**, o si tengo **autoridad sobre un actor**, se usan las funciones along with **IsServer** y **HasAuthority**

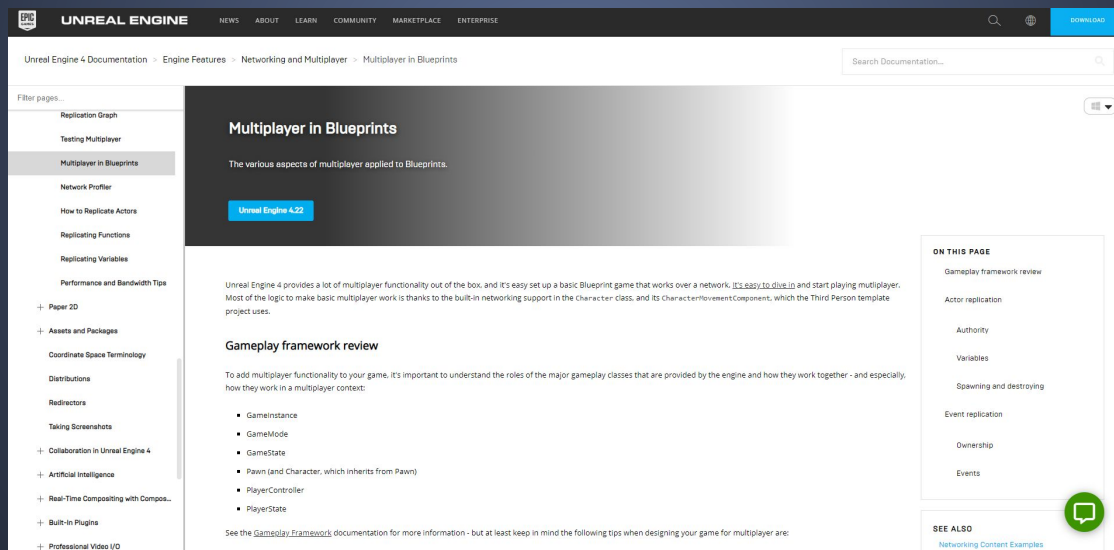
Multijugador local

- Siempre hay un jugador, pero **podemos crear más** (y asignarles *controladores*)
- En un juego multijugador, aunque sea local es importante **respetar el armazón de jugabilidad**



Multijugador en Blueprints

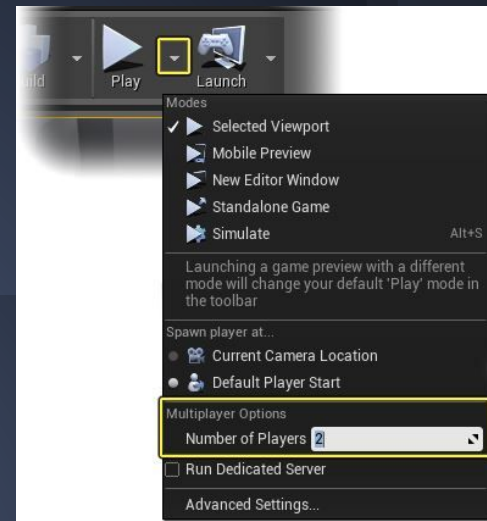
- Aquí están todos los detalles a comprender



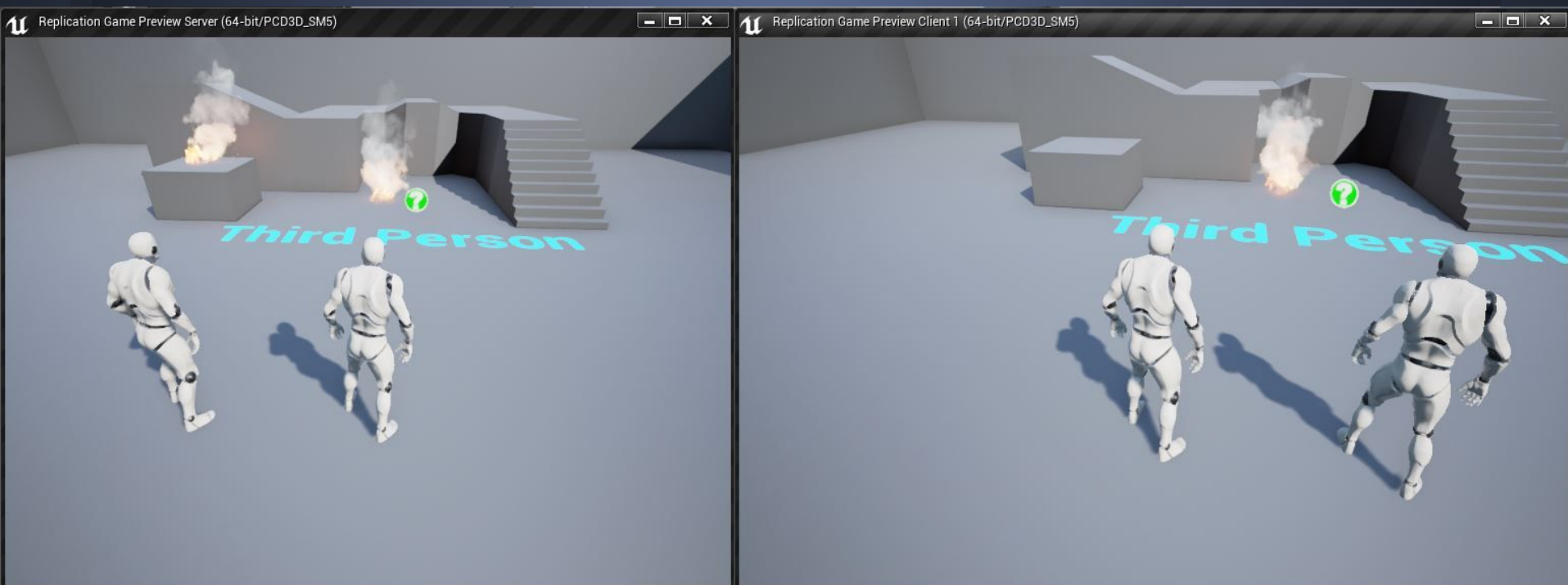
<https://docs.unrealengine.com/en-US/Gameplay/Networking/Blueprints/index.html>

Pruebas en red

- Unreal Engine tiene un **sistema** para probar multijugador



<https://docs.unrealengine.com/en-US/Gameplay/HowTo/Networking/TestMultiplayer/index.html>



Pruebas en red

- Pueden crearse accesos directos en Windows para **lanzar el servidor y los clientes** cómoda y rápidamente

```
MyGame.exe /Game/Maps/MyMap
```

```
UE4Editor.exe MyGame.uproject /Game/Maps/MyMap?game=MyGameInfo -game
```

```
UE4Editor.exe MyGame.uproject /Game/Maps/MyMap?listen -server
```

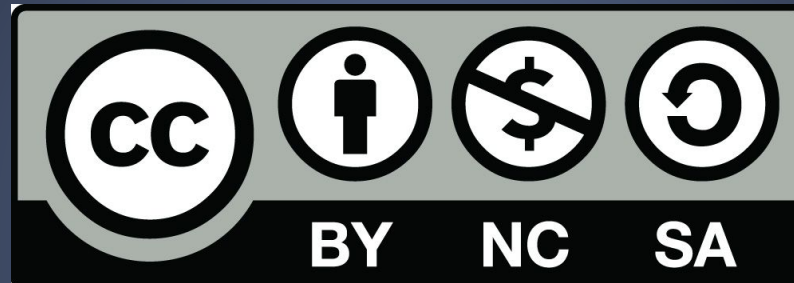
```
MyGame.exe 127.0.0.1
```

Empaquetar el proyecto

.EXE con carpetas
Engine y Content

- Configura el proyecto para empaquetarlo
 - Project
 - Description: La información del juego
 - Maps & Modes: El nivel donde vamos a empezar
 - Packaging: Use Pak File, por seguridad; Build Configuration a Shipping; Staging Directory carpeta de destino; Full Rebuild activa...
 - Supported Platforms: sólo las que necesites
 - Platforms / Windows
 - Icono del juego, pantalla de carga...
- Desde el editor lanza Platforms / Windows / Package project

Críticas, dudas, sugerencias...



** Licencia sólo aplicable al texto original de estas diapositivas*

Federico Peinado (2019-2026)

www.federicopeinado.es