# Unreal

* **Blueprints**:
  + Prototipaje
  + Es muy difícil hacer código compartido.
  + Es rápido iterar
* **C++**:
  + Iterar en C++ es lento
  + Mejor performance

Pasar de Blueprint a C++ es fácil. El problema es

* **Git**: problema para los artistas 🡪 Hay que activar LFS (para subir ficheros grande)
  + El host nos lo da Github
* **SVN**:
* **Ph** (perforse o algo así pero hay que pagar)
* **Plastic**: unity y se puede poner tamb con Unreal. Va bastante bien.

En unreal abajo a la derecha saldrá una pantallita verde que dirá que estamos conectados al Git

C++ EDITOR JUEGO

Unreal está definido por módulos. Un modulo es una entidad, que tienen una carpeta

* **Public** (\*.h que quiero que la gente vea). Si creo un actor TorreEnemiga pues va a ahí.
* **Private**: cpp y .h 🡪 no la podemos ver desde fuera del módulo

Nuestro juego va a ser un modulo (uno especial).

Puedo tener un mcvgameeditor (tools para el juego)

Carpeta plugins: son más módulos. Q tienen su parte publica y privada. Yo puedo crear un plugin para splines por ejemplo, y se pueden generar dependencias entre plugins

Luego yo genero un .uproject y es el master que usa unreal para generar la solution (.sln) y esto es lo que usamos pa compilar. Eso de añadir ficheros al proyecto, dependencias, precompiler headers… y tal… NO SIRVE PARA NADA. Visual es solo para editar.

Todo el engine está montado a partir de módulos. Cada modulo genera un .dll (transparente) que expone la parte pública.

Cada modulo tiene un nombre, todos los módulos generan una macro que tienes q poner todas las clases que vayas a exponer fuera.

Yo creo un modulo con nombre MY\_GAME y para exponer la clase le pongo la macro esta de API\_nombre.

Yo le pongo un CLASS API\_MYGAME

C++ de unreal tiene bastantes templates y preprocesador.

Yo me creo una clase .cpp y .h y pongo

Uclass (blueprint)

Class AMyBoss : public AActor {

}

Las clases de C++ de actores empiezan por A. Unreal coge todos los cpp y los preprocesa y genera un fichero. De un .h genera otra version para que se pueda exponer (son metadatos y para ello necesita que empiece por A).

La clase base de Aactor es UObject. Todos los UObjects son gestionados por Unreal. Y de vez en cuando pasa un garbaje collector q borra las objetos que no se van a usar (si un objeto no tiene referencias).

## Ptr a Clase

**NO podemos crear un puntero raw a clases, se usa UPROPERTY**

Class Enemy 🡪 AMyboss\* Boss. PELIGRO!! Si el boss se va a la porra no nos enteramos y el puntero F. En el engine estaban los Handle ahora nosotros tenemos una coas que es UPROPERTY q valida q el objeto es valido.

## También tenemos STRUCTS.

* Pueden tener métodos pero no si los pasamos a Blueprints,
* No tienen pq derivar de UObject

No hacer std::vector, std::string, std::map PROHIBIDO

* FString
* Fname (identificador, es como un string) -> Case sensitive pepe = PePe
* FText: texto localizado (para idiomas por ejemplo)

Todo unreal es Unicode. Todo los textos que pongamos hay que poner TEXT(“Hola”). Es una macro para convertir. Unreal tiene un montón de cosas para vectores, comparar strings, bla bla bla.

## Contenedores:

* TArray<>:
* TMap< , >

## Caos vs Physics

Epic debe mejorar todavía, hace la parte de colisiones bien, la parte de ropa mal.

## WIN32

No usar nada de WIN32. OpenFile de Windows y eso no

## Forward o deferred

* **Forwards** para móviles o VR (cosas que se pintan una vez). No toco nada.
* **Deferred**: Nanite no es experto el juan.

Editor precocinado y abrimos el UPROJECT. Esto genera una dll del juego. Es un modulo que se cargará y el myboss aparece por ahí. Yo estaré en el modo editor y puedo hacer cosas. El código del juego no deberíamos de poner nada del editor.

.EXE ahí NO PUEDE HABER NADA DEL EDITOR.

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Setting del projects

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Game instance**: para q algo sobreviva a una carga de niveles,

**Gamemode**: puede ser lobby / game

**PlayerController**

**World**: tiene un timer para eventos. Y cada world está asociado a un nivel. Los actores del nivel tienen un GetWorld acceder a él.

Static: solo hay uno

Los assets: son ficheros binarios.

Content > ThirdPerson > Blueprints > BP\_ThirdPersonCharacter

**CreateDefaultSubobject**: crea un componente

En los .h el ultimo include es el de la clase!!

Texto

Descripción generada automáticamente