# ¿Cómo se diseña un motor de videojuegos?

PID_00246080

Jorge Arnal Montoya

Tiempo mínimo de dedicación recomendado: 2 horas

Universitat Oberta de Catalunya

# Índice

# Introducción

En este material vamos a presentar al estudiante el concepto de *motor de videojuegos*, así como una breve historia de los motores que se han visto desde los inicios de la creación de videojuegos hasta nuestros días.

Dedicaremos un apartado a los motores actuales, y otro a los videojuegos que se están desarrollando con estos motores.

A continuación, veremos las diferentes API gráficas empleadas según las plataformas y los componentes que incluye un motor de videojuegos.

Por último, conoceremos los diferentes patrones y arquitecturas de programación que se utilizan en el sector.

# 1. Historia de los motores de videojuegos

Aunque hoy en día es muy común oír hablar de motores de videojuegos, si eres aficionado al mundo de los videojuegos o si perteneces al sector como desarrollador, en los inicios de la industria, los videojuegos eran productos realizados de manera muy artesanal.

Así pues, a lo largo de la historia del videojuego, podemos ver cómo las empresas punteras creaban sus propios motores de videojuegos, lo que les permitía poder desarrollar de forma más rápida y eficiente sus nuevos productos.

A continuación, vamos a echar la vista atrás para ver los primeros motores de videojuegos.

## 1.1. SCUMM (Script Creation Utility for Maniac Mansion)

A medio camino entre un motor y un editor para crear aventuras gráficas, durante la época de los ochenta y los noventa, los desarrolladores de LucasArts crearon este conjunto de herramientas que les permitieron realizar videojuegos de forma rápida. Algunos juegos que se desarrollaron con esta tecnología son obras maestras de la época: Maniac Mansion, Loom, Monkey Island, Indiana Jones and the Last Crusade, etc.

## 1.2. AGI (Adventure Game Interpreter)

En la misma época que LucasArts realizaba sus aventuras gráficas con el motor SCUMM, Sierra, otra de las grandes desarrolladoras de aventuras gráficas de la época, utilizaba el motor AGI para acelerar el proceso del desarrollo de cada uno de sus videojuegos. Juegos tan conocidos en la época (y que han trascendido hasta la actualidad) como Larry, King's Quest, Space Quest… utilizaron esta tecnología.

## 1.3. Westwood 3D-SAGE (Strategy Action Game Engine)

En este caso nos encontramos con la compañía Westwood Studios, que crea su propio motor para videojuegos de tipo estrategia en tiempo real. Juegos como Earth & Beyond o la saga Command & Conquer hicieron uso de este motor.

## 1.4. RenderWare

Este motor, desarrollado por Criterion Software, permitía a los desarrolladores crear videojuegos en multiplataforma, es decir, creaban el juego una vez y podían desplegar versiones del mismo para diferentes plataformas, con soporte para PC, MAC y consolas de sobremesa, como PS2, Xbox, Gamecube, PS3,

Xbox 360 y Wii; o plataformas portátiles, como GBA, PSP y NDS. Fue un motor muy utilizado en infinidad de proyectos y compañías. Criterion Software es la compañía creadora de la saga de coches Burnout y podemos encontrar juegos licenciados a terceros, como Battlefield 2 Modern Combat (PS2, Xbox, Xbox 360), Call of Duty: Finest Hour (Gamecube, PS2, Xbox) y GTA III-Vice City-San Andreas (PC, PS2, Xbox…). Como podemos apreciar, es un motor que ha sido utilizado por infinidad de videojuegos que han cambiado la industria.

### 1.5. Gamebryo

Es un motor muy utilizado por diferentes desarrolladores, creado originariamente por la empresa NDL. Nos encontramos con otro motor multiplataforma de la generación de consolas PS2, Xbox y Gamecube y la siguiente generación: Xbox 360, PS3 y Wii. Juegos como Fallout 3, The Elder Scrolls III: Morrowind y Civilization Revolution han hecho uso de este motor.

### 1.6. Quest 3D

Se trata de un motor 3D creado por la empresa holandesa Act-3D B.V. En este caso nos encontramos con un producto que no solo se utiliza en el sector de videojuegos, sino que también empresas de arquitectura u otras no enfocadas al entretenimiento lo utilizan por su facilidad para crear aplicaciones interactivas en tiempo real. Su metodología no está tan orientada al *script*, sino que permite al desarrollador utilizar un editor mediante el uso de cajitas, lo que hace la programación totalmente visual.

# 2. Motores actuales

Como podemos ver, la idea de crear un motor que pueda ser utilizado en diferentes proyectos, incluso para desplegar nuestro videojuego en distintas plataformas, no es algo nuevo, sino que la industria siempre ha buscado la forma de mejorar los procesos de desarrollo.

En los últimos años, hemos presenciado la creación de nuevos motores y herramientas que han permitido la democratización en la creación de los videojuegos, haciendo que la barrera de entrada al desarrollo bajara hasta límites en los que personas sin formación universitaria pueden desarrollar pequeños videojuegos. Así hemos visto propuestas totalmente originales y alejadas de las clásicas que podemos encontrar en los videojuegos AAA de las grandes compañías.

Algunos de los motores que se utilizan actualmente los veremos a continuación.

## 2.1. Unity

El motor de la compañía Unity Technologies, de origen danés, es posiblemente la principal referencia en el cambio de modelo de negocio en el mercado de los motores de videojuegos. A través de un editor de tipo WYSIWYG y un motor que nos permite desplegar en multitud de plataformas, este proyecto facilitó la entrada a personas interesadas en la creación de videojuegos, acompañado de un sistema de precios muy bajo y accesible para desarrolladores *amateurs*. La llegada de plataformas de distribución digital como las plataformas móviles hizo que una gran cantidad de personas empezasen a desarrollar sus propios videojuegos.

**Dato**

En el año 2016, el 34% de los 1.000 juegos gratis más usados en dispositivos móviles fueron creados con Unity.

## 2.2. Id Tech

Es el motor creado por la empresa Id Software, creadora de los videojuegos clásicos Wolfenstein, Doom o Quake. Ha tenido diferentes iteraciones hasta llegar a la versión actual, que está siendo utilizada por los proyectos que desarrolla la empresa matriz de Id Software, Zenimax. En este conglomerado de empresas, encontramos desarrolladores como Bethesda (The Older Scrolls, Fallout), Arkane (Dishonored) o la propia Id Software.

## 2.3. Unreal Engine

Si actualmente encontramos en Unity un motor muy utilizado por la mayor parte de desarrolladores independientes del mundo, Unreal Engine, creado por Epic Games, fue uno de los motores más utilizados para videojuegos AAA

en la generación de consolas PS3 y Xbox 360. Con la llegada de Xbox 360, se presentó Gears of War como referente de Unreal Engine, un videojuego que mostraba las bondades de este motor. Gears of War, Batman Arkham Asylum, Batman Arkham City, Borderlands... son algunos de los videojuegos que muestran todo el potencial de esta pieza tecnológica.

### 2.4. CryEngine

Creado por la gente de Crytek, situada en Alemania, este motor siempre ha estado a la vanguardia de la tecnología. Videojuegos como Farcry o Crysis han sido utilizados en su momento como *benchmarks* de cualquier ordenador de la época.

### 2.5. Anvil

Este motor ha sido utilizado por Ubisoft en sus proyectos AAA en sus estudios de todo mundo. Ubisoft, con su metodología de proyectos multiestudio, tiene proyectos que se desarrollan en diferentes estudios situados en múltiples lugares del mundo. El motor utilizado para videojuegos como Assassin's Creed ha sido Anvil Engine.

### 2.6. Fox Engine

Antes de la traumática salida de Hideo Kojima de Konami, el controvertido creativo participó en el diseño de este motor, Fox Engine. Con un futuro incierto, ha sido utilizado en los videojuegos Metal Gear Solid V o Pro Evolution Soccer.

### 2.7. Frostbite

Posiblemente este es el motor que en la actualidad está llamando más la atención. Creado por los suecos EA DICE, ha traído videojuegos como Star Wars Battlefront, Battlefield I, Mirror's Edge Catalyst…

### 2.8. Decima

Este motor ha sido desarrollado por la *first party* de Sony Guerrilla Games, situada en Holanda. Es el responsable de proyectos que han mostrado las bondades de las plataformas de Sony, y nos han traído videojuegos como Killzone, en PS2; Killzone II, en PS3; Killzone Shadow Fall, en PS4; Killzone Mercenaries, en PSVita, o el reciente Horizon Zero Dawn. Son componentes que demuestran las capacidades de este motor. Hideo Kojima lo ha elegido para su próximo proyecto, Death Stranding.

## 2.9. Source

Antes de conocerse a Valve como la plataforma digital Steam[1] , era famosa por videojuegos como Half Life o Counter Strike, entre otros. Dichos videojuegos fueron desarrollados bajo el paraguas del motor Source..

(1)La plataforma Steam es hoy en día la mayor plataforma de venta de videojuegos en formato digital para PC.

# 3. API gráficas

Uno de los componentes más importantes en un motor de videojuegos es la API gráfica que utiliza. Algunos motores, especialmente los multiplataforma, no se basan en una única API, sino que pueden utilizar varias.

A continuación, definiremos las diferentes API que pueden incorporar un motor.

## 3.1. DirectX

Es una API gráfica creada por Microsoft para su sistema operativo Windows. Si bien es una biblioteca que existe desde mediados de los años noventa, es con su versión 8, y especialmente con la 9, cuando se asienta totalmente en el desarrollo de videojuegos 3D. En la actualidad, en su versión 12 es utilizada tanto en PC como en Xbox One.

## 3.2. OpenGL

Es una API gráfica creada por Silicon Graphics a principios de los noventa. Se trata de una biblioteca gráfica multiplataforma que nos permite presentar gráficos 2D y 3D. Todas las plataformas actuales del mercado soportan OpenGL o alguna API que provenga de esta, adaptada a la plataforma.

## 3.3. Vulkan

Es una API gráfica presentada en la GDC 2015 por el grupo Khronos, que es el *board* que gestiona actualmente OpenGL, y que viene a competir con DirectX 12, puesto que funciona en cualquier plataforma, incluso las no basadas en Windows.

## 3.4. WebGL

Es una API gráfica basada en OpenGL y que permite el uso de gráficos 3D en navegadores web.

## 3.5. Metal

Encontramos nuevamente una API gráfica basada en OpenGL y pensada para los dispositivos que utilizan el sistema operativo iOS de Apple.

# 4. Componentes de un motor de videojuegos

Como hemos visto, uno de los elementos más importantes en un motor de videojuego es el componente de renderizado o gráfico basado en una API gráfica. Como hemos comentado, un motor es más que una API gráfica. A continuación, definiremos aquellos componentes que debe incluir un motor de videojuegos.

## 4.1. Matemático

La biblioteca matemática nos dará toda la funcionalidad de cálculo matemático. En un motor 3D, por ejemplo, podrá realizar operaciones matemáticas con vectores, matrices o cuaterniones, entre otras.

## 4.2. Gráfico

Como hemos visto, este componente encapsulará las funcionalidades de la biblioteca gráfica que estemos utilizando en nuestro motor. Encapsulará, además de la parte de renderizado específica de la plataforma, elementos como las estructuras de vértices, las mallas de triángulos o la parte de *shaders* según la plataforma y la API utilizada.

## 4.3. *Input*

Gracias a este componente, podremos acceder a la información de entrada del usuario con el juego. Los elementos que podremos leer a través de esta biblioteca serán periféricos, como el ratón, el teclado o el *pad*, así como bibliotecas más específicas por plataforma, como pantallas táctiles; lectura de controles gestuales, que pueden ser introducidos a través de cámaras de profundidad, como Kinect, o mandos de control por movimiento, como los Wiimotes.

## 4.4. Física

Esta biblioteca será la responsable de toda la implementación de la parte física de nuestro motor, así como de las colisiones que afectan a nuestro videojuego.

## 4.5. Sonido

A través de esta biblioteca, podremos reproducir efectos de sonido o música en nuestro videojuego.

### 4.6. AI

Elementos típicos que podemos encontrar en este componente son conceptos como el *pathfinding*.

### 4.7. Red

Muchos de los videojuegos actuales están pensados como servicios, y no como simples productos de un único consumo; es por ello por lo que un componente como el de red es muy utilizado. Su implementación está basada en *sockets*, pero podemos encontrar implementación de servicios web, *websockets* y cualquier tipo de comunicación que nos pueda interesar para nuestro videojuego.

### 4.8. Editor

Este componente nos va a permitir hacer un videojuego de una manera visual y facilitar a diferentes perfiles la creación del mismo. Especialmente pensado para diseñadores, hoy en día tenemos un editor como Unity, que permite una gran independencia en el trabajo de los artistas, que pueden así crear perfectamente las escenas del videojuego sin la necesidad de tener a un programador al lado para introducir los elementos en el mismo.

### 4.9. *Gameplay*

Este es el último componente de nuestro videojuego y el único que realmente no debe pertenecer al motor, ya que aquí es donde encontraremos todo el código que implementa nuestro videojuego.

# 5. Patrones de programación utilizados en videojuegos

Igual que cualquier otra pieza de software desarrollada actualmente, en videojuegos podemos encontrar patrones de programación de sobra conocidos. A continuación, recordamos algunos de estos patrones muy utilizados en videojuegos.

## 5.1. Arquitectura basada en herencia

La programación orientada a objetos es de sobra conocida por todos los programadores y no es nueva en la formación universitaria. El desarrollo de videojuegos se basa principalmente en la programación orientada al objeto; conceptos como *público*, *protegido* o *privado* tienen que ser de sobra conocidos por el estudiante. *Herencia*, *sobrecarga de métodos* u *operadores* son también aspectos que todo desarrollador de videojuegos debe conocer.

## 5.2. Arquitectura basada en entidad-componente

Una de las principales desventajas de la arquitectura basada en herencia es que, a medida que el código se va haciendo más complicado, el mantenimiento de las clases aumenta su complejidad exponencialmente.

Es aquí donde una arquitectura como la entidad-componente se está imponiendo en videojuegos. La idea de esta arquitectura parte de la necesidad de que los elementos de nuestro juego tengan características que sean independientes de otras características del mismo objeto. Por ejemplo, podemos tener una entidad *player* que tenga un componente que entienda a su controlador físico y otro componente que entienda cómo renderizar el objeto; estos no necesitan saber el uno del otro, y esta pequeña división del objeto hace mucho más sencillo el mantenimiento del código y su ampliación.

Por tanto, tendríamos, por un lado, las entidades que contienen un conjunto de componentes, y cada uno de esos componentes ejerce una funcionalidad sobre la entidad.

Un ejemplo de motor basado en la arquitectura entidad-componente es Unity.

## 5.3. *Singleton*

El patrón *singleton*[2] nos permite tener una instancia única de nuestro objeto, y que sea accesible y siempre la misma en todo nuestro código.

(2)El patrón *singleton* lo utilizaremos en nuestro código para la clase motor.

Por ejemplo, en el caso del motor de videojuegos, como desarrolladores no queremos tener dos motores de juego funcionando a la vez. Por ello utilizaríamos un patrón *singleton*, para que cada vez que necesitemos acceder al motor, accedamos al mismo y único motor en el juego.

Además, el patrón *singleton* nos asegura que nadie pueda crear una instancia del objeto más allá del mismo objeto; en el ejemplo del motor, nadie podría crear una instancia de la clase motor más allá de la clase motor.

### 5.4. Cliente-observador

Este es otro tipo de patrón muy utilizado en el desarrollo de software. Nos permite que un objeto cliente se registre a un objeto observador. Cuando el objeto observador genera un evento, este se notifica a cada uno de los clientes registrados al observador.

### 5.5. *Pool*

*Pool* es un concepto muy utilizado en videojuegos. La idea de esta implementación es la no creación constante de objetos muy volátiles. Está especialmente pensada para evitar la fragmentación de la memoria.

Imaginemos el concepto de una bala de juego; evidentemente, en nuestro videojuego tendremos una gran cantidad de balas, que se crearán y se destruirán de forma muy concurrente.

El concepto de *pool* lo que hace es crear un número máximo de balas que puede haber en pantalla y reutilizar estas balas. Cuando queramos dejar de utilizarla una bala, la ocultaremos, y cuando queramos volver a crear una bala, reinicializaremos una de las balas en la posición que nos interese y la volveremos a hacer visible.

### 5.6. *Factory*

El patrón *factory* nos permite crear un objeto que hereda de una clase base de forma transparente.

Imaginemos el caso en que nuestro juego contiene diferentes tipos de enemigos (estático, patrullero, volador, *boss*...). Así, en el videojuego contendremos una lista de enemigos con todos los enemigos de nuestro nivel.

Desde el punto de vista de la implementación, tendremos una clase base enemigo y una clase para cada tipo de enemigo que herede de la clase base enemigo. El patrón *factory* creará un objeto del tipo diferente de enemigo, pero en nuestro caso lo almacenaremos en nuestra lista que contiene elementos de tipo enemigos básicos.

# 6. Código específico por plataforma

Para terminar este documento de motores de videojuegos, explicaremos cómo se puede implementar un método que haga acceso a un código dependiente de la plataforma en la que se ejecute.

Imaginemos el siguiente caso: tenemos un método que nos devuelve el valor de un *stick* analógico; dependiendo de la plataforma en la que ejecutemos, aunque el nombre del método sea el mismo, este método se encontrará en el componente de *input* de nuestro motor.

**Macros**

Las macros también se pueden utilizar para crear código de forma automatizada, como crearemos las clases de RenderableVertexs en nuestro motor.

```
1.  float CInputManager::GetHorizontalLeftStick()
2.  {
3.  #ifdef _WIN32
4.          return Win32GetHorizontalValueFn(WIN32_LEFT_STICK);
5.  #elif _PS4
6.          return PS4GetHorizontalValue(PS4_LEFT_STICK);
7.  #elif _XBOXONE
8.          return XboxOneGetHorizontalValue(XBOXONE_LEFT_STICK);
9.  #endif
10. }
```

Cuando compilamos este fragmento de código en C++, el compilador, dependiendo de las macros de preprocesado, incluye para compilar el código según la plataforma que estemos generando, y excluye el código no incluido en el #if. Por ejemplo, si estuviésemos compilando el código con la macro de preprocesado _PS4 activa, nuestro método para el compilador sería el siguiente:

```
11. float CInputManager::GetHorizontalLeftStick()
12. {
13.      return PS4GetHorizontalValue(PS4_LEFT_STICK);
14. }
```

# 7. Retos

En este primer material tendremos el reto de presentar un diagrama con los componentes que incluirían nuestro motor de videojuego y explicar qué funcionalidades tendrían cada uno de estos componentes en nuestro motor.

# Resumen

Como hemos podido apreciar, el desarrollo de un motor de videojuego es muy complejo e incluye una gran infinidad de componentes de diferente índole, componentes que hacen que sus desarrolladores tengan que tocar conceptos como bibliotecas matemáticas, sistemas de renderizado, reproducción de sonido, comunicación a través de red…

Posiblemente la creación de un motor de videojuegos sea una de las implementaciones de software más complejas que podamos encontrar, por la gran cantidad de conceptos diferentes que incluye.

Hemos empezado viendo los distintos motores de videojuegos que se han creado a lo largo de la historia, cómo se han utilizado en diferentes proyectos y cómo esos mismos motores permitían hacer tanto videojuegos diferentes como el mismo videojuego para distintas plataformas.

Por último, hemos visto patrones de programación y tipos de arquitectura que se utilizan en motores de videojuegos, como la arquitectura entidad-componente o la arquitectura orientada al objeto.

# Bibliografía

Agi development site. http://www.agidev.com

Writing a Game Engine from Scratch - Part 1: Messaging. http://www.gamasutra.com/blogs/MichaelKissner/20151027/257369/Writing_a_Game_Engine_from_Scratch__Part_1_Messaging.php

Multithreaded Game Engine Architectures. http://www.gamasutra.com/view/feature/130247/multithreaded_game_engine_.php

**Gregory, J.** Game Engine Architecture. https://www.amazon.es/Game-Engine-Architecture-Jason-Gregory/dp/1568814135

Script utility for Maniac Mansion Virtual Machine. https://www.scummvm.org/

Gamebryo technology. http://www.gamebryo.com/

Quest3D Amazing for fly-throughs and 3D simulations. http://quest3d.com/

Unity Engine. https://unity3d.com/

CryEngine. http://crytek.com/

Unreal Engine make something unreal. https://www.unrealengine.com/

Vulkan Industry Forged. https://www.khronos.org/vulkan/

Cómo instalar la versión más reciente de DirectX. https://support.microsoft.com/es-es/help/179113/how-to-install-the-latest-version-of-directx

OpenGL. The Industry's Foundation for High Performance Graphics. https://www.opengl.org/