

Análisis de motores de juegos.

Caso práctico

BK Programación acaba de recibir un inusual encargo: la realización de un videojuego para la promoción de una marca comercial de prendas deportivas. **Ada**, socia fundadora de la empresa, es designada como directora técnica del proyecto. De hecho, quiere empezar a trabajar en él cuanto antes para aprovechar la estancia de **Ana** en BK Programación como alumna en prácticas ya que posee conocimientos de diseño gráfico y multimedia que serían muy útiles. El equipo de desarrollo será coordinado por **Juan**, su tutor laboral, y que está encantado de introducirse en el mundo de los videojuegos a pesar de carecer de experiencia previa como programador en dicho campo. Por supuesto, **Ada** les proporcionará toda la ayuda y formación necesaria para llevar el producto a buen puerto.



Materiales formativos de FP Online propiedad del Ministerio de Educación, Cultura y Deporte.

Aviso Legal (*link: <http://www.educacion.gob.es/fponline/aviso-legal.html>*)

1.- Introducción a los videojuegos.

Caso práctico

Como hemos visto antes, **Juan** no tiene experiencia en la programación de videojuegos aunque se confiesa un ávido consumidor de juegos de videoconsolas. La empresa editora que ha realizado el encargo les ha dado libertad creativa, ya que no tienen una idea clara de cuál puede ser el mejor enfoque para promocionar la marca. Por ello, antes de juntar a su equipo **Juan** decide ponerse manos a la obra e investigar el origen de los videojuegos para buscar inspiración, estudiando de paso los distintos tipos y géneros que existen y viendo cómo trabaja un equipo de desarrollo.



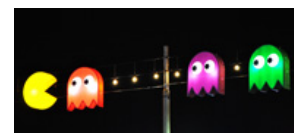
Todos tenemos una idea intuitiva más o menos clara de lo que es un videojuego. Como puedes deducir, hoy en día un videojuego no deja de ser más que un tipo de aplicación más que puede ejecutarse en un ordenador, dispositivo móvil o consola.

Reflexiona

¿Te has parado a pensar qué es lo que diferencia las aplicaciones de videojuegos de otros tipos de aplicaciones? ¿Dónde está la frontera entre una y otra?

Podemos definir un videojuego como un juego electrónico que interactúa con uno o varios jugadores con el fin de que se consigan ciertos objetivos y que muestra visualmente una respuesta a sus acciones.

Un videojuego tiene que informar mediante algún mecanismo al jugador o jugadores del resultado de sus acciones, ya sea visualmente, acústicamente, mediante vibraciones, etc. Normalmente, esa realimentación se realiza mediante la proyección de una imagen de vídeo en una pantalla, de ahí el nombre: videojuegos.



Puede llamarte la atención que no hayamos incluido el requisito de que sea entretenido o divertido. Eso es correcto, aunque es deseable, un videojuego no tiene por qué ser divertido de jugar al igual que ocurre, por ejemplo, al ver una película. Al cine se puede ir a pasar miedo (películas de terror), a ver historias tristes (dramas) o para estar en tensión constante (acción). ¿Quiere decir que esas películas no son buenas? Nada más lejos: **lo importante es que sea una experiencia agradable**, que te deje con ganas de repetir. Lo mismo ocurre con los videojuegos, como veremos más adelante cuando analicemos los géneros.

Autoevaluación

Marca las características que tendría que tener un buen videojuego:

- ☐ *(link:)*
Tiene que ser divertido.
- ☐ *(link:)*
Tiene que ofrecer una experiencia agradable.
- ☐ *(link:)*
No es necesario que se informe al jugador del resultado de sus acciones.
- ☐ *(link:)*
Siempre interviene al menos un jugador.

1.1.- Orígenes de los videojuegos.

Es difícil poner una fecha exacta al origen de los videojuegos, pero podemos comenzar en 1947. Fue en ese año cuando se registró una patente en Estados Unidos acerca de un dispositivo de entretenimiento que consistía en disparar a aviones en una pantalla de tubo de rayos catódicos (CRT). Thomas T. Goldsmith, Jr. y Estle Ray Mann aparecen como sus autores. Bien pudieron ser los padres del concepto de videojuego.

El Nimrod, construido en 1951, se considera primer ordenador diseñado exclusivamente para jugar al Nim, un juego donde jugador y máquina se alternaban para retirar palitos de unos montones.

Poco después, en 1952, Alexander S. Douglas programó una versión de las tres en raya (Noughts and Crosses) que se ejecutaba en el ordenador EDSAC de la Universidad de Cambridge en 1952. No tuvo mucha repercusión dado que solamente podía ejecutarse en el ordenador de dicha universidad.



William Higinbotham podría considerarse como el creador del videojuego multijugador. Su "tenis para dos" permitía a dos jugadores competir en el mismo juego. Se exhibió públicamente durante unas jornadas de puertas abiertas celebradas en su empresa. Aunque fue disfrutado por cientos de personas, no llegó al público adecuado y por tanto no condicionó la aparición de otros videojuegos.

En 1961, un estudiante del MIT de nombre Stephen Russell desarrolló un videojuego para dos jugadores llamado **Spacewar!** que funcionaba en ordenadores PDP-11. Posteriormente el juego fue mejorado por otros compañeros de Stephen, distribuyéndose como software de dominio público a través de la ARPAnet, la antecesora de Internet.

Pero no fue hasta comienzos de los años 70 cuando se extendió el uso de los videojuegos en las casas y en los salones de máquinas recreativas. Los responsables fueron Ralph Baer y Nolan Bushnell, el primero mediante el diseño y la comercialización de la primera consola de videojuegos: la Magnavox Odyssey, y el segundo por la fundación de una de las empresas de videojuegos más influyentes de la historia: Atari. Esta compañía creó innumerables máquinas recreativas y la primera consola que fue un éxito de ventas: la Atari 2600.

A partir de ahí, todo fue historia.

Reflexiona

¿Te has dado cuenta de la velocidad con la que evoluciona la tecnología de los videojuegos? Compara cualquier videojuego actual con los de hace tan solo 15 años y verás la diferencia.

Para saber más

Si tienes curiosidad por saber en qué consistía el juego de Nim, puedes visitar este enlace. Concretamente, la versión que jugaba el Nimrod era el conocido como "juego de Marienbad".

El juego al que jugaba el Nimrod. ([link: http://html.rincondelvago.com/juego-de-nim.html](http://html.rincondelvago.com/juego-de-nim.html))

Puedes ver en funcionamiento de uno de los primeros videojuegos documentados de la historia siguiendo este enlace.

Resumen textual alternativo ([link: PMDM05_Descripcion_video_primeros_videojuegos.html](http://PMDM05_Descripcion_video_primeros_videojuegos.html))

En el siguiente enlace podrás comprobar cómo era el primer videojuego multijugador: **Tenis para dos**.

Puedes echarle una partida al Spacewar! original de Stephen Russell si tu navegador Web soporta Java y sigues este enlace.

Spacewar! (*link: <http://spacewar.oversigma.com/>*)

1.2.- Los videojuegos en la actualidad.

Mucho ha cambiado en el campo desde los tiempos de esos primeros videojuegos. Los de hoy en día pueden llegar a ser auténticas producciones multimillonarias que superan en presupuesto a las grandes películas de Hollywood e involucran a cientos de personas.

Hoy en día, la industria de los videojuegos está centrada en 3 grandes plataformas: los ordenadores personales, las consolas y los dispositivos móviles. Hace unos años tendríamos que haber mencionado las máquinas recreativas, pero la democratización de los ordenadores y las consolas han terminado por relegarlas prácticamente al olvido.



Por lo general, los saltos tecnológicos aparecen primero en los ordenadores ya que están en permanente evolución y sus posibilidades de expansión permiten la inclusión de nuevos tipos de dispositivos. Eso sucedió es su momento con la introducción de las tarjetas de sonido, los gráficos 3D o las tarjetas de aceleración de cálculos físicos. Las novedades más demandadas pasan poco después al mundo de las consolas. A su vez, los dispositivos móviles son cada vez más potentes y van adquiriendo poco a poco las características técnicas de los otros dispositivos antes mencionados.

Programar para una de estas plataformas implica una serie de ventajas y de inconvenientes respecto a las demás. Veamos algunos aspectos interesantes:

- ✓ Consolas: Son dispositivos diseñados exclusivamente para ejecutar videojuegos. Su hardware evoluciona con cada iteración (que en ellas se llama "generación"), lo que suele suceder cada varios años. Generalmente, las capacidades de una consola en el momento del lanzamiento son punteras aunque permanecen constantes durante la vida del producto. Eso implica que con el paso del tiempo se quedarán obsoletas, dando paso a una nueva generación. La gran ventaja desde el punto de vista de la programación de videojuegos es que el hardware no varía durante este tiempo lo que permite concentrar los esfuerzos en una única plataforma. De cara al usuario final, su uso es mucho más simple que el de un ordenador personal. La generación actual de consolas a la fecha de escribir este texto está formada por la Sony PlayStation 3, la Microsoft Xbox360 y la Nintendo Wii.
- ✓ Ordenadores personales: Dado que son dispositivos de propósito general, los ordenadores personales han sido uno de las primeras plataformas en permitir la ejecución de juegos. Dado que continuamente están saliendo al mercado nuevos procesadores, tarjetas gráficas, memorias, etc., se produce un crecimiento lento pero firme en las capacidades de los ordenadores. Por ello, los equipos de última generación casi siempre van a superar a las consolas en potencia y memoria ya que adaptan las nuevas tecnologías con mayor velocidad. La cruz de la moneda es que los videojuegos tienen que soportar un abanico muy amplio de hardware ya que la configuración de dos ordenadores personales pueden ser muy distintas y eso necesita una inversión extra de tiempo y dinero durante el desarrollo.
- ✓ Dispositivos móviles: Los últimos en llegar han sido los dispositivos móviles. Si bien hace años que existen las videoconsolas portátiles (Nintendo Gameboy, Atari Lynx, Sega GameGear, Nintendo DS, Sony PSP, etc.) el concepto ha evolucionado para dar soporte a teléfonos móviles y a otros dispositivos similares (por ejemplo, reproductores de música). Concretamente, la aparición de los smartphones con grandes capacidades gráficas y con un sistema operativo de grandes prestaciones (iOS, Android, etc.) ha disparado el consumo de videojuegos en este tipo de dispositivos.



Características de las plataformas de desarrollo de videojuegos

Característica	Evolución de la tecnología	Diversidad del hardware para el mismo software	Potencia de cálculo	Memoria y capacidad de almacenamiento
Ordenadores personales.	Continúa.	Alta.	Alta.	Alta.
Consolas.	Se mantiene constante durante años.	Baja.	Se mantiene constante durante años.	Media.
Dispositivos móviles.	Continúa.	Depende de la plataforma, suele ser alta.	Baja, aunque incrementándose con el tiempo.	Baja.

Reflexiona

¿Por qué crees que hay gente prefiere jugar con una consola de videojuegos antes de hacerlo en un ordenador personal?

Autoevaluación

Selecciona una de las ventajas que tiene desarrollar videojuegos para una consola

- ☐ *(link:)*
Se puede ampliar la potencia de la consola fácilmente.
- ☐ *(link:)*
El hardware permanece sin cambios durante la vida de la consola
- ☐ *(link:)*
Permite que varias personas jueguen a la vez en el mismo juego.
- ☐ *(link:)*
Suelen contar con grandes cantidades de memoria RAM y de almacenamiento.

1.3.- Clasificación de los videojuegos.

No existe una clasificación unificada de los géneros de videojuegos, ni siquiera los expertos se ponen de acuerdo. De hecho, lo normal es que un juego moderno sea una mezcla de más de uno. Muchos de estos géneros han surgido como respuesta a un juego innovador que en su momento sentó escuela y cuya idea fue repetida y/o mejorada en desarrollos posteriores. A continuación, vamos a describir algunos de los más importantes, resaltando algunos de los juegos que dieron paso al género o bien supusieron su popularización:



- ✓ **Acción:** Los videojuegos de acción son aquellos donde se realizan movimientos y/o combates rápidos. Aquí podemos incluir algunas de los géneros que veremos más adelante como los juegos de lucha, simulaciones de combate, juegos de disparo en primera persona, etc. Como representantes de los primeros juegos de acción podríamos nombrar **Pong**, **Asteroids** y **Space Invaders**.
- ✓ **Aventura:** Son aquellos que sumergen al jugador en una historia que se va desarrollando conforme avanza en el juego. Suelen incluir puzzles y otros desafíos que se superan con astucia e intuición. Las más antiguas no tenían gráficos y funcionaban mediante descripciones del entorno en forma de texto. Entre ellas destacan **Zork** (en modo texto) y las series: **King's Quest** y **Monkey Island**.
- ✓ **Lucha:** En ellos, el jugador pelea contra otros jugadores o bien contra personajes controlados por el videojuego. Eso incluye los juegos de lucha uno contra uno o los juegos cooperativos donde el jugador o jugadores se enfrentan a hordas de enemigos mientras avanzan por un nivel. Como ejemplos podríamos nombrar: **Double Dragon**, **Street Fighter**, **Mortal Kombat** y **Virtua Fighter**.
- ✓ **Videojuegos de disparo en primera persona (FPS):** Los FPS son juegos donde la pantalla representa lo que vería el personaje a través de sus ojos. En ellos, el personaje suele ir fuertemente armado y dispara a los enemigos mientras recoge ítems desperdigados por el nivel. Los más conocidos fueron: **Wolfenstein 3D**, **Doom**, **Duke Nukem 3D** y **Quake**.
- ✓ **Plataformas:** Este tipo de juegos suele consistir en un personaje, controlado por el jugador, que avanza por un nivel mediante movimientos rápidos y saltos. Los niveles suelen recorrerse en horizontal o vertical. Se podría considerar **Super Mario Bros.** como el iniciador del género al que siguieron otros éxitos como **Bubble Bobble** o **Rainbow Island**.
- ✓ **Estrategia:** Son juegos donde la planificación de las acciones suele tener bastante peso a la hora de obtener una victoria. Podemos clasificar los juegos en dos subgéneros: estrategia en tiempo real y estrategia basada en turnos. La diferencia principal entre ambas es el tiempo de reacción ante los sucesos, ya que el juego no se detiene en el primer caso, obligando a realizar acciones de forma rápida. Uno de los juegos que sentó las bases de la estrategia en tiempo real fue **The Ancient Art of War**, aunque el género fue popularizado por **Dune 2**, **Warcraft** y **Command & Conquer**. En el caso de la estrategia por turnos **Civilization** fue el gran iniciador, al que siguieron otros títulos célebres como **Master of Orion** o **X-Com**.
- ✓ **Puzzle:** Estos juegos permiten al jugador aplicar estrategias y deducciones lógicas con el fin de conseguir ciertos objetivos. Suelen trabajar mucho con formas geométricas y combinaciones de elementos y colores que pueden ser movidos, intercambiados o rotados. También pueden incluir un tiempo limitado o bien introducir una competición con otro jugador para darle más emoción. El caso más claro de este tipo de juegos es **Tetris**.
- ✓ **Simuladores:** Como su nombre indica, intentan recrear un sistema del mundo real. Desde el funcionamiento de una ciudad (**SimCity**), de un avión (**Microsoft Flight Simulator**), de un negocio de transportes (**Transport Tycoon**) o de una nave espacial (**X-Wing**).

1.3.1.- Clasificación de los videojuegos (II).

- ✓ **Juegos de rol (RPG):** Es la versión electrónica de los juegos de rol tradicionales que se juegan con papel y dados. Los jugadores tienen unas estadísticas (fuerza, sabiduría, dinero, etc.) que se van incrementando conforme aumenta su experiencia. Además, pueden adquirir y usar objetos y armas que van recopilando durante el juego. Al contrario que sus equivalentes en papel, aquí no es necesario echar a volar la imaginación para situarse en el juego; además, las reglas están implícitas, lo que evita tener que memorizarlas. Las series **Ultima**, **Baldur's Gate** o **Final Fantasy** son algunos ejemplos de este tipo de videojuegos.



[\(link: PMDM05_CONT_R08_RPG.jpg.\)](#)

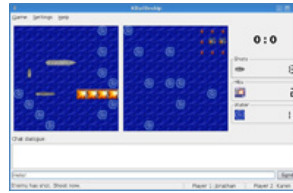
- ✓ **Juegos de rol multijugador masivos (MMORPG):** Son juegos de rol cuyo principal interés es que el desarrollo de la partida se produce en un escenario donde coinciden cientos o miles de otros jugadores. En sus orígenes eran juegos de texto y se llamaban **MUD** (Mazmorras multiusuario). Suelen ser juegos en los que se paga una suscripción mensual para acceder. Algunos ejemplos representativos son: **Ultima Online** y **World of Warcraft**.
- ✓ **Carreras:** El objetivo de estos juegos es llegar el primero a la meta. En muchos de ellos se intenta recrear lo mejor posible las sensaciones de conducir un vehículo en una competición. El primero en convertirse en éxito de ventas fue **Pole Position**.
- ✓ **Deportivos:** Estos videojuegos tratan de transmitir la emoción de practicar un deporte real. Esto incluye desde deportes individuales, a dobles o en equipo. El juego **Summer Games** o las series **PC Fútbol**, **Pro Evolution Soccer** y **FIFA**.
- ✓ **Musicales:** El jugador tiene que seguir o acompañar el ritmo dado por el juego ya sea mediante la pulsación de botones, movimientos o entonación al cantar. El juego clásico es el **Dance Dance Revolution**, donde el jugador tiene que saltar sobre unas flechas que hay en el suelo siguiendo las indicaciones de la pantalla. Otros juegos que popularizaron el género son las series **Singstar**, **Guitar Hero** y **Rock Band**.



[\(link: PMDM05_CONT_R09_DDR.jpg.\)](#)

1.3.2.- Clasificación de los videojuegos (III).

✓ **Minijuegos:** Son muchos juegos en uno, generalmente de muy corta duración y con unas reglas simples. Pueden ser parte de otro tipo de juego, ofreciéndose los minijuegos como una recompensa o como un requisito para seguir avanzando. En otros casos, el juego en sí consiste en la superación de un conjunto de minijuegos (por ejemplo, **Mario Party**).



✓ **Tradicionales:** Los equivalentes electrónicos de los juegos de toda la vida. El juego **Noughts and Crosses (OXO)** puede considerarse uno de los primeros juegos de este género, pero aquí también se incluyen los juegos de cartas, de mesa (ajedrez, damas), etc.

[\(link: PMDM05_CONT_R10_Tradicional.jpg.\)](#)

✓ **Educativos:** Su misión es que el jugador aprenda mientras se divierte. El aprendizaje puede producirse mediante la necesidad de memorización de ciertos datos para conseguir cierto objetivo o bien mediante la observación y repetición de ciertas tareas o comportamientos. Algunas de las series más conocidas son la de **Carmen Sandiego** y la saga **The Incredible Machine**.

✓ **Serios:** Los juegos serios son un género muy reciente. Están diseñados no solo para entretener sino para que al jugar repetidamente se adquieran ideas que entran dentro del ámbito de la ética, la realidad social o el trabajo en equipo.

Reflexiona

¿Tienes claro cuál es tu género favorito? ¿O tu juego ideal tiene una mezcla de varios géneros? Simplemente piensa en cuáles son los juegos que más te gustan y extrae conclusiones.

Autoevaluación

Considera una serie de videojuegos llamado "Legend of Zelda". En la mayoría de sus juegos, Link debe rescatar a la princesa Zelda en el mundo de fantasía llamado Hyrule. Para ello debe combatir con cientos de enemigos, hablar con aldeanos, realizar misiones para conseguir ciertos objetos e ir incrementando sus estadísticas (fuerza, habilidades mágicas, etc.). Todo ellos mientras vamos aprendiendo nuevos detalles de Hyrule y su historia¿A qué géneros crees que pertenece este videojuego?

- ☐ (link:)
Acción.
- ☐ (link:)
Rol (RPG).
- ☐ (link:)
Lucha.
- ☐ (link:)
Aventura.

1.4.- La industria del videojuego.

El hecho de que un usuario final disfrute de un videojuego es el resultado del esfuerzo de muchos elementos que han de trabajar coordinados. Lo primero que podemos preguntarnos es ¿de dónde sale la inversión inicial para pagar el desarrollo? Normalmente procede de un editor o productor que adelanta parte del dinero que espera a recibir por las ventas del videojuego. Dado que es la entidad que paga, el editor suele tener gran poder de decisión en el desarrollo.

El desarrollador (puede ser una empresa o un particular) es el encargado de diseñar e implementar el videojuego. Suele estar especializado en unos determinados géneros y/o plataformas (por ejemplo, juegos de rol multijugador en ordenadores personales). Respecto al desarrollador podemos distinguir tres casos según los proyectos que desarrollen:

- ✓ **Un desarrollador interno** es el que trabaja siempre para el mismo editor o fabricante de hardware.
- ✓ **Un desarrollador de terceros** firma contratos con los editores para llevar a cabo un proyecto concreto en particular. De hecho, la misma empresa desarrolladora puede tener a la vez proyectos con distintos editores.
- ✓ **Un desarrollador independiente** es un equipo pequeño, puede que incluso unipersonal. Suele autopublicar sus propios juegos y, al no estar atado a un editor, tienen plena libertad creativa. Eso permite que surjan ideas innovadoras que nunca recibirían dinero de un editor que las percibiría como una inversión arriesgada. La falta de recursos económicos suelen suplirla con soluciones y productos ingeniosos. A la hora de escribir estas líneas hay un repunte de este tipo de equipos debido a que los costes de distribución que existen en los dispositivos móviles son muy bajos.

La etapa de distribución es la encargada de hacer llegar el videojuego a las estanterías de las tiendas o a las tiendas de descargas digitales. Es aquí donde se genera el beneficio para el desarrollador y/o el editor. Hay que tener en cuenta que el distribuidor se lleva parte de esas ganancias.

[\(link: PMDM05_CONT_R11_Industria.png.\)](#)



[\(link: PMDM05_CONT_R11_Industria.png.\)](#)

1.4.1.- El equipo de desarrollo.

Si bien en los años 70 y 80 los videojuegos eran diseñados e implementados por una persona o un grupo reducido, **hoy en día lo normal es que un proyecto se lleve a cabo por decenas de personas**. Como se puede deducir, en ellos no solamente hay programadores y artistas sino que también intervienen editores, productores, distribuidores, escritores, probadores, publicistas, gestores económicos, soporte técnico, etc.



[./link: PMDM05_CONT_R12_EquipoVideojuego.png./](#)

Tan solo tienes que fijarte en los títulos de créditos de un videojuego moderno, ¡parece tanta gente como en una película!

Observa el siguiente gráfico sobre la estructura de un equipo de desarrollo derivado del propuesto en "Game Development and Production" de Erik Bethke:

Todas esas secciones forman el equipo típico de desarrollo de un videojuego moderno de gran presupuesto.

Cada persona que participa posee un área de especialización, aunque no siempre la aplica trabajando en lo más obvio. Por ejemplo, un programador puede estar echando una mano en gestión de calidad o en soporte técnico ya que sus conocimientos benefician el funcionamiento de esa parte del proyecto.

En el caso de la producción, podemos dividir al personal en otros grupos según la función que desempeñan: diseño general, programación, diseño artístico, audio y gestión. En **este módulo profesional nos centraremos en la parte de producción, y dentro de ella en los equipos de diseño y programación**, especialmente en esta última.

La parte del equipo dedicada al diseño son las personas encargadas de determinar la idea, la mecánica del juego, los diseñadores de niveles o de misiones y los escritores de la historia y los diálogos. Suelen incluir un **diseñador principal** (en inglés, "Lead Designer"), que es quien coordina a los demás.

Por otro lado, y dado que el producto final es software, debe haber programadores que se encarguen de la creación del código específico, del motor de juegos y de todas las demás herramientas relacionadas (editores de niveles, instaladores, etc.). Los equipos de programadores suelen construirse alrededor del **programador principal** (en inglés: "Lead Programmer"), que suele ser el más experimentado. Si el tamaño del equipo es grande, puede aparecer el rol de **director técnico** ("Technical Director"), que tiene un papel menos directo en la programación porque dedica tiempo a gestionar el equipo; si existe, es él quien se comunica directamente con el director de proyecto.

Reflexiona

¿Existe hueco en la industria actual para desarrolladores de pequeño tamaño? ¿Cuáles de los videojuegos a los que has jugado últimamente estaban hechos por desarrolladores independientes?

Caso práctico

Juan continúa su proceso de investigación. Ya tiene una idea de cómo fueron los inicios del mundo de los videojuegos, cuáles son las plataformas más importantes de hoy en día y cuáles son algunos de los géneros más extendidos. También ha tomado contacto con la estructura que tienen los estudios de desarrollo profesionales con el fin de organizar su propio equipo.

Como responsable del proyecto se dispone a dar el siguiente paso formulándose más preguntas: "¿Por dónde empiezo? ¿Cómo se programa un videojuego? ¿Qué estructura tiene?" Esto le será de utilidad para planificar y repartir las tareas que habrá que realizar para poder llevar el proyecto a buen puerto.



Hoy en día en la gran mayoría de los videojuegos presentan tres grandes componentes a grandes rasgos: el **código específico del juego**, el **motor de juegos** y los **recursos**.

El **código específico** es donde se implementa la lógica de ese juego es particular, es decir, el sitio donde se definen las reglas, el comportamiento de los elementos del juego (por ejemplo, la pelota, los enemigos, etc.), cómo se reacciona ante las posibles acciones del jugador, las condiciones para ganar, etc.



[.\(link: PMDM05_CONT_R13_Arquitectura_videojuego.png.\)](#)

El motor de juegos contiene la funcionalidad sobre la que se apoya el código específico para realizar esas actividades. Como veremos más adelante, un buen motor de juegos es independiente del tipo de juego que se está implementando y sólo incluye conceptos genéricos como la captura de eventos del teclado/ratón, la gestión de recursos, la visualización de los objetos, las comunicaciones por red, la reproducción de sonidos, etc.

El motor de juegos nos permite abstraernos del hardware del equipo y el sistema operativo, aunque usan sus servicios para llevar a cabo sus funciones. Esto es muy interesante, veamos la razón con un ejemplo: si nuestro motor de juegos soportara 3 clases de consola y 3 sistemas operativos distintos de ordenadores personales, podemos tener un juego funcionando en 6 plataformas con muy poco esfuerzo adicional.

El tercer componente, los **recursos** (o assets, en inglés), es el compendio de los contenidos del juego. Entre esos contenidos podemos encontrar música, efectos de sonido, modelos 3D, fondos de pantalla, tipos de letra, niveles, etc. El motor de juegos cargará los recursos según le indique el código específico. De la creación de los contenidos suelen encargarse los diseñadores artísticos, ingenieros de sonido, escritores, diseñadores de niveles, etc. aunque su desarrollo puede estar supervisado por programadores.

Entonces, si ya tengo la idea y quiero programar un juego, ¿por dónde empiezo? ¿Qué componente es el primero sobre el que debo trabajar? Pues dado que la elección del motor de juegos determinará cómo será el código específico y el formato de los recursos lo lógico es comenzar seleccionando un motor de juegos. Es una decisión que no debe tomarse a la ligera ya que cambiar el motor de juego a mitad de un proyecto puede tomar mucho tiempo.

En los siguientes apartados haremos una revisión de los distintos componentes que tiene un motor de juego típico y veremos cuáles son sus características más importantes.

Autoevaluación

¿Cuál de los siguientes elementos NO forma parte de un componente de un videojuego?

- ☐ *(link:)*
El comportamiento de los enemigos.
- ☐ *(link:)*
Los efectos de sonido.
- ☐ *(link:)*
Las comunicaciones por red.
- ☐ *(link:)*
Las reglas del juego.
- ☐ *(link:)*
El sistema operativo.

2.1.- Clasificación de motores de juegos.

Un motor de juegos expone una interfaz de programación de aplicaciones (**API**) que permite al código específico utilizar su funcionalidad. Sin embargo, no todos los motores ofrecen las mismas características.

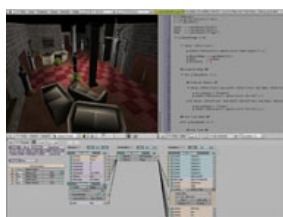
Según el nivel de abstracción cubierto por el motor de juegos, podemos distinguir 3 tipos de complejidad creciente:

- ✓ Motores diseñados únicamente para facilitar la representación en pantalla y el acceso al hardware en general (audio, dispositivos de entrada, red, etc). De hecho, muchos de ellos son considerados como librerías más que como motores de juegos. Algunos ejemplos: **SDL**, **LWJGL** o **Allegro**.
- ✓ Motores que, además de lo anterior, añaden un soporte total o parcial a la lógica del juego. Podemos destacar: **JGame** y **Ogre3D**.
- ✓ Motores que, además de todo lo anterior, incluyen plataformas para la creación, modificación o integración de contenidos. Algunos de ellos: **Blender Game Engine**, **jMonkeyEngine 3** o **XNA Game Studio**.

Otro aspecto a considerar es el del coste económico y de la licencia de uso. Para proyectos personales o no comerciales es posible encontrar motores que no nos supongan ninguna inversión, lo que puede ser determinante. La licencia de uso de otros motores pueden impedirnos vender el resultado final o limitar el número de unidades vendidas antes de tener que pagar.

Teniendo todo esto en cuenta, **podemos clasificar los motores en dos grandes grupos según el tipo de licencia de uso** :

- ✓ **De código abierto**: No es necesario pagar por su uso e incluyen el código fuente del motor. Hay que prestar atención al tipo de licencia concreto, pues algunas restringen el uso del motor a la creación de videojuegos que también sea software libre (por ejemplo, la licencia **GPL**). La documentación y el soporte suele ser llevado a cabo por la comunidad de usuarios. Algunos de los más conocidos son **Ogre3D**, **Irrlicht**, **id Tech 4** (publicado como código abierto en noviembre de 2011), **jMonkeyEngine 3** y **Crystal Space 3D**.
- ✓ **Propietario**: Son desarrollados por empresas que cobran por la comercialización de juegos basados en sus motores. En algunos casos pueden ofrecerse de forma gratuita si se cumplen determinadas condiciones. El coste final puede depender de muchos factores: número de empleados, de copias vendidas o del dinero entregado por los editores para su realización. Suelen estar excelentemente documentados y, por lo general, incorporan herramientas adicionales que facilitan la gestión de los contenidos. Las empresas ofrecen normalmente asistencia técnica de sus productos. Por todo ello, las grandes producciones suelen basarse en este tipo de motores de juego. A la hora de escribir estas líneas, los más reconocidos son **id Tech 5**, **C4 Engine**, **Gamestudio**, **Source Engine**, **RAGE**, **SCUMM** (específico para aventuras gráficas), **Unreal Engine**, **IW** y **Torque Game Engine**.



[.\(link: PMDM05_CONT_R14_01_BlenderEngine.jpg.\)](#)

2.1.1.- Programación de un motor. APIs básicas.

Cada motor soporta solamente determinados lenguajes de programación para el código específico del juego, así que debemos elegirlo con cuidado. Es importante destacar que algunos motores usan un lenguaje propio para especificar el comportamiento del juego. El cómo se integra el código específico del juego con el motor puede variar considerablemente de un motor a otro, ya que no todos los motores comparten la misma estructura interna ni filosofía de programación. La mejor forma de empezar es estudiar los tutoriales que suelen incluirse con el motor.



Un motor de juegos está especializado solamente en determinados aspectos, no todos incluyen las mismas características. De hecho, es posible combinar distintos motores de manera que cada uno aporte parte de la funcionalidad necesaria para completar los requisitos del juego. En esos casos, el código específico se comunicará con ambos motores para integrarlos.

Muchos motores de juegos se apoyan en librerías ya existentes que les proporcionan parte de su funcionalidad básica. Por ejemplo, **OpenGL** o **DirectGraphics** suelen utilizarse para mostrar objetos en 2 y 3 dimensiones, siendo estas librerías las que acceden directamente a los controladores de la tarjeta gráfica. Esto permite al motor no estar atado a un determinado hardware. Más aún, algunas de esas librerías están disponibles para múltiples plataformas (por ejemplo **OpenGL**) lo que facilita la portabilidad del motor a múltiples sistemas.

Las librerías no tienen por qué ser exclusivamente para acceder al apartado gráfico: existen muchas otras librerías para otros usos. Por ejemplo, **OpenAL** o **FMOD** hace lo mismo con el audio y **OpenCL** con las operaciones de computación intensiva.

Para saber más

En la versión inglesa de la Wikipedia hay una comparativa de motores de juegos de gratuitos y de código abierto:

Comparativa de motores de juegos gratuitos y de software libre (en inglés). (*link:*
http://en.wikipedia.org/wiki/List_of_game_engines#Free_and_open_source)

Puedes encontrar una exhaustiva lista de motores de juegos en el siguiente enlace:

Base de datos de motores de juegos (en inglés). (*link:* *<http://devmaster.net/devdb/engines>*)

2.2.- Ventajas de la utilización de motores.

¿Qué pasaría si no usáramos un motor de juegos? De hecho, los primeros videojuegos no los utilizaban. Cuando esto ocurre, el código específico es el encargado de todas las tareas: comunicarse directamente con el sistema operativo y el hardware, dibujar en la pantalla, interceptar los dispositivos de entrada, conectarse con otras instancias a través de la red en el caso multijugador, implementar la "inteligencia" de los enemigos, simular la gravedad, detectar cuando dos objetos colisionan, etc.

A priori, puede parecer una idea buena porque sólo se implementa aquello que se va a necesitar, con lo que el resultado será más óptimo. Además, el código estará más integrado.

Imaginemos que realizamos el proyecto así. **¿Qué problemas podríamos encontrarnos?** Analicemos algunos escenarios posibles:



Ventajas del uso de un motor de juegos

Escenarios	Sin un motor de juegos	Con un motor de juegos
El juego tiene tanto éxito que nos encargan una continuación.	Tendremos que revisar todo el código del proyecto anterior y diferenciando lo que nos sigue sirviendo de lo que no. El motivo es que no hay una separación clara entre lo genérico y lo específico del juego.	El código del proyecto ya está separado en código específico del juego y el motor (código genérico).
Hay que migrar el juego a una nueva plataforma.	<p>No nos queda más remedio que revisar todo el código del proyecto anterior y analizar qué parte es dependiente de la plataforma antigua para sustituirlo por código nuevo. Por supuesto, tendríamos que tener cuidado de no estropear nada.</p> <p>Es posible que los recursos ya no sean válidos para la nueva plataforma y haya que adaptarlos.</p> <p>Además, si detectamos un error en el juego o mejoramos alguna herramienta habrá que cambiar el código por separado en ambos proyectos.</p>	En el momento que el motor de juegos soporte la nueva plataforma bastará con recompilar el proyecto.
Llega un nuevo programador a la empresa y queremos formarlo para trabajar en nuestros proyectos.	Habrà que explicarle la estructura del código, que será distinta para cada proyecto en el que vaya a trabajar y plataforma soportada.	Bastará con enseñarle a usar el motor de juego, que será el mismo en muchos proyectos. Luego se le señalarán las particularidades de cada proyecto concreto en el que vaya a trabajar.
Queremos incorporar a nuestro juego el efecto gráfico X de un juego de la competencia.	Será necesario investigar cómo funciona la característica X, implementarla en todas las plataformas del proyecto y hacer las pruebas correspondientes antes de usarla.	Seguramente haya un motor de juego que implemente la característica X de serie en todas las plataformas, con lo únicamente tendremos que usarla.

Autoevaluación

Indica cuáles de los siguientes elementos son ventajas que obtenemos al usar un motor de juegos multiplataforma:

☐ (link:)
Los mismos recursos sirven para distintas plataformas.

☐ (link:)
El mismo código específico sirve para distintas plataformas.



(link:)

El código estará más integrado.



(link:)

Lo aprendido del motor en un proyecto no puede reutilizarse en otros proyectos con el mismo motor.

2.3.- Componentes de un motor de juegos.

Como hemos visto antes, no todos los motores de juegos tienen las mismas funcionalidades, aunque sí existen algunos aspectos que son comunes entre muchos de ellos. Vamos a ver cuáles son los componentes más habituales que componen un motor de juegos:

- ✓ Motor gráfico 2D
- ✓ Motor gráfico o de renderizado 3D
- ✓ Detector de colisiones
- ✓ Motor de físicas
- ✓ Motor de inteligencia artificial (IA)
- ✓ Motor de sonidos
- ✓ Gestor de comunicaciones en red



En los siguientes apartados desarrollaremos en qué consiste cada uno de estos componentes y lo que aporta a un videojuego.

2.3.1.- Motor gráfico 2D.

La palabra "vídeo" proviene del latín y significa "yo veo". Es por ello que los videojuegos tienen una componente visual tan importante y que la sucesión de imágenes es la forma principal de comunicación con el jugador. El número de imágenes por segundo que se crean es un número llamado **framerate** y se mide en cuadros por segundo (FPS). Para que el ojo no perciba parpadeo, el framerate debe ser superior a 46 FPS.



Antes de detallar las características de los motores gráficos debemos introducir el concepto de 2D y 3D.

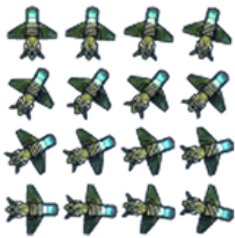
[\(link: PMDM05_CONT_R15_01_2D.JPG.\)](#)

Los juegos 2D son aquellos en los que los dibujos tienen dos dimensiones, por ejemplo ancho y alto, pero no profundidad. Para entendernos, son formas planas que se mueven por la pantalla. Podemos dibujarlas más grandes para parecer que están más cerca y más pequeños para parecer que se alejan, pero siguen siendo planos. Un motor 2D permite dibujar de forma sencilla figuras geométricas (líneas, rectángulos, círculos, etc.) con lo que se evita tener que implementar los algoritmos en el código específico.

Normalmente, un juego 2D consiste en una o varias capas de imágenes en el fondo sobre las que se dibujan **objetos** 2D como pueden ser el jugador, los enemigos, monedas, obstáculos, disparos, etc. A esos objetos que son representativos en el juego se les denomina **sprites**. El motor controla todos los que estén activos en un momento dado, almacenando la posición en la que están, si hay que dibujarlos rotados, si están o no animados, etc.

Las **animaciones** en los juegos 2D se producen cambiando el dibujo del sprite rápidamente de manera que haya pequeñas diferencias entre uno y otro, lo que produce en el cerebro una sensación de continuidad. Observa las siguientes imágenes y verás como cada dibujo está un poco más girado que el anterior: si se dibujara uno tras de otro (de izquierda a derecha y de arriba a abajo) en la misma posición el efecto que captarías es que estaría girando en el sentido de las agujas del reloj.

Los motores gráficos 2D dan soporte a los sprites, ofreciendo herramientas para escalar, rotar y mover objetos por la pantalla con comodidad. En la unidad de trabajo siguiente se verán estos conceptos con más detenimiento. También la información que se superpone en la pantalla (como el número de vidas o la puntuación).



Cuando dos sprites se superponen el resultado final dependerá del orden en que se dibujen. Dicho orden normalmente se denomina **plano o capa**. Si el motor lo soporta, hay que elegir el plano de cada sprite cuidadosamente para que se produzca el efecto que deseamos.

[\(link: PMDM05_CONT_R15_02_2DAnimacion.png.\)](#)

En algunos motores gráficos podemos representar mundos que sean varias veces más grandes que el tamaño de la pantalla. El motor se encargará de mostrar solamente aquella parte que le indique el código específico. Esto permite, por ejemplo, seguir al jugador a lo largo y ancho de un nivel.

Otro elemento gráfico que poseen muchos motores 2D es el de **tilemap** (celosía), que es una composición de imágenes pequeñas del mismo tamaño. Este recurso es muy útil para dibujar fondos, mapas y niveles.

Reflexiona

¿Todos los juegos tienen que ser en 3D para ser divertidos? ¿No hay mercado hoy en día para las 2D ?

Autoevaluación

Relaciona los conceptos entre sí.

Ejercicio de relacionar

Concepto	Relación	Concepto
FPS.	<input type="checkbox"/>	1. Superposición de sprites.

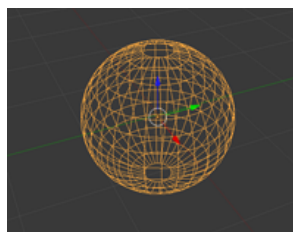
Concepto	Relación	Concepto
Capa.	<input type="checkbox"/>	2. Dibujo de un mapa.
Tilemap.	<input type="checkbox"/>	3. Alto y ancho.
2D.	<input type="checkbox"/>	4. Imágenes por segundo.

2.3.2.- Motor gráfico o de renderizado 3D.

Por otro lado, en los juegos 3D los objetos tienen tres dimensiones y, por tanto, pueden moverse en el espacio con total libertad. Esto intenta imitar la visión del ser humano, que también es tridimensional. El problema es que la gran mayoría de los monitores sólo son capaces de representar imágenes 2D. Para solucionarlo podemos realizar una operación llamada **proyección**, que transforma esa imagen 3D en una 2D. Ese proceso de generar las imágenes recibe el nombre de **renderizado** y es tarea del motor gráfico.

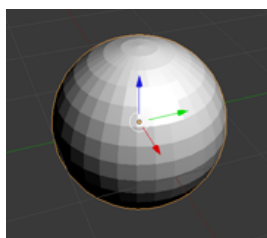
[\(link: PMDM05_CONT_R16_3DEsfera.png.\)](#)

Normalmente, los objetos 3D están constituidos por **vértices** que juntos forman **polígonos**, generalmente de tres o cuatro lados.



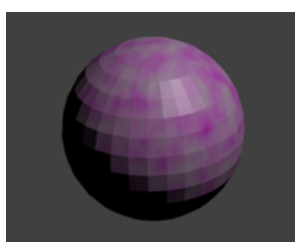
[\(link: PMDM05_CONT_R16_3DEsfera.png.\)](#)

En la figura se observa una esfera 3D. Los vértices son los puntos donde confluyen las líneas. La flecha roja indica el eje X, la verde el eje Y y la azul el eje Z.



[\(link: PMDM05_CONT_R17_3DEsfera2.png.\)](#)

Los polígonos formados por los vértices se observan mejor en esta otra figura. Fíjate que algunos se ven en un color más claro que otros. Todo ello es debido a la iluminación de la escena.



[\(link: PMDM05_CONT_R18_3DEsfera3.png.\)](#)

Sobre los polígonos se pueden aplicar **texturas** para darle un aspecto más realista. Simplificando mucho, podemos considerarlas como unas pegatinas que se pegan a su superficie. Las veremos con más detalle en la unidad de trabajo de motores 3D.

Un motor gráfico 3D simplificará el trabajo con los objetos tridimensionales permitiendo al código específico realizar operaciones con ellos independientemente del hardware sobre el que se esté ejecutando el juego. Entre estas operaciones está la de crear objetos, moverlos, escalarlos, rotarlos, deformarlos, animarlos, cambiarles las texturas, etc. Además, deberá soportar la creación de luces que iluminen dichos objetos y cámaras que permitan observar el mundo renderizado desde la perspectiva que necesitemos en cada momento.

El conjunto de todos los objetos de este mundo virtual junto con las cámaras, las luces y demás elementos forman la escena. Internamente, la escena se suele almacenar mediante el denominado **grafo de escena** que se estudiará más adelante.



[.\(link: PMDM05_CONT_R19_3D.jpg.\).](#)

Autoevaluación

Relaciona los conceptos entre sí.

Ejercicio de relacionar

Concepto	Relación	Concepto
3D.	<input type="checkbox"/>	1. Pasar de 3D a 2D.
Vértice.	<input type="checkbox"/>	2. Aspecto del polígono.
Proyección.	<input type="checkbox"/>	3. X, Y, Z.
Textura.	<input type="checkbox"/>	4. Ayuda a definir un polígono.

2.3.3.- Detector de colisiones.

Tanto en el motor gráfico 2D como en el 3D tenemos objetos que se mueven por la pantalla. Una de las acciones que realizaremos más frecuentemente es comprobar si dos objetos colisionan entre sí. Por ejemplo: nos interesará saber si una bala ha chocado contra un enemigo o si nuestro coche ha llegado a la meta. Ahí es donde entran los detectores de colisiones.



¿Es realmente necesario que se encargue un motor? ¿No podemos comprobarlo nosotros mismo desde el código específico de una forma sencilla? Resulta que detectar

colisiones no es nada fácil, especialmente en ciertas circunstancias. En el caso 2D, dos objetos no han colisionado hasta que tienen un punto (**píxel**) en común. Eso implica tener que comprobar punto a punto de las dos imágenes buscando que ambas coincidan en algún sitio. Si tenemos en cuenta que puede haber decenas de objetos en pantalla de forma simultánea y que el objeto puede ser de gran tamaño, las cosas se complican. **Podríamos simplificar el proceso de detección** de una colisión suponiendo que hay un círculo o una caja que envuelve cada objeto y comprobar si dichas figuras geométricas se superponen: matemáticamente es sencillo de comprobar. **El problema es que pueden darse falsos positivos** de colisión, veamos un ejemplo.

En la parte izquierda de la imagen puedes comprobar como hay un falso positivo: las cajas de la izquierda están superpuestas pero los objetos en realidad no se tocan. En la parte derecha sí lo hacen. Si el motor hace bien su trabajo, cuando las figuras geométricas colisionen realizará un **estudio más cuidadoso** en la zona superpuesta para saber si efectivamente han chocado; **de esta forma podemos solucionar los falsos positivos**.



[.\(link: PMDM05_CONT_R20_EjemploColisiones.jpg.\)](#)

[.\(link: PMDM05_CONT_R21_01_2DColision.png.\)](#)

Algunos detectores de colisiones clasifican los objetos en tipos de manera que sólo comprueban ciertas combinaciones. Por ejemplo: las colisiones de los objetos tipo "bala" con los tipo "enemigo" sí se comprueban, pero las colisiones de los objetos tipo "enemigo" con otros tipo "enemigo" no. **Esto permite ahorrar mucho tiempo de cálculo.**

Si el caso de los objetos 2D no es sencillo de solucionar, con los objetos 3D será más difícil aún porque los objetos tienen volumen y no todo está en el mismo plano.

Para saber más

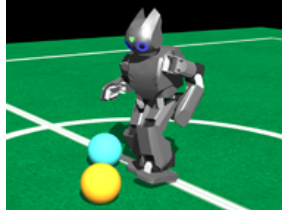
Si quieres aprender cómo se realizaría mediante programación la comprobación de colisión usando figuras geométricas simples puedes echar un vistazo a esta presentación:

Resumen textual alternativo [\(link: PMDM05_Descripcion_presentacion_ejemplo_comision.html \)](#)

2.3.4.- Motor de físicas.

Muchos de los videojuegos intentan dotar a los objetos que aparecen de un comportamiento creíble. Esto es, si empuja un objeto hacia el borde de una mesa, el jugador espera que el objeto caiga al suelo. Si se golpea un cristal, esperamos que se haga añicos. O si lanzo golpeo una pelota, espero que siga una trayectoria natural y al llegar a la pared rebote de una forma natural.

Todos **estos comportamientos son calculados por el motor de físicas** y pueden ir desde dotar de un efecto de gravedad al escenario donde estamos jugando hasta el cálculo de la dirección que tomarán dos objetos al chocar, calcular por donde discurre un líquido derramado, por donde se romperá un objeto al golpearlo o que al tirar de una cuerda se mueva un objeto que está atado al otro extremo. De hecho, en **entornos 3D suele ser el motor de físicas el encargado de detectar las colisiones entre objetos**. Para poder realizar todas estas actividades el motor asignará una serie de propiedades físicas a cada objeto (velocidad, aceleración, masa, etc.) y aplicará las ecuaciones de la **física newtoniana**.



Para saber más

Puedes aprender más sobre la mecánica newtoniana que es utilizada en la mayoría de los motores de física visitando este enlace de la Wikipedia:

Mecánica newtoniana. ([link: http://es.wikipedia.org/wiki/Mec%C3%A1nica_newtoniana](http://es.wikipedia.org/wiki/Mec%C3%A1nica_newtoniana))

Los cálculos de este tipo de motores requieren un uso intensivo del procesador . Es por ello que hace unos años surgieron tarjetas aceleradoras de física, que permitían dejar libre la **CPU** para otras tareas. Su funcionamiento era similar a las **GPU** de las tarjetas gráficas 3D pero su procesador (**PPU**) estaba dedicado solamente a realizar cálculos de simulaciones físicas. Poco después se trasladó esa funcionalidad a las propias GPU de las tarjetas gráficas, ya que éstas son capaces de realizar millones de cálculos por segundo en paralelo.

Por tanto, es importante conocer que algunos motores de físicas son capaces de descargar la CPU y realizar los cálculos en la **GPU** de la tarjeta gráfica o en la PPU de la tarjeta aceleradora de física.

Para saber más

JBullet es el motor de físicas que está integrado en JMonkeyEngine 3. Dado que será el motor que usaremos posteriormente en el módulo, puedes echar un vistazo a una demostración de sus posibilidades en el siguiente enlace. Usa el ratón y las teclas del cursor para moverte por las distintas demostraciones.

Demostración JBullet. ([link: http://jbullet.advel.cz/applet](http://jbullet.advel.cz/applet))

Autoevaluación

Indica cuáles de los siguientes problemas podrían ser resueltos con un motor de físicas:

- ☐ ([link:](#))
Dibujar un objeto con el color de la luz que incide sobre él.
- ☐ ([link:](#))
Detectar que dos objetos han colisionado.
- ☐ ([link:](#))
Producir un sonido cuando un objeto cae al suelo.
- ☐ ([link:](#))

Hacer que un objeto frene por el rozamiento con el suelo.

☐ *(link:)*

Saber qué dirección tomará un objeto cuando choca contra otro.

2.3.5.- Motor de inteligencia artificial (IA).

Los jugadores esperan que los enemigos que aparezcan en el videojuego le supongan un desafío. De lo contrario, rápidamente se perdería el interés por jugar. Es ahí donde entra el motor de inteligencia artificial (IA) cuya misión es la de proveer a algunos de los elementos del juego de un comportamiento que parezca ser inteligente. Dada la complejidad de algunos de los problemas que soluciona el motor de IA y teniendo en cuenta que **la toma de decisiones tiene que hacerse en muy poco tiempo** , la mayoría de las veces se opta por usar técnicas simples y rápidas.

7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1		1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9		11	12	13	14
6	5	4	5	6	7	8	9	10		10	11	12	13

Por ejemplo, para encontrar de forma eficiente el trayecto que tendría que seguir un guardia de seguridad para llegar a un punto en un mapa se usan algoritmos de búsqueda de caminos. El más conocido, es el A*.

En la figura, el **algoritmo A*** ha encontrado el camino óptimo (en rojo) para llegar al cuadro azul desde el cuadro verde, teniendo en cuenta que los cuadrados grises son obstáculos.

Un recurso muy utilizado para simular comportamientos sería el uso de **máquinas de estados finitos**. Su aplicación permite que un objeto se comporte de forma distinta según la situación. Por ejemplo: un guardia puede estar vigilando una zona y, en caso de detectar al jugador, dejaría inmediatamente esa tarea para proceder a perseguirlo. Si lo perdiera de vista, volvería a su tarea original. Al personaje del juego que no está bajo control directo del ordenador se le denomina **NPC** (personaje no jugador) y suele ser controlados por la IA.

En algunos juegos el comportamiento inteligente se produce mediante la ejecución de código específico condicionado a ciertos eventos. No es extraño que ese código esté en **lenguaje de guiones** (scripting) formando parte de los recursos; así no será necesario recompilar todo el proyecto para hacer cambios en el comportamiento del juego.

Cualquier técnica en el campo de la inteligencia artificial podría aplicarse aquí: **lógica difusa, redes neuronales, redes bayesianas, algoritmos genéticos**, etc. Eso sí, hay que considerar que no deben ser muy costosas computacionalmente hablando para que el juego no se ralentice.

Reflexiona

¿Por qué hay tanta variedad de técnicas de inteligencia artificial? ¿Crees que existe una que sirva para todo?

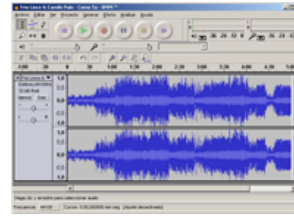
Autoevaluación

Indica cuáles de los siguientes problemas podrían ser resueltos con un motor de IA:

- ☐ *(link:)*
Sustituir al contrincante en un juego de dobles cuando solamente haya un jugador humano.
- ☐ *(link:)*
Determinar cuándo se tiene que reproducir el sonido de una explosión.
- ☐ *(link:)*
Buscar el camino óptimo para llegar a un punto del mapa.
- ☐ *(link:)*
Elegir el momento adecuado para disparar una bala dirigida al jugador.
- ☐ *(link:)*
Permitir que los personajes no jugadores (NPC) tomen decisiones propias.

2.3.6.- Motor de sonidos.

Es el encargado de controlar la reproducción de música y efectos de sonido de manera sincronizada con el resto del juego. El código específico puede disponer de esta funcionalidad de forma independiente al hardware de la plataforma. Esto quiere decir que si, por ejemplo, el hardware no soportara la reproducción de más de un sonido a la vez, el motor podría realizar la mezcla él mismo para producir el mismo efecto sin tener que cambiar el código específico.



[\(link: PMDM05_CONT_R24_AudioEditor.png.\)](#)

Tanto la música como los efectos de sonido son parte de los recursos del juego. Ambos pueden incluirse como partitura

(usando archivos) o bien de forma digitalizada, ya sea grabada directamente del mundo real o modificada con algún editor de audio como el de la figura. En los juegos modernos también **pueden incluirse voces** que se reproducirán de forma sincronizada con el motor de gráficos con el fin de dar la sensación de que el modelo está moviendo los labios.

Los motores de sonido pueden ser muy complejos: algunos tienen conexiones con el motor de físicas para cambiar las características del sonido según a situación. Por ejemplo, podría producir un **efecto doppler** (el que hace que sepamos si una ambulancia está acercándose o alejándose simplemente por el sonido que emite). También, otros pueden distorsionar el sonido para **simular la acústica** de una sala o de un corredor, dotando de más realismo al juego.

Por último, algunos motores permiten generar sonido posicional en 3D. Eso quiere decir que pueden localizar espacialmente el sonido en aquellos sistemas que soporten audio multicanal (por ejemplo, 5.1) o bien simularlo en la salida de auriculares. Si el motor de sonidos está integrado con el motor gráfico 3D, una forma de conseguirlo es asociando el sonido a uno de los objetos de la escena; el resultado se obtiene automáticamente.

Autoevaluación

Indica con cuáles de los siguientes motores puede interactuar el motor de sonidos:

- ☐ *(link:)*
Con el motor gráfico para generar audio posicional.
- ☐ *(link:)*
Con el motor de IA para reproducir voces grabadas moviendo los labios adecuadamente.
- ☐ *(link:)*
Con el motor de físicas para modificar las características del sonido según el entorno.
- ☐ *(link:)*
Con el código específico, para iniciar un sonido cuando el jugador haga una acción.
- ☐ *(link:)*
Con el detector de colisiones, para que cuando se reproduzca un sonido se informe de un choque.

2.3.7.- Gestor de conexiones en red.

El último componente de los motores que vamos a estudiar es el gestor de conexiones en red. **Este gestor nos permite que nuestro juego no esté aislado y pueda comunicarse con otros sistemas.** El uso más obvio es la creación de juegos multijugador, pero no es el único.



Otros escenarios que pueden resolverse con este componente son:

- ✓ Subida de la puntuación de la partida a un servidor Web o a una red social.
- ✓ Descarga de actualizaciones del juego.
- ✓ Implementación de salas para poder seleccionar contrincantes.
- ✓ Comunicaciones de voz y/o de texto entre varios jugadores.

[\(link: PMDM05_CONT_R25_01_NetMultijugador.jpg.\)](#)

Todo ello, como siempre, sin que el código específico tenga que saber nada de la infraestructura de red, el sistema operativo o la plataforma en general.

Muchos de los gestores de conexiones en red facilitan la creación de juegos multijugador permitiendo que el estado y las propiedades de los objetos (por ejemplo, su posición, su nivel de salud, etc.) se sincronice con todos las demás instancias del juego a través de un servidor. En estos casos es posible que el código específico no necesite apenas cambios para adaptarse al entorno multijugador.

La forma de implementar el multijugador varía de un motor a otro. Algunos usan la arquitectura cliente/servidor **arquitectura cliente/servidor**, de manera que todos las instancias de juego se conectan a un mismo servidor, que recibe los datos de cada cliente y los reenvía a los demás. Otros usan una **arquitectura distribuida** o **P2P** donde todos se comunican directamente con todos, aunque sólo suele usarse en redes de área local donde eso no supone una penalización de rendimiento.

Para saber más

Puedes ver un ejemplo de cómo se realiza la comunicación en las dos arquitecturas en la siguiente presentación:

Resumen textual alternativo ([link: PMDM05_Descripcion_presentacion_dos_arquitecturas.html](#))

La separación entre los roles de cliente y servidor es la más utilizada. De hecho, algunos juegos como Quake 2 o Quake 3 , siguen este modelo incluso para el modo de un jugador, simplemente la máquina se conecta consigo misma. El servidor guarda en memoria el estado de la partida (objetos activos, vida restante de cada jugador, ítems , munición restante, etc.) y los clientes sirven solamente para enviar los movimientos o acciones (saltar, disparar, etc) y para mostrar por pantalla la proyección de lo indicado por el servidor. El haberlo diseñado así permite extender soportar sin apenas cambios los modos multijugador.

Autoevaluación

Indica cuáles de los siguientes problemas podrían ser resueltos con un gestor de conexiones de red:

- ☐ ([link:](#))
Permitir que varias personas jueguen en el mismo juego sin estar en el mismo ordenador.
- ☐ ([link:](#))
Buscar el camino óptimo para llegar a un punto del mapa teniendo en cuenta la posición de cada jugador.
- ☐ ([link:](#))
Posicionar los sonidos en función de la situación de los otros jugadores.
- ☐ ([link:](#))
Dibujar a los personajes que están conectados por la red.
- ☐ ([link:](#))

Permitir que los jugadores conversen entre sí, ya sea mediante texto o audio.

2.4.- Librerías que dan soporte a los motores.

Internamente los motores también se apoyan en otros componentes. Por ejemplo: el sistema operativo, librerías de representación gráfica o librerías de reproducción de sonidos. Algunas de esas librerías son multiplataforma mientras que otras sólo funcionan en ciertos sistemas operativos o un hardware en concreto.

Veamos algunas de las librerías más utilizadas respecto al aspecto gráfico:

- ✓ **DirectGraphics (DirectDraw y Direct3D)**: Son librerías de Microsoft que se encuentran disponibles en sus productos, como el sistema operativo Microsoft Windows y en su consola Xbox360. DirectDraw soporta funciones para dibujar en 2D, mientras que Direct3D se usa para dibujar escenas 3D a partir de primitivas sencillas. En otros sistemas operativos puede ser emuladas usando Wine, un entorno que permite ejecutar programas Windows. Las dos librerías forman parte de la colección de APIs DirectX. La versión 11 de DirectX está incorporada de serie en Windows 7 y Windows Server 2008R2, aunque puede también instalarse en Windows Vista y Windows Server 2008.
- ✓ **OpenGL**: Es la librería gráfica 2D y 3D más utilizada. Se soporta en múltiples sistemas operativos y plataformas. Está gestionada por el consorcio sin ánimo de lucro Khronos Group y está en constante revisión. La última versión a la hora de escribir estas líneas es la 4.2. Una de las grandes ventajas de OpenGL es el soporte de extensiones que permite ampliar la funcionalidad de la librería o bien implementar funciones opcionales.
- ✓ **OpenGL ES**: Es una versión reducida de OpenGL dirigida a dispositivos empujados o con poca potencia. Suele ser la librería utilizada en las plataformas móviles modernas como los smartphones. Una aplicación que haga uso de OpenGL ES puede funcionar casi sin cambios en su versión correspondiente de OpenGL estándar pero lo contrario no suele ser cierto.

Respecto al audio:

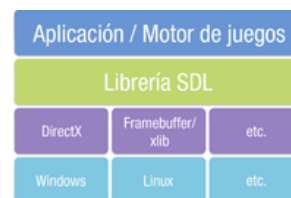
- ✓ **DirectSound y DirectSound 3D**: También forman parte de DirectX. La primera de ellas, especializada en audio 2D fue dividida posteriormente en dos librerías (XAudio2 y XACT3). Presentan el mismo problema que DirectGraphics ya que sólo funcionan en software o hardware de Microsoft.
- ✓ **OpenAL**: Es la equivalente de OpenGL en audio, es decir, tiene soporte multiplataforma, integrándose muy bien con la librería gráfica. Soporta efectos especiales avanzados que pueden ser acelerados por el hardware si éste lo soporta. La versión inicial fue programada por Loki Software para facilitarse el trabajo de adaptar juegos de Windows a Linux.

Otras librerías interesantes:

- ✓ **OpenCL**: Es una librería estándar de programación paralela. Puede usarse para motores de físicas o para motores de IA. Es gestionada por la misma gente de OpenGL y soporta aceleración hardware allí donde esté disponible.

Y por último, librerías que ofrecen funcionalidad combinada:

- ✓ **SDL (Simple DirectMedia Layer)**: Ofrece una interfaz de programación multiplataforma muy sencilla para acceder a los dispositivos gráficos (2D) y de sonido. Además, permite gestionar los dispositivos de entrada, como joysticks, gamepads, teclados, pantallas táctiles y ratones. Suele combinarse con OpenGL si se desea funcionalidad 3D. Si no se necesitan otros motores específicos, SDL puede ser la base sobre la que construir un videojuego.



[\(link: PMDM05 CONT R26 SDL.png.\)](#)

- ✓ **Allegro**: Similar a SDL, aunque incorpora soporte 3D (sin aceleración, de forma limitada).

3.- Estudio de juegos existentes.

Caso práctico

Por fin ha terminado el trabajo de investigación y documentación de **Juan**. Ya tiene una idea bastante exacta de qué se necesita para programar un videojuego. Antes de ponerse manos a la obra con su equipo, decide echar un vistazo a videojuegos de código abierto para tomar nota de algunas decisiones tomadas durante su desarrollo. Dado que el código fuente y la documentación interna de muchos de esos juegos está disponible en Internet, no le supondrá mucho problema acceder a esa información.



Para acabar la unidad vamos a estudiar un juego existente. Concretamente un juego GPL llamado **SuperTuxKart**. Puedes descargarlo o buscar información sobre él en la [página del proyecto](http://supertuxkart.sourceforge.net/) (link: <http://supertuxkart.sourceforge.net/>).

Tras jugar un rato a él y navegar por la página web (en inglés) hemos extraído información sobre los siguientes aspectos. Intenta localizarlos por tu cuenta, te vendrá bien para realizar la tarea de la unidad de trabajo.



[\(link: PMDM05_CONT_R28_SuperTuxKart.jpg.\)](#)

✓ Género:

- ✓ **Carreras:** El juego consiste en una competición de karts.
- ✓ **Acción:** Hay que tomar decisiones rápidas, como frenar con anticipación, usar armamento que cogemos en cajas, etc.
- ✓ **Lucha:** Podemos usar las armas para atacar al resto de jugadores y enlentecer su avance.

✓ Plataformas:

- ✓ **Ordenadores:** Windows, Linux, FreeBSD, NetBSD, Solaris y MacOS X.
- ✓ **Consolas:** Ninguna.
- ✓ **Dispositivos móviles:** Ninguno.

✓ Librerías/motores del juego:

- ✓ **Motor de gráficos:** Irrlicht.
- ✓ **Librería 3D:** OpenGL.
- ✓ **Librería de audio:** OpenAL.

✓ Soporte multijugador

- ✓ Sí, hasta 4 jugadores en el mismo ordenador. No soporta juego en red.








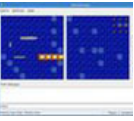








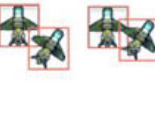

✓ Inteligencia artificial





- ✓ Sí, para los participantes no humanos que compiten con el jugador.

La mejor forma de iniciarse en el mundo del desarrollo de videojuegos es jugando e investigando. Estudiar algunos casos concretos en detalle puede darnos mucha información. Busca juegos que sean software libre y que usen el motor de juegos que vayas a utilizar para analizar su código fuente. Nosotros, te adelantamos, utilizaremos JGame para la parte 2D y JMonkeyEngine 3.0 para la parte 3D.

Anexo.- Licencias de recursos

Licencias de recursos utilizados en la Unidad de Traba

Recurso (1)	Datos del recurso (1)	Recurso (1)
	<p>Autoría: Monsoleiil.</p> <p>Licencia: CC-BY-SA.</p> <p>Procedencia: http://commons.wikimedia.org/wiki/File:Arbres_et_lumi%C3%A8res_Pac.JPG</p>	
	<p>Autoría: Michel Ngilen.</p> <p>Licencia: CC-by.</p> <p>Procedencia: http://commons.wikimedia.org/wiki/File:PlayStation_Comparaci%C3%B3n.jpg</p>	
	<p>Autoría: SharkD derivando una obra de David Vignoni.</p> <p>Licencia: GNU Lesser General Public License.</p> <p>Procedencia: http://en.wikipedia.org/wiki/File:Vg_icon.svg</p>	
	<p>Autoría: Poiuyt Man.</p> <p>Licencia: Dominio público.</p> <p>Procedencia: http://commons.wikimedia.org/wiki/File:Dance_Dance_Revolution_North_American_arcade_machine_3.jpg</p>	
	<p>Autoría: Wikibge.</p> <p>Licencia: GNU General Public License.</p> <p>Procedencia: http://commons.wikimedia.org/wiki/File:LogicBricksPythonScript.jpg</p>	
	<p>Autoría: Allen Gathman.</p> <p>Licencia: CC BY-SA.</p> <p>Procedencia: http://commons.wikimedia.org/wiki/File:Engranajes_-_Mezquita_de_C%C3%B3rdoba.jpg</p>	
	<p>Autoría: White Timberwolf.</p> <p>Licencia: GNU GPL.</p> <p>Procedencia: http://commons.wikimedia.org/wiki/File:SuperTux-Milestone1_9.png</p>	
	<p>Autoría: Steve Baker.</p> <p>Licencia: GPL.</p> <p>Procedencia: http://commons.wikimedia.org/wiki/File:Tuxracer.png</p>	
	<p>Autoría: Luis Ramón López.</p> <p>Licencia: CC-by.</p>	

	<p>Procedencia: Montaje a partir de la imagen Autoría: Killy Overdrive Licencia: CC-by Procedencia: http://opengameart.org/content/spaceship-360</p>	
	<p>Autoría: dbenzhuser.</p> <p>Licencia: CC BY-SA.</p> <p>Procedencia: http://commons.wikimedia.org/wiki/File:Pathfinding_A_Star.svg</p>	
	<p>Autoría: Guillaume Cottenceau, Alexis Younes (Ayo73), Matthias Le Bidan (Matths), Kim and David Joham, Amaury Amblard-Ladurantie.</p> <p>Licencia: GNU GPL.</p> <p>Procedencia: http://commons.wikimedia.org/wiki/File:Frozen_Bubble_2_-_Network.png</p>	
	<p>Autoría: SuperTuxKart Team.</p> <p>Licencia: GPL.</p> <p>Procedencia: http://supertuxkart.sourceforge.net/File:0_7_2_golf.jpg</p>	

