UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Curso de Desarrollo de Videojuegos con Blender 3D

Aprende a crear videojuegos 3D con el potente editor de Blender. Define el comportamiento de la aplicación mediante los Bloques Lógicos. Publica tus videojuegos, generando ejecutables para Windows, Mac y GNU/Linux.

Carlos González Morcillo Carlos.Gonzalez@uclm.es
David Vallejo Fernández David.Vallejo@uclm.es

esi Escuela Superior de Informática

tsi Departamento de Tecnología y Sistemas de Información

Escuela Superior de Informática · Paseo de la Universidad, 4 · 13071 Ciudad Real

Curso de Desarrollo de Videojuegos 3D con Blender

Escuela Superior de Informática
Departamento de Tecnologías y Sistemas de Información
Universidad de Castilla-La Mancha

Carlos González Morcillo (Carlos.Gonzalez@uclm.es)
David Vallejo Fernández (David.Vallejo@uclm.es)

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Curso de Desarrollo de Videojuegos con Blender 3D

Aprende a crear videojuegos 3D con el potente editor de Blender. Define el comportamiento de la aplicación mediante los Bloques Lógicos. Publica tus videojuegos, generando ejecutables para Windows, Mac y GNU/Linux.

Carlos González Morcillo Carlos.Gonzalez@uclm.es
David Vallejo Fernández David.Vallejo@uclm.es

esi Escuela Superior de Informática

tsi Departamento de Tecnología y Sistemas de Información

Escuela Superior de Informática · Paseo de la Universidad, 4 · 13071 Ciudad Real

Título: Curso de Desarrollo de Videojuegos 3D con Blender
Autores: (*Por orden alfabético*)
Carlos González Morcillo,
David Vallejo Fernández.
Diseño: Carlos González Morcillo
1ª Edición: Mayo 2014
Publica: Escuela Superior de Informática
(Universidad de Castilla-La Mancha)

Este libro fue compuesto y maquetado con LibreOffice4.1.
Imágenes editadas con GIMP, Inkscape y Blender.

Escuela Superior de Informática

Departamento de Tecnología y Sistemas de Información

Sintel:
CC-BY
Blender
Foundation.

Prefacio

La industria del videojuego ocupa el primer lugar en el ocio audio-visual e interactivo de España, con una cuota de mercado que supera el 50%. Nuestro país ocupa posiciones destacadas en el ranking mundial de consumidores de videojuegos y es quinto a nivel europeo. Esta situación como consumidores se ve igualmente reflejada en el ámbito del desarrollo. Gran cantidad de programadores españoles trabajan en importantes compañías del sector a nivel internacional. De igual modo, pequeños grupos de desarrolladores *Indie* crean producciones que triunfan Mundialmente. Desde el punto de vista del creador, las herramientas existentes han evolucionado igualmente.

Desde los primeros videojuegos que se «*cableaban*» directamente con hardware dedicado, hasta los modernos sistemas de construcción de alto nivel (como Game Maker, Unity o Blender). En este pequeño manual se recogen, a modo de apuntes, el material para el trabajo en el curso de introducción al desarrollo de videojuegos con Blender. El curso de 3 horas de duración está enfocado a ser una introducción básica al uso de Blender, pero muy enfocada a la producción de videojuegos. En la sesión se construye un ejemplo básico desde cero. Este manual recoge mucha más información que puede servir como primeros pasos para el trabajo autónomo.

Los archivos necesarios para seguir los ejemplos del capítulo 3 «*Blender Game Engine*» pueden descargarse de la siguiente página web:

⊫ http://www.esi.uclm.es/videojuegos/

Esperamos que este material sirva para que despierte el gusanillo del desarrollo de videojuegos, una de las disciplinas más apasionantes de la informática.

Disfruta del camino y... *Happy Blending!*

«El hombre sólo juega cuando es libre en el pleno sentido de la palabra, y sólo es plenamente hombre cuando juega.»

Friedrich Schiller (1759-1805)
Dramaturgo, filósofo e historiador

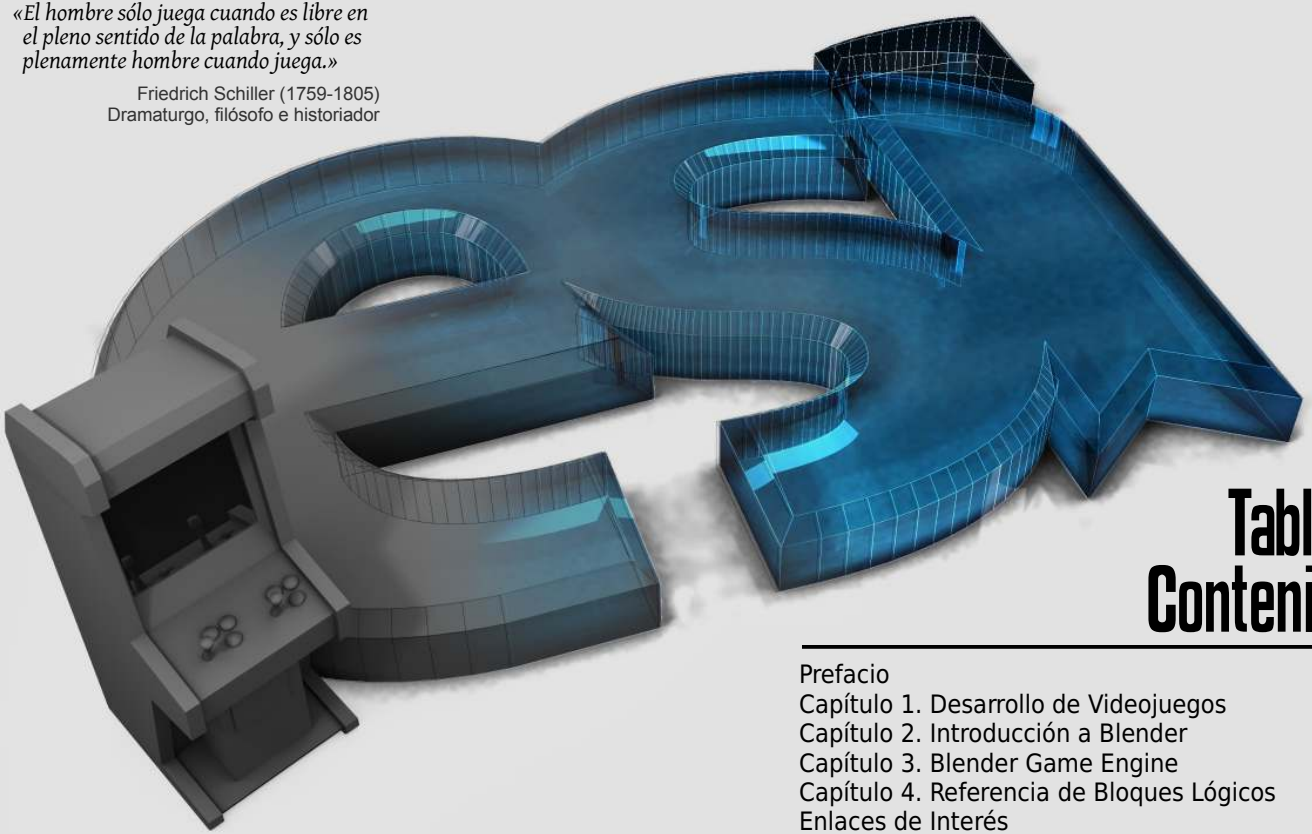
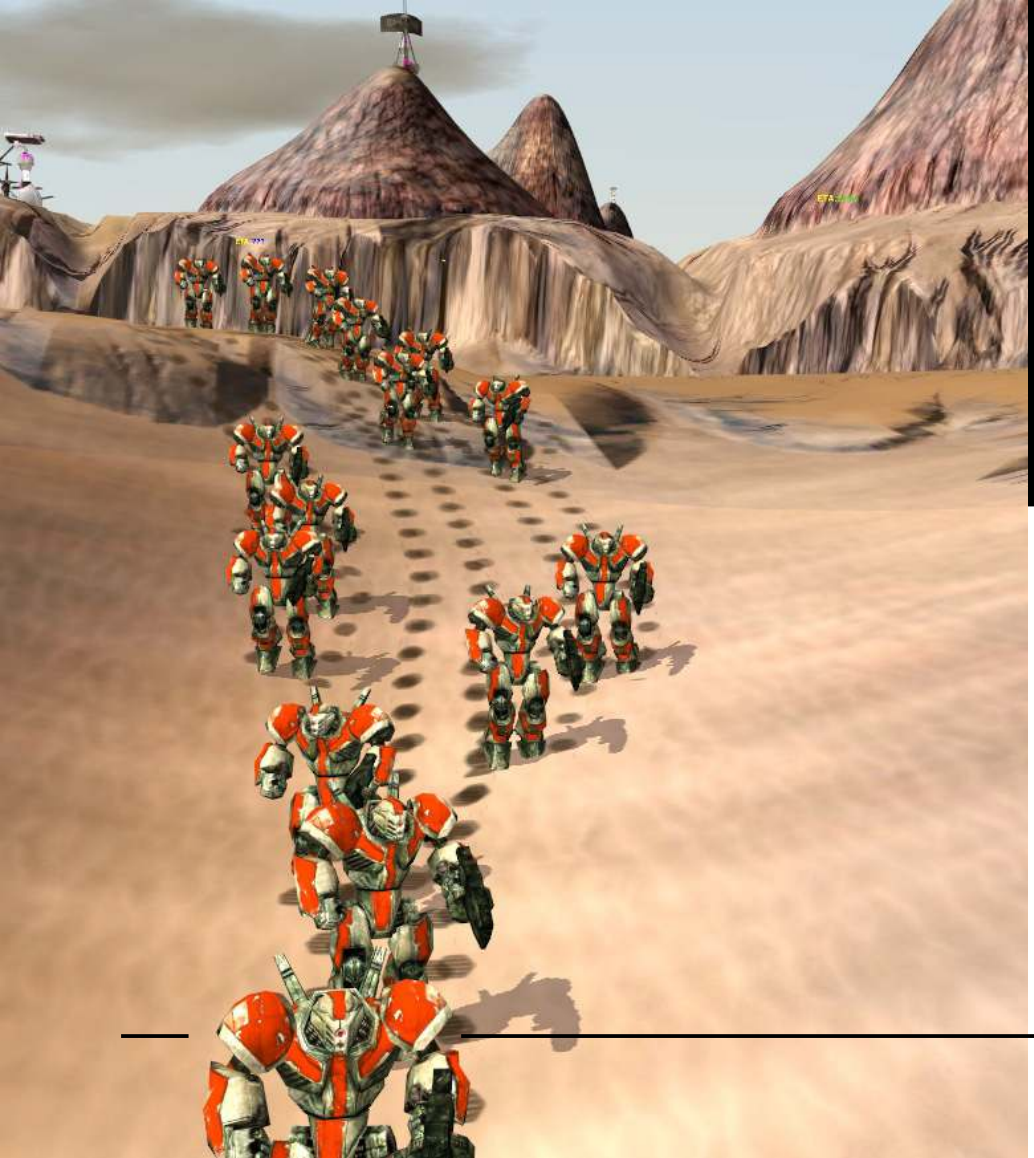


Tabla de Contenidos

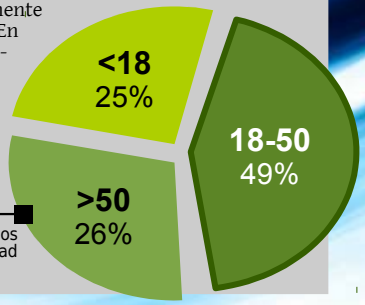


Capítulo 1
Desarrollo de Videojuegos

Actualmente, la industria del videojuego goza de una muy buena salud a nivel mundial, rivalizando en presupuesto con las industrias cinematográfica y musical. En este capítulo se discute, desde una perspectiva general, el desarrollo de videojuegos, haciendo especial hincapié en su evolución y en los distintos elementos involucrados en este complejo proceso de desarrollo.

Sabías que...

Según el informe de ESA (*Entertainment Software Association*), la edad media del usuario de software de entretenimiento está situada en **35 años**, contando con un **26%** sobre el porcentaje total de usuarios **mayores de 50 años**. Del total, aproximadamente un **40% son mujeres**. En los últimos años, la proporción de jugadores y jugadoras de videojuegos se está igualando.



Usuarios de videojuegos por rangos de edad

III La industria del videojuego

Lejos han quedado los días desde el desarrollo de los primeros videojuegos, caracterizados principalmente por su simplicidad y por el hecho de estar desarrollados completamente sobre hardware. Debido a los distintos avances en el campo de la informática, no sólo a nivel de desarrollo software y capacidad hardware sino también en la aplicación de métodos, técnicas y algoritmos, la industria del videojuego ha evolucionado hasta llegar a cotas inimaginables, tanto a nivel de jugabilidad como de calidad gráfica, tan sólo hace unos años.

La evolución de la industria de los videojuegos ha estado ligada a una serie de hitos, determinados particularmente por juegos que han marcado un antes y un después, o por fenómenos sociales que han afectado de manera directa a dicha industria. Juegos como *Doom*, *Quake*, *Final Fantasy*, *Zelda*, *Tekken*, *Gran Turismo*, *Metal Gear*, *The Sims* o *World of Warcraft*, entre otros, han marcado tendencia y han contribuido de manera significativa al desarrollo de videojuegos en distintos géneros.

Por otra parte, y de manera complementaria a la aparición de estas obras de arte, la propia evolución de la informática ha posibilitado la vertiginosa evolución del desarrollo de videojuegos. Algunos hitos clave son por ejemplo el uso de la tecnología poligonal en 3D en las consolas de sobremesa, el boom de los ordenadores personales como plataforma multipropósito, la expansión de Internet, los avances en el desarrollo de microprocesadores, el uso de shaders programables, el desarrollo de motores de juegos o, más recientemente, la eclosión de las redes sociales y el uso masivo de dispositivos móviles.

Por todo ello, los videojuegos se pueden encontrar en ordenadores personales, consolas de juego de sobremesa, consolas portátiles, dispositivos móviles como por ejemplo los smartphones, o incluso en las redes sociales como medio de soporte para el entretenimiento de cualquier tipo de usuario. Esta diversidad también está especialmente ligada a distintos tipos o géneros de videojuegos, como se introducirá más adelante en esta misma sección.

La ***expansión del videojuego*** es tan relevante que actualmente se trata de una industria multimillonaria capaz de rivalizar con las industrias cinematográfica y musical. Un ejemplo representativo es el valor total del mercado del videojuego en Europa, tanto a nivel hardware como software, el cual alcanzó la nada desdeñable cifra de casi 11.000 millones de euros, con países como Reino Unido, Francia o Alemania a la cabeza. En este contexto, España representa el cuarto consumidor a nivel europeo y también ocupa una posición destacada dentro del *ranking* mundial.

A pesar de la vertiginosa evolución de la industria del videojuego, hoy en día existe un gran número de ***retos*** que el desarrollador de videojuegos ha de afrontar a la hora de producir un videojuego. En realidad, existen retos que perdurarán eternamente y que no están ligados a la propia evolución del hardware que permite la ejecución de los videojuegos.

El más evidente de ellos es la necesidad imperiosa de ofrecer una experiencia de entretenimiento al usuario basada en la diversión, ya sea a través de nuevas formas de interacción, como por ejemplo la realidad aumentada o la tecnología de visualización 3D, a través de una mejora evidente en la calidad de los títulos, o mediante innovación en aspectos vinculados a la jugabilidad.

No obstante, actualmente la evolución de los videojuegos está estrechamente ligada a la evolución del hardware que permite la ejecución de los mismos. Esta evolución atiende, principalmente, a dos factores: i) la potencia de dicho hardware y ii) las capacidades interactivas del mismo.

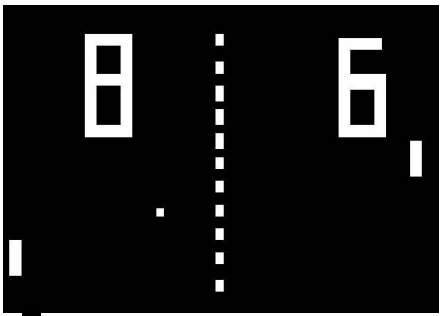


Figura 1

El videojuego Pong se considera como unos de los primeros videojuegos de la historia. Desarrollado por Atari en 1975, el juego iba incluido en la consola Atari Pong. Se calcula que se vendieron unas 50.000 unidades.

En el primer caso, una mayor potencia hardware implica que el desarrollador disfrute de mayores posibilidades a la hora de, por ejemplo, mejorar la calidad gráfica de un título o de incrementar la IA *(Inteligencia Artificial)* de los enemigos. Este factor está vinculado al multiprocesamiento. En el segundo caso, una mayor riqueza en términos de interactividad puede contribuir a que el usuario de videojuegos viva una experiencia más inmersiva (por ejemplo, mediante realidad aumentada) o, simplemente, más natural (por ejemplo, mediante la pantalla táctil de un *smartphone*).

Finalmente, resulta especialmente importante destacar la existencia de motores de juego (*game engines*), como por ejemplo *Quake* o *Unreal*, *middlewares* para el tratamiento de aspectos específicos de un juego, como por ejemplo la biblioteca *Havok* para el tratamiento de la física, o motores de renderizado.

Este tipo de herramientas, junto con técnicas específicas de desarrollo y optimización, metodologías de desarrollo, o patrones de diseño, entre otros, conforman un aspecto esencial a la hora de desarrollar un videojuego. Al igual que ocurre en otros aspectos relacionados con la Ingeniería del Software, desde un punto de vista general resulta aconsejable el uso de todos estos elementos para agilizar el proceso de desarrollo y reducir errores potenciales.

III Estructura de un Equipo de Desarrollo

El desarrollo de videojuegos comerciales es un proceso complejo debido a los distintos requisitos que ha de satisfacer y a la integración de distintas disciplinas que intervienen en dicho proceso. Desde un punto de vista general, un videojuego es una ***aplicación gráfica en tiempo real*** en la que existe una interacción explícita mediante el usuario y el propio videojuego. En este contexto, el concepto de tiempo real se refiere a la necesidad de generar una determinada tasa de frames o imágenes por segundo, típicamente 30 o 60, para que el usuario tenga una sensación continua de realidad. Por otra parte, la interacción se refiere a la forma de comunicación existente entre el usuario y el videojuego. Normalmente, esta interacción se realiza mediante *joysticks* o mandos, pero también es posible llevarla a cabo con otros dispositivos como por ejemplo teclados, ratones, cascos o incluso mediante el propio cuerpo a través de técnicas de visión por computador o de interacción táctil.

A continuación se describe la estructura típica de un equipo de desarrollo atendiendo a los distintos roles que juegan los componentes de dicho equipo. En muchos casos, y en función del número de componentes del equipo, hay personas especializadas en diversas disciplinas de manera simultánea.

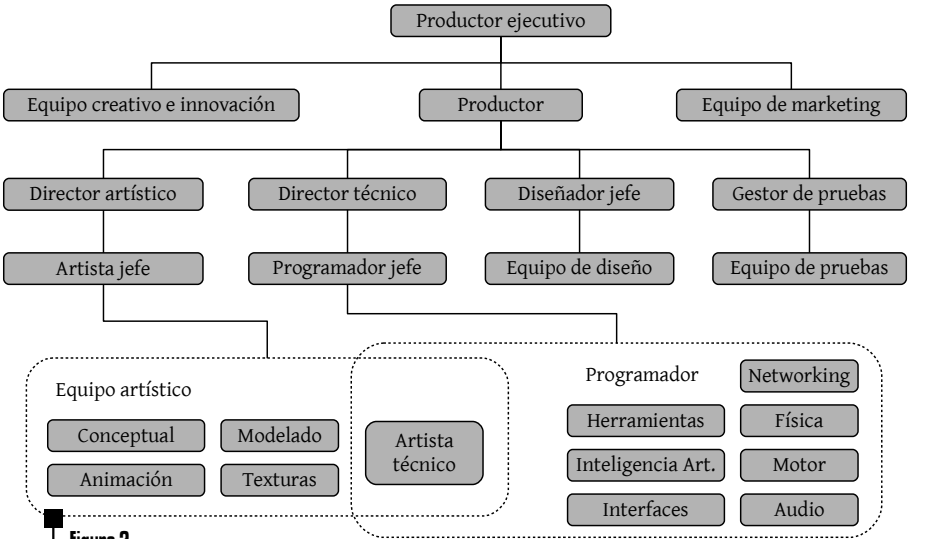


Figura 2

Visión conceptual de un equipo de desarrollo de videojuegos, considerando especialmente la parte de programación.

Los ***ingenieros*** son los responsables de diseñar e implementar el software que permite la ejecución del juego, así como las herramientas que dan soporte a dicha ejecución. Normalmente, los ingenieros se suelen clasificar en dos grandes grupos:

- Los ***programadores del núcleo*** del juego, es decir, las personas responsables de desarrollar tanto el motor de juego como el juego propiamente dicho.
- Los ***programadores de herramientas***, es decir, las personas responsables de desarrollar las herramientas que permiten que el resto del equipo de desarrollo pueda trabajar de manera eficiente.

De manera independiente a los dos grupos mencionados, los ingenieros se pueden especializar en una o en varias disciplinas.

Por ejemplo, resulta bastante común encontrar perfiles de ingenieros especializados en programación gráfica o en *scripting* e IA. Sin embargo, tal y como se sugirió anteriormente, el concepto de ingenie-ro transversal es bastante común, particularmente en equipos de desarrollo que tienen un número reducido de componentes o con un presupuesto que no les permite la contratación de personas especializadas en una única disciplina.

En el mundo del desarrollo de videojuegos, es bastante probable encontrar ingenieros senior responsables de supervisar el desarrollo desde un punto de vista técnico, de manera independiente al diseño y generación de código. No obstante, este tipo de roles suelen estar asociados a la supervisión técnica, la gestión del proyecto e incluso a la toma de decisiones vinculadas a la dirección del proyecto. Así mismo, algunas compañías también pueden tener directores técnicos, responsables de la supervisión de uno o varios proyectos, e incluso un director ejecutivo, encargado de ser el director técnico del estudio completo y de mantener, normalmente, un rol ejecutivo en la compañía o empresa.

Los ***artistas*** son los responsables de la creación de todo el contenido audio-visual del videojuego, como por ejemplo los escenarios, los personajes, las animaciones de dichos personajes, etc. Al igual que ocurre en el caso de los ingenieros, los artistas también se pueden especializar en diversas cuestiones, destacando las siguientes:

- ***Artistas de concepto***, responsables de crear bocetos que permitan al resto del equipo hacerse una idea inicial del aspecto final del videojuego. Su trabajo resulta especialmente importante en las primeras fases de un proyecto.
- ***Modeladores***, responsables de generar el contenido 3D del videojuego, como por ejemplo los escenarios o los propios personajes que forman parte del mismo.
- ***Artistas de texturizad***o, responsables de crear las texturas o imágenes bidimensionales que formarán parte del contenido visual del juego. Las texturas se aplican sobre la geometría de los modelos con el objetivo de dotarlos de mayor realismo.
- ***Artistas de iluminación***, responsables de gestionar las fuentes de luz del videojuego, así como sus principales propiedades, tanto estáticas como dinámicas.
- ***Animadores***, responsables de dotar de movimientos a los personajes y objetos dinámicos del videojuego. Un ejemplo típico de animación podría ser el movimiento de brazos de un determinado carácter.
- ***Actores de captura de movimiento,*** responsables de obtener datos de movimiento reales para que los animadores puedan integrarlos a la hora de animar los personajes.
- ***Diseñadores de sonido***, responsables de integrar los efectos de sonido del videojuego.
- ***Otros actores***, responsables de diversas tareas como por ejemplo los encargados de dotar de voz a los personajes.

General Vs Específico

En función del tamaño de una empresa de desarrollo de videojuegos, el nivel de especialización de sus empleados es mayor o menor. Sin embargo, las ofertas de trabajo suelen incluir diversas disciplinas de trabajo para facilitar su integración.

Al igual que suele ocurrir con los ingenieros, existe el rol de artista senior cuyas responsabilidades también incluyen la supervisión de los numerosos aspectos vinculados al componente artístico.

Los diseñadores de juego son los responsables de diseñar el contenido del juego, destacando la evolución del mismo desde el principio hasta el final, la secuencia de capítulos, las reglas del juego, los objetivos principales y secundarios, etc. Evidentemente, todos los aspectos de diseño están estrechamente ligados al propio género del mismo. Por ejemplo, en un juego de conducción es tarea de los diseñadores definir el comportamiento de los coches adversarios ante, por ejemplo, el adelantamiento de un rival.

Los diseñadores suelen trabajar directamente con los ingenieros para afrontar diversos retos, como por ejemplo el comportamiento de los enemigos en una aventura. De hecho, es bastante común que los propios diseñadores programen, junto con los ingenieros, dichos aspectos haciendo uso de lenguajes de *scripting* de alto nivel, como por ejemplo *Lua* o *Python*.

Como ocurre con las otras disciplinas previamente comentadas, en algunos estudios los diseñadores de juego también jue-gan roles de gestión y supervisión técnica.

Finalmente, también están presentes roles vinculados a la producción, especialmente en grandes estudios.

En algunas ocasiones, los productores también asumen roles relacionados con el diseño del juego. Así mismo, los responsables de marketing, de administración y de soporte juegan un papel relevante. También resulta importante resaltar la figura de publicador como entidad responsable del marketing y distribución del videojuego desarrollado por un determinado estudio. Mientras algunos estudios tienen contratos permanentes con un determinado publicador, otros prefieren mantener una relación temporal y asociarse con el publicador que le ofrezca mejores condiciones para gestionar el lanzamiento de un título.

El concepto de Juego

Dentro del mundo del entretenimiento electrónico, un ***juego*** normalmente se suele asociar a la evolución, entendida desde un punto de vista general, de uno o varios personajes principales o entidades que pretenden alcanzar una serie de objetivos en un mundo acotado, los cuales están controlados por el propio usuario. Así, entre estos elementos podemos encontrar desde superhéroes hasta coches de competición pasando por equipos completos de fútbol. El mundo en el que conviven dichos personajes suele estar compuesto, normalmente, por una serie de escenarios virtuales recreados en tres dimensiones y tiene asociado una serie de reglas que determinan la interacción con el mismo.

De este modo, existe una ***interacción*** explícita entre el jugador o usuario de videojuegos y el propio videojuego, el cual plantea una serie de retos al usuario con el objetivo final de garantizar la diversión y el entretenimiento. Además de ofrecer este componente emocional, los videojuegos también suelen tener un componente cognitivo asociado, obligando a los jugadores a aprender técnicas y a dominar el comportamiento del personaje que manejan para resolver los retos o puzzles que los videojuegos plantean.

Desde una perspectiva más formal, la mayoría de videojuegos suponen un ejemplo representativo de lo que se define como aplicaciones gráficas o ***renderizado en tiempo real***, las cuales se definen a su vez como la rama más interactiva de la Informática Gráfica. Desde un punto de vista abstracto, una aplicación gráfica en tiempo real se basa en un bucle donde en cada iteración se realizan los siguientes pasos:

- El usuario visualiza una imagen renderizada por la aplicación en la pantalla o dispositivo de visualización.
- El usuario actúa en función de lo que haya visualizado, interactuando directamente con la aplicación, por ejemplo mediante un teclado.
- En función de la acción realizada por el usuario, la aplicación gráfica genera una salida u otra, es decir, existe una retroalimentación que afecta a la propia aplicación.

En el caso de los videojuegos, este ciclo de visualización, actuación y renderizado ha de ejecutarse con una frecuencia lo suficientemente elevada como para que el usuario se sienta inmerso en el videojuego, y no lo perciba simplemente como una sucesión de imágenes estáticas. En este contexto, el ***frame rate*** se define como el número de imágenes por segundo, comúnmente fps, que la aplicación gráfica es capaz de generar.

figura 3

Para mantener constante el número de fps, los recursos artísticos deben ajustarse a las características soportadas por el Hardware. Sorprende la evolución en 12 años del número de polígonos del personaje principal de Tomb Raider.

A mayor *frame rate*, mayor sensación de realismo en el videojuego. Actualmente, una tasa de 30 fps se considera más que aceptable para la mayoría de juegos. No obstante, algunos juegos ofrecen tasas que doblan dicha medida.

Frames por Segundo...

Generalmente, el desarrollador de videojuegos ha de buscar un compromiso entre los fps y el grado de realismo del videojuego. Por ejemplo, el uso de modelos con una alta complejidad computacional, es decir, con un mayor número de polígonos, o la integración de comportamientos inteligentes por parte de los enemigos en un juego, o NPC (*Non-Player Character*), disminuirá los fps.

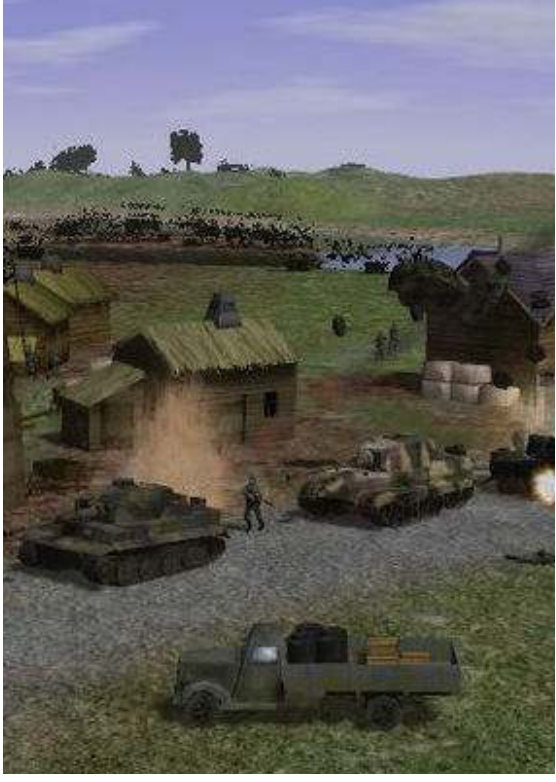
En otras palabras, los juegos son aplicaciones interactivas que están marcadas por el tiempo, es decir, cada uno de los ciclos de ejecución tiene un *deadline* que ha de cumplirse para no perder realismo.

Aunque el ***componente gráfico*** representa gran parte de la complejidad computacional de los videojuegos, no es el único. En cada ciclo de ejecución, el videojuego ha de tener en cuenta la evolución del mundo en el que se desarrolla el mismo.

Dicha evolución dependerá del estado de dicho mundo en un momento determinado y de cómo las distintas entidades dinámicas interactúan con él. Obviamente, recrear el mundo real con un nivel de exactitud elevado no resulta manejable ni práctico, por lo que normalmente dicho mundo se aproxima y se simplifica, utilizando modelos matemáticos para tratar con su complejidad. En este contexto, destaca por ejemplo la simulación física de los propios elementos que forman parte del mundo.

Por otra parte, un juego también está ligado al comportamiento del personaje principal y del resto de entidades que existen dentro del mundo virtual. En el ámbito académico, estas entidades se suelen definir como agentes (*agents*) y se encuadran dentro de la denominada simulación basada en agentes. Básicamente, este tipo de aproximaciones tiene como objetivo dotar a los NPC con cierta inteligencia para incrementar el grado de realismo de un juego estableciendo, incluso, mecanismos de cooperación y coordinación entre los mismos.

Respecto al personaje principal, un videojuego ha de contemplar las distintas acciones realizadas por el mismo, considerando la posibilidad de decisiones impredecibles a priori y las consecuencias que podrían desencadenar.



En resumen, y desde un punto de vista general, el desarrollo de un juego implica considerar un gran número de factores que, inevitablemente, incrementan la complejidad del mismo y, al mismo tiempo, garantizar una tasa de *fps* adecuada para que la inmersión del usuario no se vea afectada.

III El Motor de Juego

Al igual que ocurre en otras disciplinas en el campo de la informática, el desarrollo de videojuegos se ha beneficiado de la aparición de herramientas que facilitan dicho desarrollo, automatizando determinadas tareas y ocultando la complejidad inherente a muchos procesos de bajo nivel. Si, por ejemplo, los SGBD han facilitado enormemente la gestión de persistencia de innumerables aplicaciones informáticas, los motores de juegos hacen la vida más sencilla a los desarrolladores de videojuegos.

El término motor de juego surgió a mediados de los años 90 con la aparición del famosísimo juego de acción en primera persona *Doom*, desarrollado por la compañía id Software bajo la dirección de *John Carmack* (ver Figura 4). Esta afirmación se sustenta sobre el hecho de que Doom fue diseñado con una arquitectura orientada a la reutilización mediante una separación adecuada en distintos módulos de los componentes fundamentales, como por ejemplo el sistema de renderizado gráfico, el sistema de detección de colisiones o el sistema de audio, y los elementos más artísticos, como por ejemplo los escenarios virtuales o las reglas que gobernaban al propio juego.

Figura 4 John Carmack, uno de los desarrolladores de juegos más importantes.

Este planteamiento facilitaba enormemente la reutilización de software y el concepto de motor de juego se hizo más popular a medida que otros desarrolladores comenzaron a utilizar diversos módulos o juegos previamente licenciados para generar los suyos propios. En otras palabras, era posible diseñar un juego del mismo tipo sin apenas modificar el núcleo o motor del juego, sino que el esfuerzo se podía dirigir directamente a la parte artística y a las reglas del mismo.

Este enfoque ha ido evolucionando y se ha expandido, desde la generación de *mods* por desarrolladores independientes o amateurs hasta la creación de una gran variedad de herramientas, bibliotecas e incluso lenguajes que facilitan el desarrollo de videojuegos.

A día de hoy, una gran parte de compañías de desarrollo de videojuego utilizan motores o herramientas pertenecientes a terceras partes, debido a que les resulta más rentable económicamente y obtienen, generalmente, resultados espectaculares. Por otra parte, esta evolución también ha permitido que los desarrolladores de un juego se planteen licenciar parte de su propio motor de juego, decisión que también forma parte de su política de trabajo.

Obviamente, la separación entre motor de juego y juego nunca es total y, por una circunstancia u otra, siempre existen dependencias directas que no permiten la reusabilidad completa del motor para crear otro juego. Por ejemplo, un motor de juegos diseñado para construir juegos de acción en primera persona, será difícilmente reutilizable para desarrollar un juego de carreras de coches.

Como conclusión final, resulta relevante destacar la evolución relativa a la generalidad de los motores de juego, ya que poco a poco están haciendo posible su utilización para diversos tipos de juegos. Sin embargo, el compromiso entre generalidad y optimalidad aún está presente. En otras palabras, a la hora de desarrollar un juego utilizando un determinado motor es bastante común personalizar dicho motor para adaptarlo a las necesidades concretas del juego a desarrollar.

III Géneros de Juegos

Los motores de juegos suelen estar, generalmente, ligados a un tipo o género particular de juegos. Por ejemplo, un motor de juegos diseñado con la idea de desarrollar juegos de conducción diferirá en gran parte con respecto a un motor orientado a juegos de acción en tercera persona. No obstante, existen ciertos módulos, sobre todo relativos al procesamiento de más bajo nivel, que son transversales a cualquier tipo de juego, es decir, que se pueden reutilizar en gran medida de manera independiente al género al que pertenezca el motor. Un ejemplo representativo podría ser el módulo de tratamiento de eventos de usuario, responsable de recoger y gestionar las pulsaciones en el mando de juego.

A continuación, se realizará una descripción de los distintos géneros de juegos más populares atendiendo a las características que diferencian unos de otros en base al motor que les da soporte. Esta descripción resulta útil para que el desarrollador identifique los aspectos críticos de cada juego y utilice las técnicas de desarrollo adecuadas para obtener un buen resultado.

Probablemente, el género de juegos más popular ha sido y es el de los los denominados FPS, abreviado tradicionalmente como *shooters.*

Figura 5 Captura de pantalla del juego Tremulous, videojuego GPL y desarrollado sobre el motor de Quake III.

Algunos FPS famosos son *Quake*, *Half-Life*, *Call of Duty* o *Gears of War*, entre muchos otros. En este género, el usuario normalmente controla a un personaje con una vista en primera persona a lo largo de escenarios que tradicionalmente han sido interiores, como los típicos pasillos, pero que han ido evolucionando a escenarios exteriores de gran complejidad.

Los FPS representan juegos con un desarrollo complejo, ya que uno de los retos principales que han de afrontar es la inmersión del usuario en un mundo hiperrealista que ofrezca un alto nivel de detalle, al mismo tiempo que se garantice una alta reacción de respuesta a las acciones del usuario.

Figura 6 Captura de pantalla del juego Turtlearena, videojuego GPL y desarrollado sobre el motor de Quake III.

Otro de los géneros más relevantes son los denominados juegos en ***tercera persona,*** donde el usuario tiene el control de un personaje cuyas acciones se pueden apreciar por completo desde el punto de vista de la cámara virtual.

Aunque existe un gran parecido entre este género y el de los FPS, los juegos en tercera persona hacen especial hincapié en la animación del personaje, destacando sus movimientos y habilidades, además de prestar mucha atención al detalle gráfico de la totalidad de su cuerpo.

Ejemplos representativos de este género son *Resident Evil, Metal Gear, Gears of War* o *Uncharted*, entre otros.

Dentro de este género resulta importante destacar los juegos de ***plataformas***, en los que el personaje principal ha de ir avanzado de un lugar a otro del escenario hasta alcanzar un objetivo. Ejemplos representativos son las sagas de *Super Mario, Sonic* o *Donkey Kong*. En el caso particular de los juegos de plataformas, el avatar del personaje tiene normalmente un efecto de dibujo animado, es decir, no suele necesitar un renderizado altamente realista y, por lo tanto, complejo. En cualquier caso, la parte dedicada a la animación del personaje ha de estar especialmente cuidada para incrementar la sensación de realismo a la hora de controlarlo.

Otro género importante está representado por los juegos de ***lucha***, en los que, normalmente, dos jugadores compiten para ganar un determinado número de combates minando la vida o *stamina* del jugador contrario. Ejemplos representativos de juegos de lucha son *Virtua Fighter, Street Fighter, Tekken,* o *Soul Calibur*, entre otros.

Otro género representativo en el mundo de los videojuegos es la ***conducción***, en el que el usuario controla a un vehículo que normalmente rivaliza con más adversarios virtuales o reales para llegar a la meta en primera posición. En este género se suele distinguir entre simuladores, como por ejemplo *Gran Turismo*, y arcade, como por ejemplo *Ridge Racer* o *Wipe Out*.

Otro género tradicional son los juegos de estrategia, normalmente clasificados en tiempo real o RTS (*Real-Time Strategy*) y por turnos (*turn-based strategy*). Ejemplos representativos de este género son *Warcraft, Command & Conquer, Comandos, Age of Empires* o *Starcraft*, entre otros. Este tipo de juegos se caracterizan por mantener una cámara con una perspectiva isométrica, normalmente fija, de manera que el jugador tiene una visión más o menos completa del escenario, ya sea 2D o 3D. Así mismo, es bastante común encontrar un gran número de unidades virtuales desplegadas en el mapa, siendo responsabilidad del jugador su control, desplazamiento y acción.

Teniendo en cuenta las características generales de este género, es posible plantear diversas optimizaciones. Por ejemplo, una de las aproximaciones más comunes en este tipo de juegos consiste en dividir el escenario en una rejilla o grid, con el objetivo de facilitar no sólo el emplazamiento de unidades o edificios, sino también la planificación de movimiento de un lugar del mapa a otro. Por otra parte, las unidades se suelen renderizar con una resolución baja, es decir, con un bajo número de polígonos, con el objetivo de posibilitar el despliegue de un gran número de unidades de manera simultánea.

Finalmente, en los últimos años ha aparecido un género de juegos cuya principal característica es la posibilidad de jugar con un gran número de jugadores reales al mismo tiempo, del orden de cientos o incluso miles de jugadores. Los juegos que se encuadran bajo este género se denominan comúnmente MMOG (*Massively Multiplayer Online Game*). El ejemplo más representativo de este género es el juego *World of Warcarft*. Debido a la necesidad de soportar un gran número de jugadores en línea, los desarrolladores de este tipo de juegos han de realizar un gran esfuerzo en la parte relativa al networking, ya que han de proporcionar un servicio de calidad sobre el que construir su modelo de negocio, el cual suele estar basado en suscripciones mensuales o anuales por parte de los usuarios.

Al igual que ocurre en los juegos de estrategia, los MMOG suelen utilizar personajes virtuales en baja resolución para permitir la aparición de un gran número de ellos en pantalla de manera simultánea.

Además de los distintos géneros mencionados en esta sección, existen algunos más como por ejemplo los juegos deportivos, los juegos de rol o RPG (*Role-Playing Games*) o los juegos de puzzles.

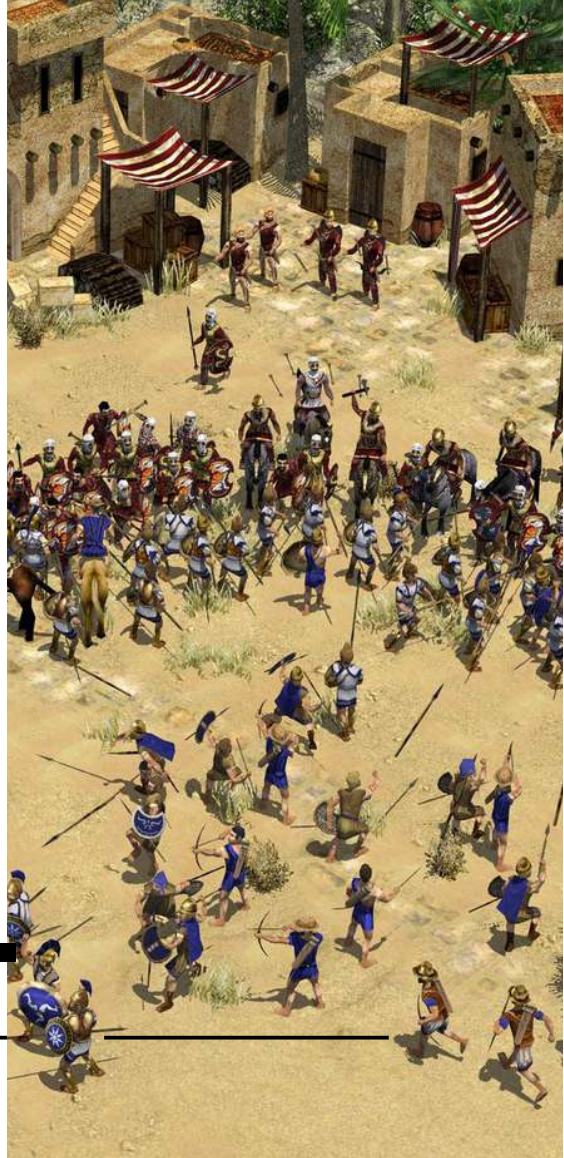


Figura 7

Captura de 0.A.D, un videojuego de estrategia en tiempo real de Wildfire.

Capítulo 2
Introducción a Blender

En los últimos 10 años se ha experimentado un crecimiento muy importante en el mundo del diseño por computador, y en concreto de la síntesis de imágenes tridimensionales. Las necesidades de perso nal cualificado en este sector son notables y continúan creciendo unidas al cada día mayor ámbito de aplicación del diseño 3D (visualización de datos, representación de imagen médica, simulación, videojuegos, etc.). Estas necesidades, igualmente paten tes desde hace años en el mundo del Soft ware Libre, han sido cubiertas con aplicaciones de calidad y uso profesional. En este capítulo estudiaremos los aspectos esenciales de Blender: la herramienta libre más potente en diseño 3D.

Introducción al Software Libre

Habitualmente existe confusión en lo referente al *Software Libre*; principalmente debido a que se confunde libertad con precio. Esta inter pretación errónea está ampliamente difundida, en parte, por los intereses comerciales de grandes multina cionales. Hay que entender el término libre como sinónimo de libertad no de barra libre. Entendemos como software libre aquel que se basa en el cumplimiento de (al menos) cuatro libertades básicas:

1. Libertad para utilizar el programa para lo que quieras.
2. Libertad para estudiar el el programa (para poder realizar un estudio del programa es necesario disponer del código fuente; que es una descripción entendible por el programador).
3. Libertad para redistribuir el programa, y compartir sus beneficios con quien tú quieras.
4. Libertad para mejorar el programa y distribuir sus mejoras.

Para saber más...

Puedes consultar la definición de Software Libre dada por la Free Software Foundation, la principal organización mundial de Software Libre.

▸http://www.gnu.org/philosophy/free-sw.es.html

Si leemos con atención las libertades básicas anteriores, veremos que en ningún caso se menciona nada sobre el precio del software. Para que un programa sea conside rado Software Libre, debe cumplir las condiciones anteriores. Existen multitud de licencias que son consideradas válidas para Software Libre. Una de ellas (y quizás la más extendida) es la GNU GPL (Licencia Pública General) de la FSF (Free Software Foundation).

Las licencias *Freeware* no permiten estudiar y mejorar el programa (por no disponer del código fuente). Esto es un impedimento crítico, ya que no permite la generación de software basado en estos programas, e impide el desarrollo económico local. Tampoco es correcta (aunque sí está muy extendida) la distinción entre **Software Comercial** y **Software Libre**. Un programa Libre puede ser comercial; se puede obtener un beneficio económico de él, y es totalmente lícito (y recomendable) ganar dinero realizando adaptaciones y dando soporte de este software. Es más correcto denominar **Software Privativo** al software que no es libre, porque nos priva de nuestras libertades básicas.

Linux es el núcleo de un Sistema Operativo que se distribuye bajo una licencia de Software Libre. En realidad, cuando instalamos un Linux, estamos instalando el núcleo del Sistema Operativo y un conjunto de utilidades que forman parte del proyecto GNU (GNU is Not Unix).

Figura 1

Linux, un núcleo de Sistema Operativo Libre.

Es más correcto hablar de distribuciones de GNU/Linux, ya que están formadas por el núcleo del Sistema Operativo y las herramientas libres de GNU. Existen muchas distribuciones de Linux; la mayoría basadas en Red Hat o Debian. Entre ellas podemos destacar Fedora (Red Hat), Suse, Mandriva, Debian, Ubuntu (basada en Debian), etc... Todas tienen un núcleo de Linux común, e internamente se diferen cian en pocos aspectos importantes.

Se puede encontrar multitud de Software Libre para otros sistemas operativos que no son Linux. De hecho, la mayoría de las aplicaciones libres de GNU/Linux se pueden encontrar también en Windows y Mac entre otros. Existen multitud de razones por las que es conveniente utilizar exclusivamente (en la medida de lo posible) software libre, entre otras podemos citar:

- **El software libre favorece el Desarrollo Económico Local.** Las in versiones, en vez de realizarse "alquilando" licencias a grandes multinacionales, se realizan en empresas de desarrollo nacionales. Esto es claramente positivo para la comunidad.
- **El software libre favorece la Innova ción.** No se repite mil veces lo mismo; los programadores sólo se ocupan de programar nueva funcionalidad.
- **El software libre es una clave Competitiva.** Utilizando software libre los usuarios siempre pueden estar actualizados, normalmente con coste cero. No hay problemas de licencias utilizando Software Libre.
- **El software libre nos hace ser más libres.** No existen formatos cerrados o propietarios. Las aplicaciones pueden compartir sus datos sin problema. Cada usuario puede utilizar el software que desee y, con seguridad, existirán conversores libres entre formatos de ficheros.

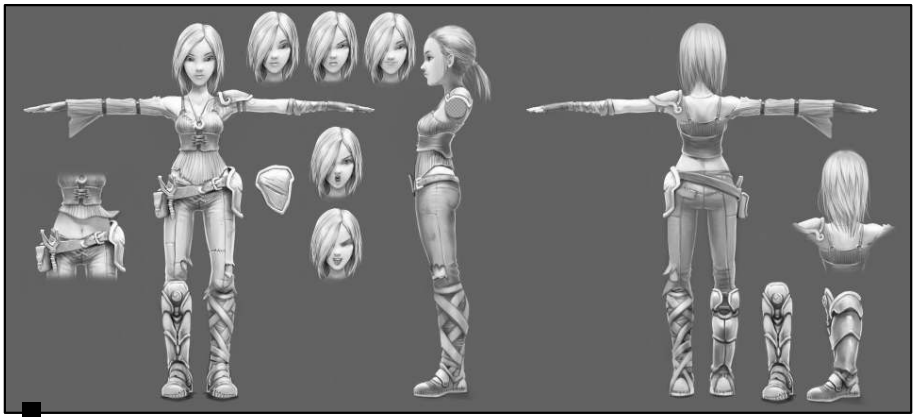


Figura 2 Hoja de personaje de Sintel por David Revoy. Pintada sobre una base en 3D. CC-BY Blender Foundation - www.sintel.org.

El Ciclo de Producción 3D

Podemos definir el ciclo de trabajo en producciones 3D estableciendo tres grandes fases de producción, cada una de las cuales tendrá asociadas un conjunto de tareas (algunas de estas tareas pueden no estar presentes en ciertos proyectos, como la postproducción en el caso de Videojuegos). Las fases de producción suelen completarse secuencialmente, y normalmente no hay que ejecutar tareas de una fase anterior, cuando ésta se ha completado. Las tareas dentro de una fase de trabajo siguen un orden aunque ajustes en tareas posteriores pueden requerir cambios en tareas que estaban completas.

Preproducción

Esta fase comienza habitualmente con la escritura del guión en el caso de generar un vídeo, o en el caso de Videojuegos, con la definición de un documento de diseño de videojuego. Con el guión (o el documento de diseño del videojuego) definido, el equipo de desarrollo visual, habitualmente formado por ilustradores, establecen la dirección visual y el estilo del proyecto. Se eligen los colores clave que complemen tarán visualmente las metas de cada parte de la narración. De igual forma, se desarrollan las hojas de personajes (ver Figura 2), con los bocetos de los personajes que serán incluidos en el desarrollo.

Modelado

En esta etapa se obtiene una representación tridimensional de los objetos que interven drán en la escena. Existen multitud de técni cas y herramientas de modelado. Dependien do de la forma a modelar y el acabado que se desee obtener, será mejor emplear una u otra.

Materiales y Texturas

Mediante los materiales, aplicamos propiedades básicas de reflexión de la luz, color, transparencia a las superficies de nuestros modelos. El material se aplica de forma constante a lo largo de toda la superficie del modelo. Las texturas permiten variar las propiedades del material.

Iluminación

En la búsqueda de la generación de imagen lo más realista posible, un punto clave es la simulación de la luz. Según las interacciones de la luz que sea capaz de simular el método de render, obtendremos resultados con diferente nivel de realismo. En videojuegos es habitual simular los rebotes de la luz y emplear métodos que asignan la iluminación como texturas. En Blender este precálculo se realiza emple ando opciones de *Baking* (o cocinado) del render sobre texturas.

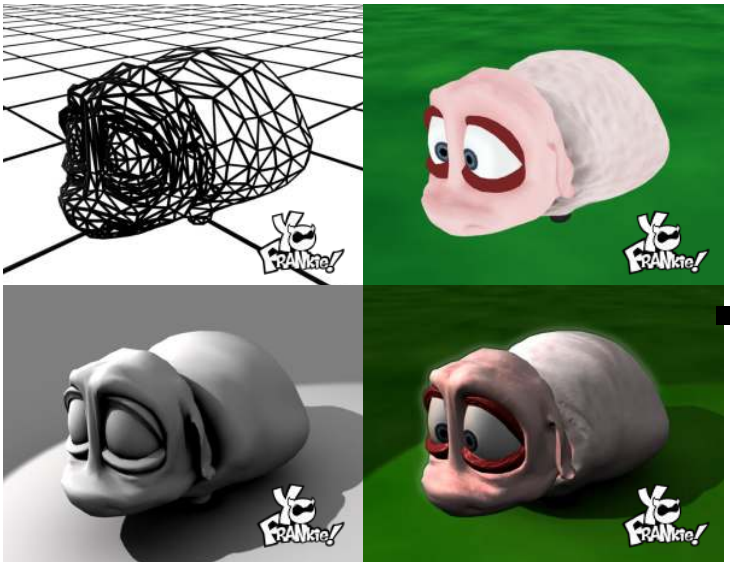


Figura 3

Diferentes etapas de Producción: Modelado, Descripción de Materiales y Texturas, Iluminación y resultado de Render Interactivo.

Animación

Tanto las herramientas de animación 2D como las 3D suelen emplear curvas de interpolación para calcular cambios en los parámetros a animar (ya sea posición de un objeto, propiedades de material, etc...) entre frames clave. De esta forma, el usuario de la aplicación únicamente debe identi ficar esos fotogramas clave, establecer los valores de las propiedades en esos puntos y el programa se encarga de calcular los valores intermedios.

Para saber más...

Queda fuera del alcance de este pequeño manual enseñar las opciones de modelado, asignación de materiales, texturas y animación en Blender. Es muy recomendable que el lector profundice su formación en todos estos aspectos. En la sección de Bibliografía se recogen algunos manuales y tutoriales On-Line línea para aprender más sobre Blender.

En animación de personajes suelen emple arse esqueletos internos de animación. Así, el animador establece las rotaciones de estos huesos y el programa calcula la deformación de debe aplicar a la superficie exterior. Este tipo de animación jerárquica es ampliamente utilizada en el desarrollo de videojuegos profesionales. La complejidad en la definición del esqueleto (llamado *rig*) hace que algunas personas del equipo de desarrollo de un videojuego estén especiali zadas únicamente en esta tarea.

Render

En este paso se realiza el cálculo de la imagen 2D (habitualmente de tipo mapa de bits, o imagen *Raster*) correspondiente a la escena definida. En Videojuegos se utilizan métodos de render interactivos, que tratan de obtener entre 50 y 60 fotogramas por segundo. En la generación de vídeos pueden emplearte métodos más realistas, y los tiempos de *Render* suelen ser notablemente superiores (desde minutos hasta horas o días por *fotograma*).

Postproducción

Esta fase toma como entrada las imágenes generadas en la etapa anterior y las compone, aplicándoles una serie de filtros. Algunos efectos (como profundidad de campo, motion blur, etc.) es menos costoso generarlos independientemente y compo nerlos mediante el uso de capas. Si el fondo es estático, puede suponer un importante ahorro de tiempo (como en algunas escenas de la película *Big Buck Bunny*, ver Figura 4).

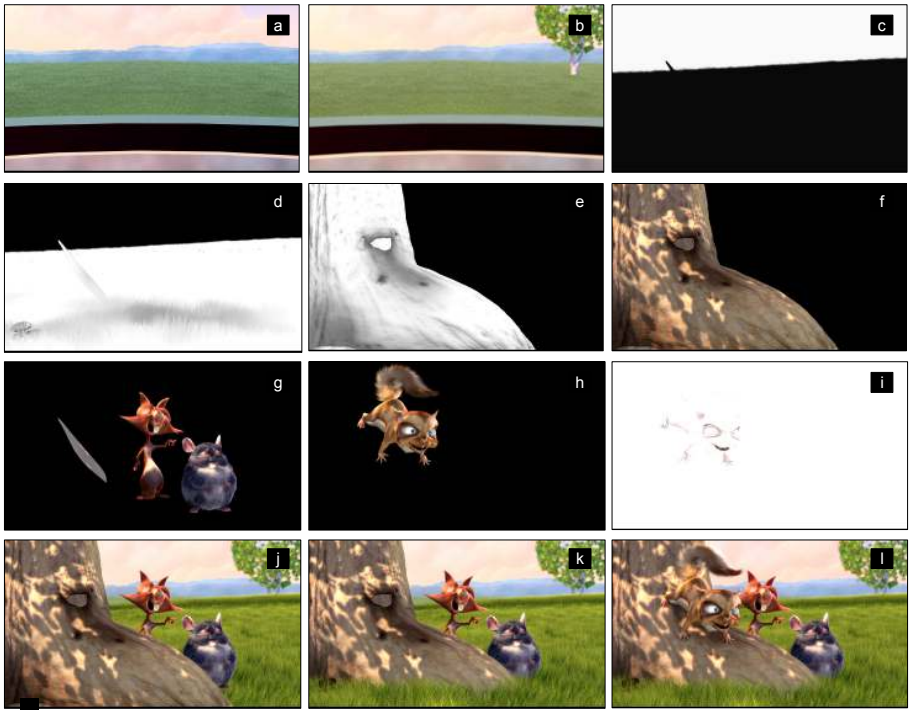


Figura 4 Etapas de render por capas que serán compuestas en la etapa de Postproducción. a) Fondo estático. b) Profundidad de campo. c) Máscara de plano medio. d) Sombra de primer plano. e) Capa de oclusión ambiental (primer plano). f) Sombra + Color primer plano (animado). g y h) Capa con personajes (animados). i) Sombra oclusión ambiental propia del personaje primer plano. j y k) Composición de capas anteriores. l) Resultado final.

III Blender en Entornos Profesionales

Han pasado 20 años desde que *NeoGeo* (posteriormente *Not a Number*), la compañía liderada por Ton Roosendaal, creara la primera versión de Blender en 1995 para interno. Inicialmente distribuida como una herramienta freeware en 1998, fue posterior mente liberada en 2002 por la comunidad de usuarios en una espectacular colecta de fondos por parte de la Blender Foundation. En tan sólo siete semanas, los usuarios de Blender donaron un total de 100.000 euros para liberar la herramienta. En octubre de 2002 se liberó la primera versión de este programa bajo licencia GPL.

Desde entonces, el número de usuarios y prestaciones de Blender no ha parado de crecer hasta convertirse, según la revista 3D World, en la herramienta gráfica más instalada del mundo, situándose por encima de 3D Studio MAX (en tercer puesto) o Maya (décimo puesto). Más de 2.000 millones de descargas anuales convierten a Blender en una de las herramientas libres más demandadas del mundo. Tras estas impac tantes cifras se esconde un software para producción 3D con posibilidades profesionales (Por ejemplo, fue utilizado en Spiderman II en el estudio de cinemática, en anuncios de Oral-B, Coca-Cola, etc.)

En el ámbito de los videojuegos, proyectos como *Yo Frakie!* Muestran el potencial de Blender como herramienta de desarrollo integral. Todos los ficheros de desarrollo fueron liberados en su web oficial. En foros de usuarios de Blender, como *BlenderArtists* o foros generales de gráficos 3D, como *3D Poder* o *CG Talk* encontramos multitud de empresas nacionales e internacionales que emplean Blender profesionalmente.

Para saber más...

Descarga el videojuego de *YoFrankie* en la web oficial del mismo:

►/http://www.yofrankie.org/

Por qué elegir Blender

Blender incorpora gran cantidad de herramientas avanzadas para el modelado, construcción de materiales, animación, render, composición y creación de Video juegos que lo convierten en una alternativa perfecta tanto para el aprendizaje de los gráficos 3D como para su explotación a nivel profesional... ¡y en poco más de 70MB!

Modelado. En lo referente al modelado, además de los operadores clásicos que podemos utilizar en modo de edición de vértices, aristas y caras, Blender imple menta superficies de subdivisión de *Cat mull-Clark*, métodos para trabajar con mayas de resolución adaptativa, metabolas y meta superficies, y un largo etcétera. Una de las mejoras más útiles para el modelado orgánico es la incorporación del modo de Esculpir (*Sculpt Mode*), que permite utilizar la metáfora del «*pincel 3D*» con una semántica similar a la aplicación comercial Z-Brush.

Materiales y Texturas. La gestión en nodos y modificadores de materiales y texturas ha permitido combinar las propie dades de diversos materiales. La flexibilidad de esta aproximación, unida a la incorpo ración de propiedades avanzadas como el conocido *SubSurface Scattering* (que simula los rebotes de la luz en el interior de los objetos), el *Render Baking* (que precalcula el resultado del render proyectándolo sobre una textura con *UV Mapping*) o la posibi lidad de dibujar directamente sobre la malla 3D permiten que Blender supere a muchas aplicaciones comerciales en esta etapa del flujo de trabajo.

Animación. Con la creación de la película Elephants Dream se añadieron contro ladores de alto nivel para la animación de personajes, animación facial y herramientas de sincronización con audio, que posterior mente mejoraron en Big Buck Bunny, y fueron mejoradas profundamente en Sintel. El módulo de Animación No Lineal (NLA), que tantos problemas presentaba hasta la versión 2.40, fue mejorado hasta conver tirse en una alternativa real a la animación de personajes profesional. Gracias a los proyectos financiados por Google se han realizado grandes avances en animación basada en sistemas de partículas, como fluidos, pelo, etc. La conexión con el *Game Engine* (y el motor físico *Bullet*) permite realizar simulaciones físicas de gran calidad. Además, desde la versión 2.61 se incorpora un completo módulo de tracking de movimiento que puede ser utilizado para componer imagen real con objetos vir tuales.

Render. El motor interno de Render que implementa Blender ofrece muy buenos resultados con un bajo tiempo de cómputo. Blender incorpora un trazado de rayos clásico, sin soporte de métodos de iluminación global pero con algunas aproxi maciones interesantes (como métodos de Oclusión Ambiental). Desde la versión 2.61, Blender incorpora Cycles, un motor de iluminación global que utiliza la GPU. Además, se pueden utilizar otros motores externos como YafaRay, LuxRender y Pov Ray, entre otros.

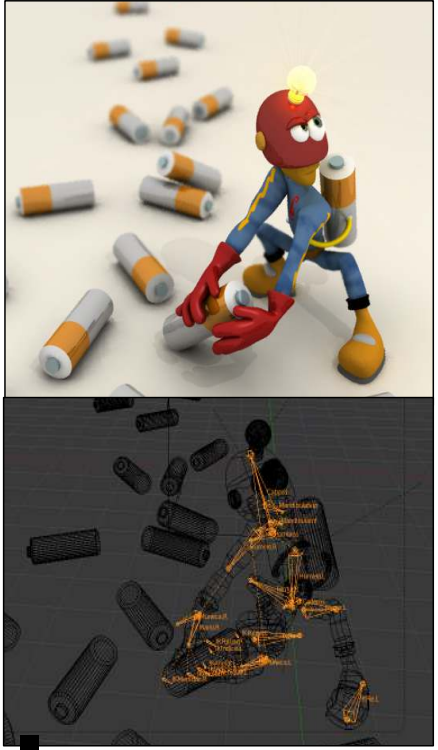


Figura 5 Ejemplo de uso de un sistema de huesos jerárquico en un personaje.

Toma de Contacto

Si es la primera vez que ejecutas Blender, posiblemente tengas la misma sensación que tuvimos todos los usuarios de Blender cuando tomamos contacto con la herra mienta; la interfaz de usuario de Blender es tan complejo que simplemente jugando con la aplicación no conseguíamos hacer casi nada. Por suerte la interfaz de Blender está diseñado para ser totalmente consistente, de forma que casi cualquier acción que puedas realizar en una ventana podrás realizarla con el mismo operador en ventanas de otro tipo.

La interfaz de Blender fue diseñada para ser altamente productivo. Por esta razón suelen emplearse atajos de teclado. No sufras, toda operación tiene su equivalente accediendo a opciones de menú, pero es aconsejable que las más habituales trates de utilizarlas mediante los atajos de teclado.

Antes de comenzar indicaremos los sím bolos que se utilizarán a lo largo de este manual. Los símbolos que se emplearán se resumen en el siguiente listado.

- **Ratón.** Mediante los siguientes iconos representaremos la pulsación del botón izquierdo, botón central o botón derecho. Es muy recomendable disponer de un ratón de tres botones para trabajar con Blender. En muchos modelos, el botón central es una rueda que también puedes presionar.
- **Teclado.** Representaremos combinacio nes de teclas empleando el siguiente estilo de texto: Ctrl + ⇧ + A. La combi nación anterior implica que hay que presionar la tecla Control, la tecla Shift (tiene dibujada una flecha que apunta hacia arriba, situada justo encima de Control; en muchos teclados en castellano es «*Mayús*») y la tecla A.
- **Botón.** Las pulsaciones de botones del interfaz de blender se indicarán mediante el siguiente estilo: Render. En el caso de ser un botón con una imagen asociada, utilizaremos el icono corres pondiente, como por ejemplo .
- **Opciones de Menú.** Para indicar operaciones en algunos de los menús de Blender, utilizaremos el siguiente estilo de texto: **File/Save As...**

Primeros Pasos

Al ejecutar Blender, obtenemos una interfaz como el que se muestra en la Figura 7. El espacio de trabajo de Blender es totalmente personalizable; podemos dividir la pantalla como nos resulte más cómodo y situar en cada porción el tipo de ventana que queramos (veremos cómo realizar esta personalización más adelante).

Por defecto Blender divide la pantalla en varias secciones, la ventana superior de tipo *Info*, un área central de tipo *3D View*, una inferior *Timeline* y a la derecha dos secciones (*Outliner* y *Properties*).

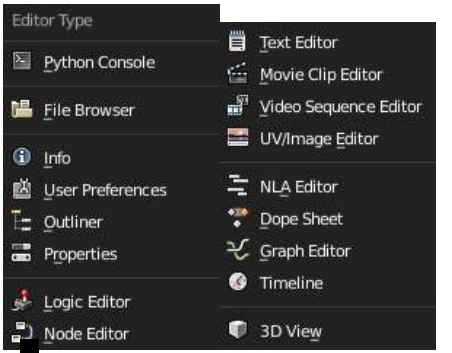


Figura 6 Diferentes tipos de Editores disponibles en Blender.

Podemos cambiar el tipo de editor pinchando en el selector de tipo de ventana y eligiendo alguno de los tipos que se describen brevemente a continuación. En este manual nos centraremos principal mente en el editor de vista 3D (*3D View*) y el editor de Lógica (*Logic Editor*).

Python Console. Permite utilizar directa mente un intérprete de *Python* interactivo (además de poder una consola de shell). Incluye un modo de autocompletado muy útil para el desarrollo de scripts que utili cen la API de *Python.*

File Browser. Un completo sistema de navegación de archivos, con capacidad de añadir *Bookmarks*, previsualizar archivos de tipos conocidos, filtrar búsquedas, etc.

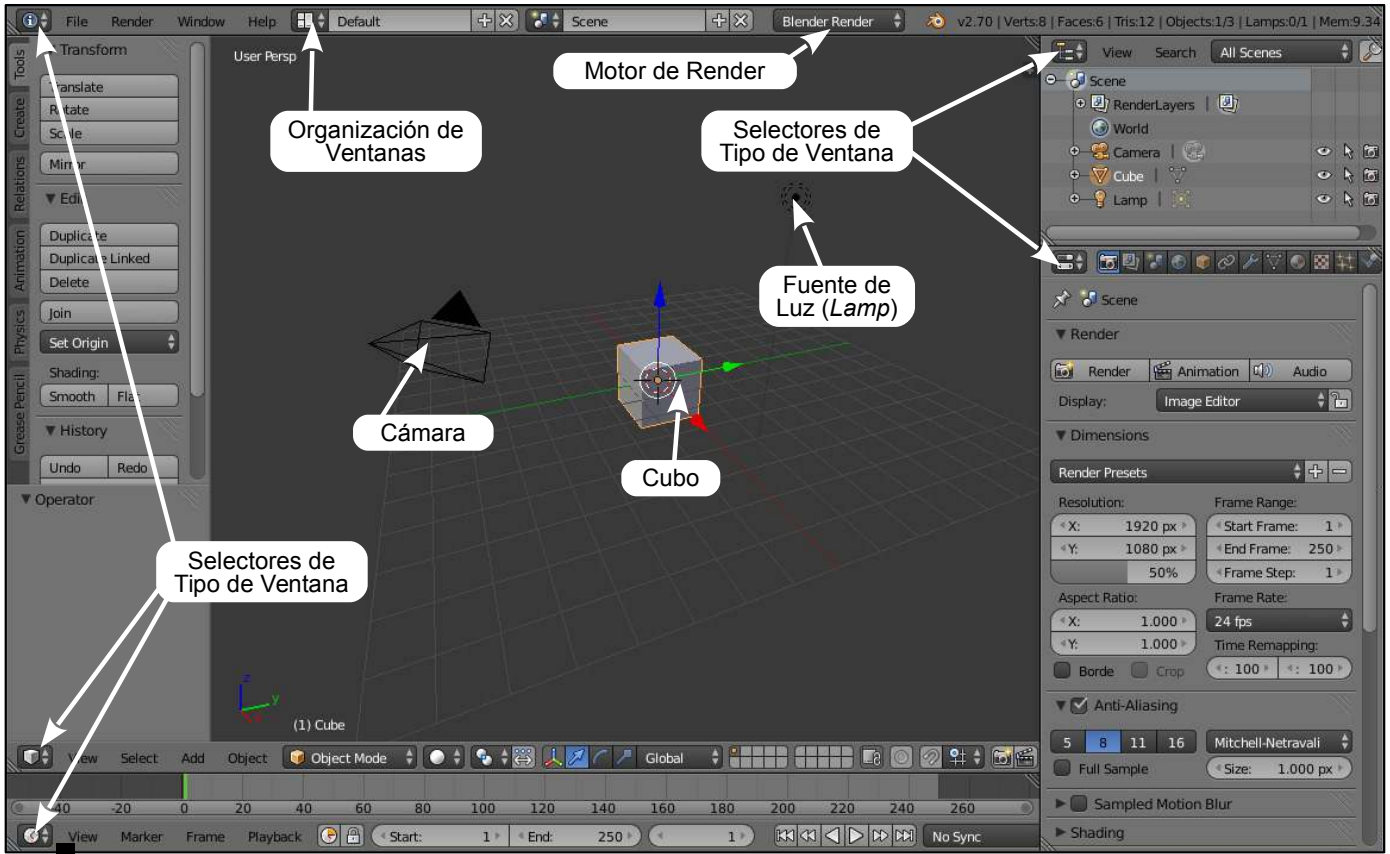


Figura 7 Identificación de algunos de los elementos principales del interfaz de Blender.

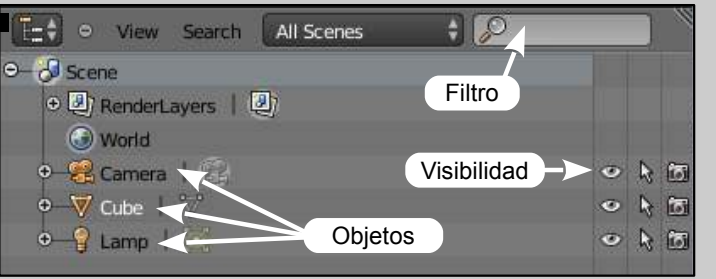
Info. Menú principal de Blender con información sobre la escena actual (polígonos, memoria utilizada, etc). Con tiene el menú general para trabajar con archivos, añadir objetos a la escena, elegir el motor de render (entre el motor inte ractivo para Videojuegos o dos motores para síntesis de imagen realista). Por defecto, aparece en la parte superior del interfaz.

User Preferences. Mediante esta ventana pueden configurarse las preferencias del usuario relativas al interfaz, métodos de edición, rutas de archivos por defecto y opciones de *OpenGL* entre otras.

Outliner. Representación en modo de árbol de los elementos que actualmente forman nuestra escena. Resulta muy útil cuando la escena tiene muchos objetos, y se hace difícil su gestión.

Outliner

Este editor permite seleccionar fácilmente los objetos de la escena y filtrarlos por su nombre.



Cuando la complejidad de la escena aumenta, resulta muy útil para seleccionar los objetos fácilmente.

Properties. Esta ventana es una de las más importantes de la interfaz. Muestra una serie de submódulos para acceder a las propiedades de los objetos de la escena.

Logic Editor. Permite definir propiedades de interacción a los elementos de la escena en términos de sensores, controladores y actuadores en el motor de videojuegos.

Node Editor. Esta ventana permite acceder al potente sistema de nodos de materiales, texturas y de composición. Mediante el editor de nodos de composición se pueden realizar ajustes de postproducción sin salir de Blender. Esta funcionalidad permite refinar el resultado del render con muy poco coste computacional.

Sobre el Info Editor..

El *Info Editor* incorpora la funcionalidad básica del menú principal de cualquier aplicación, como Abrir, Guardar, Exportar... Además, resulta muy útil el menú de preferencias de usuario ⊫ **File/ User Preferences** que utilizaremos para configurar diversos aspectos de la herramienta. En este menú se selecciona igualmente el motor de Render que emplearemos. En el caso de desarrollar videojuegos, elegiremos «*Blender Game*».

Text Editor. Editor integrado de texto. Permite editar y guardar cualquier fichero de texto.

Movie Clip Editor. Desde este módulo se realizan las operaciones de tracking de movimiento. En el futuro se añadirán nuevas funcionalidades relacionadas con la creación de máscaras.

Video Sequence Editor. Editor de vídeo integrado. Permite combinar varios vídeos, imágenes y ficheros de audio empleando efectos de transición. Es otro de los módulos que no son habituales en pro gramas de edición 3D y que permiten a los usuarios de Blender cerrar el ciclo de producción completo sin salir del progra ma.

UV/Image Editor. Editor de imágenes y mapas UV. Permite posicionar con exac titud texturas de imagen en los modelos 3D.

NLA Editor. Editor de Animación No Lineal. Permite trabajar con barras de animación de forma independiente y combinarlas para obtener composiciones de movimiento complejas.

Dope Sheet. Este término se refiere a la hoja que permite planificar cómo se realizará la animación. En Blender esta ventana permite trabajar cómodamente con los marcos clave definidos a nivel de propie dades de los objetos, cámara, etc.

La ventana de vista 3D nos muestra un cubo gris con borde en color naranja en el centro de la pantalla. Este cubo lo crea Blender en la escena por defecto (ver Figura 7). El color naranja del borde indica que está seleccio nado. El círculo situado a su derecha es el foco por defecto, y la pirámide negra representa la cámara virtual. La Figura 7 muestra también una rejilla que permitirá situar los objetos sobre ese sistema de referencias. El círculo de color rojo y blanco situado en el centro de la ventana (justo en el centro del cubo) es el puntero 3D. Los elementos nuevos que se añadan a la escena lo harán en la posición del puntero 3D.

Figura 8 **Arriba**. Zonas para la división y unión de ventanas. **Abajo**. Para dividir una ventana, pinchamos con botón izquierdo del ratón en la zona de división y arrastramos (en el caso de la figura, horizontalmente).

Antes de empezar a trabajar, es importante saber cómo crear y destruir ventanas en Blender. Estas operaciones (ver Figura 8) se pueden realizar de dos formas. La primera es pinchando pinchando con y despla zando hacia un lado en la zona de unión de dos ventanas, marcadas con tres líneas diagonales (si movemos sobre ella el puntero del ratón, cambiará a la forma de una cruceta). La otra forma de dividir ventanas es situando el puntero del ratón sobre la línea entre dos ventanas y pinchando . Como se muestra en la Figura 9, aparecerá una nueva ventana flotante titulada *Area Options* que nos permitirá dividir (*Split Area*) o unir (*Join Area*) las ventanas.

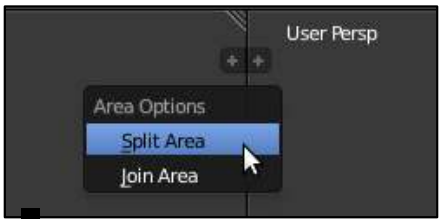


Figura 9 Ventana flotante para la división de áreas de trabajo.

Atajos de Teclado

3D View (Básico)

Cambiar Cursor 3D	
Rotar Vista	
Seleccionar	
Zoom	(Rueda)
Desplazar	⇧
Añadir Objeto	Alt A
Borrar Objeto	X o Del
Deshacer (*Undo*)	Ctrl Z
Rehacer (*Redo*)	Ctrl ⇧ Z
Herramientas	T
Propiedades	N

3D View (Selección)

Seleccionar	
Seleccionar Múltiple	⇧
(De)Seleccionar Todo	A
Invertir Selección	Ctrl I
Selección Caja (Box)	B
Selección Circular	C
Selección Lazo	Ctrl

3D View (Transform.)

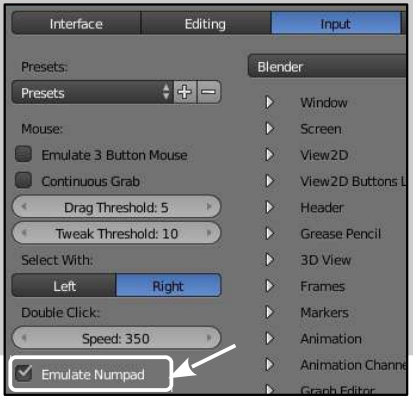
Mover (Grab)	G
Rotar	R
Escalar (Scale)	S
Precisión	⇧ (Mantener)
Rejilla (Grid)	Ctrl (Mant.)
Forzar Eje	X o Y o Z

3D View (Vista)

Vista Superior (Top)	7 (T. Numérico)
Vista Frontal (Front)	1 (T. Numérico)
Vista Lateral (Side)	3 (T. Numérico)
Vista Cámara	0 (T. Numérico)
Ver Seleccionado	. (T. Numérico)
Vista Quad (Act/Des)	Ctrl Alt Q
Perspectiva/Ortogonal	5 (T. Numérico)

Si utilizas un portátil...

Si usas un ordenador portátil, en muchas ocasiones el teclado numérico no es fácilmente accesible. Puedes emularlo en el menú ⊳ **File/ User Preferences/ Input** y activando la opción **Emulate Numpad** (ver figura).



3D View (Modos)

Modo Edición/Objeto [Tab]

3D View (Otros)

Maximizar Editor [Ctrl] [↑]
Duplicar [⇧] [D]
Mover a Capa [M]
Establecer Cámara [Ctrl] [0]

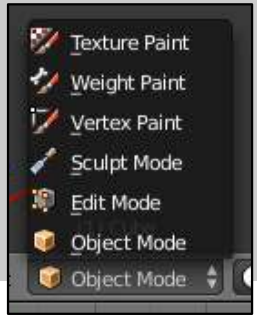
El Sistema de Capas

Blender, igual que los programas de diseño 2D como *Photoshop* o *Gimp* utiliza capas para organizar los objetos. En la cabecera del Editor 3D hay 20 «cuadritos». Cada cuadrito representa una capa. Si el cuadro tiene un círculo dentro, indica que esa capa tiene algún objeto. Las capas activas aparecen sombreadas. Puedes activar o desactivar capas pulsando sobre ellas con [⇧] y [🖱].

Logic Editor (Game)

Jugar (Play) [P]
Salir del Juego [Esc]
Eliminar Enlace [Ctrl] [🖱]
Añadir Bloque Lógico [⇧] [A]

Dependiendo del tipo de objeto, existen diferentes modos. Los más utilizados son el Modo Objeto, donde las transfor maciones se aplican a todo el objeto (globalmente) y el Modo Edición que se emplea para modificar los elementos que forman el objeto (vértices, aristas y caras principalmente).



Sobre Modos...

Otras Órdenes Útiles

Añadir Objetos [⇧] [A]
Emparentar Objetos [Ctrl] [P]
Para emparentar objetos, primero seleccionar el objeto hijo y después, con [⇧] pulsado, seleccionar el objeto padre. Pulsar [Ctrl] [P]
Eliminar Parentesco [Alt] [P]
Añadir Frame Clave [I]
Reproducir Animación [Alt] [A]
Aplicar Transformación [Ctrl] [A]

Buscador de Herramientas

Si no te acuerdas del atajo de teclado de una determinada orden, o no recuerdas la opción del menú donde se activa, no te preocupes, Blender cuenta con un po tente Buscador de Herramientas (*Tool Browser*). Pulsando [ ] (Barra Espa ciadora), accederemos a un menú de búsqueda interactiva. Bastará con teclear algunas letras del comando que queremos ejecutar y en la lista aparecerán todas las órdenes que contienen esa palabra clave. ¡Fácil y rápido!

Moviéndonos en la Tercera Dimensión

A continuación veremos algunos de los comandos esenciales para movernos en el espacio 3D.

Para rotar el punto de vista de una ventana 3D, pinchamos y arrastramos con . Para hacer zoom sobre una ventana 3D podemos utilizar la rueda central del ratón (sin pinchar). Para desplazar el punto de vista (tanto horizontal como verticalmente) emplearemos + .

En muchas ocasiones, tras realizar varias operaciones respecto del punto de vista no encontraremos los objetos de la escena. Al principio puede resultar difícil volver a situar los objetos dentro de la ventana. En estas ocasiones podemos colocar todo de nuevo simplemente pulsando la tecla Home (*Inicio*). Esto es válido para multitud de tipos de ventanas de Blender (entre ellas, la de Vista 3D).

Cambiaremos el modo de vista de cada vista 3D mediante atajos de teclado o los botones destinados a tal efecto. Situándonos con el puntero del ratón en el interior de alguna ventana de tipo *3D View* y, pulsando 5 en el teclado numérico, cambiaremos la vista entre proyección ortográfica (que proyecta paralelamente los vértices sobre la cámara) y perspectiva (el tamaño relativo del objeto disminuye cuando aumenta la distancia).

Figura 10 Acceso a las vistas desde el menú *View* en el *3D View Editor*.

Con la tecla 0 del teclado numérico tendremos la vista de la cámara. 1 , 3 y 7 nos darán las vistas de la escena en planta, alzado, y perfil respectivamente.

En las dos páginas anteriores se resumen los principales atajos de teclado en Blender. Resulta especialmente interesante la nota sobre cómo emular el teclado numérico si estás trabajando con un ordenador portátil que no tenga teclado numérico a la derecha.

Contexto y atajos de teclado

Blender es sensible al contexto sobre el que se realizan los atajos de teclado. Esto quiere decir que el atajo de teclado que se ejecuta sobre la ventana que contenga el puntero del ratón. No es necesario pinchar con el ratón, únicamente posicionar el puntero sobre la ventana a la que queremos enviar el atajo de teclado.

Tras esta brevísima introducción al interfaz de usuario de Blender, ya estamos en condiciones de empezar con la primera práctica. Al principio puede abrumarte la cantidad de atajos de teclado y comandos a recordar. No te preocupes, tras un poco de entrenamiento lo conseguirás sin problema. Los principales atajos para posicionar objetos en el espacio 3D y mover la vista 3D los dominarás en minutos. Deberás revisar estas páginas si no recuerdas cómo realizar algún desplazamiento en el punto de vista que te haga falta para realizar una determinada acción.

Creando una escena sencilla

En esta primera práctica veremos un ejemplo sencillo de trabajo con algunas de las principales etapas de trabajo en gráficos 3D. Comenzaremos personalizando las vistas de la ventana 3D en cuatro divisiones, como se muestra en la Figura 11. Aunque esta división estándar puede realizarse cómodamente desde el menú ▸ **View/ Toggle Quad View** en la cabecera de la Vista 3D, es recomendable al principio practicar la división manual de las ventanas de trabajo en Blender.

A partir de la versión 2.5 de Blender, las ventanas pueden tener una serie de paneles con herramientas e información adicional. Estos paneles pueden ocultarse arrastrando el borde con 🖱. Pueden mostrarse de nuevo pinchando sobre el icono (+) que estará situado en las esquinas de la ventana. Por ejemplo, la Figura 1.28 muestra cómo puede ocultarse la cabecera de una ventana (el icono (+) se situa en este caso en la esquina inferior derecha). De igual modo, la pestaña de transformación puede mostrarse u ocultarse con la tecla N. El panel de herramientas se oculta o se muestra de forma análoga con la tecla T.

La posición de estos iconos de ocultación de elementos puede variar según el tipo de ventana (como en el caso de las cabeceras de las ventanas).

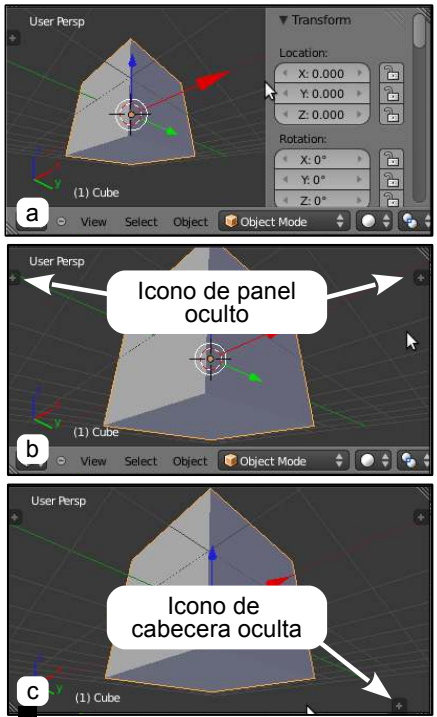


Figura 12 Arrastrando el borde de los paneles y cabeceras pueden ocultarse, como se muestra en (b) y (c). En ese caso aparecerá un icono que puede pincharse para que vuelvan a ser visibles.

Para comenzar esta práctica, primero seleccionaremos el cubo que crea Blender por defecto. Para seleccionar un objeto de la escena pinchamos sobre él con 🖱. El objeto seleccionado aparecerá con el borde de color color naranja. Vamos a eliminarlo de la escena pulsando Del (*Supr*), y aceptamos el mensaje *OK? Delete*. Añadiremos ahora una primitiva de Blender; por ejemplo, la cabeza de *Suzanne*. Para insertar nuevos elementos a la escena, pulsaremos ⇧ A y elegiremos ▸ **Add/ Mesh/ Monkey**. Podemos acceder al menú de añadir nuevos objetos a la escena igualmente en la cabecera de la ventana *Info*, en la entrada *Add*. Como es una opción que utilizaremos muy frecuentemente es recomendable recordar el atajo ⇧ A.

Dónde se añaden los nuevos objetos...

Recordemos que mediante 🖱 podemos seleccionar los objetos. Para seleccionar un objeto, deberemos estar en *Modo Objeto*. Pulsando 🖱 cambiamos la posición del puntero 3D (el círculo de color rojo y blanco que nos define dónde se añadirán nuevos objetos a la escena). Pulsando y arrastrando 🖱 el rotaremos el punto de vista en la ventana 3D.

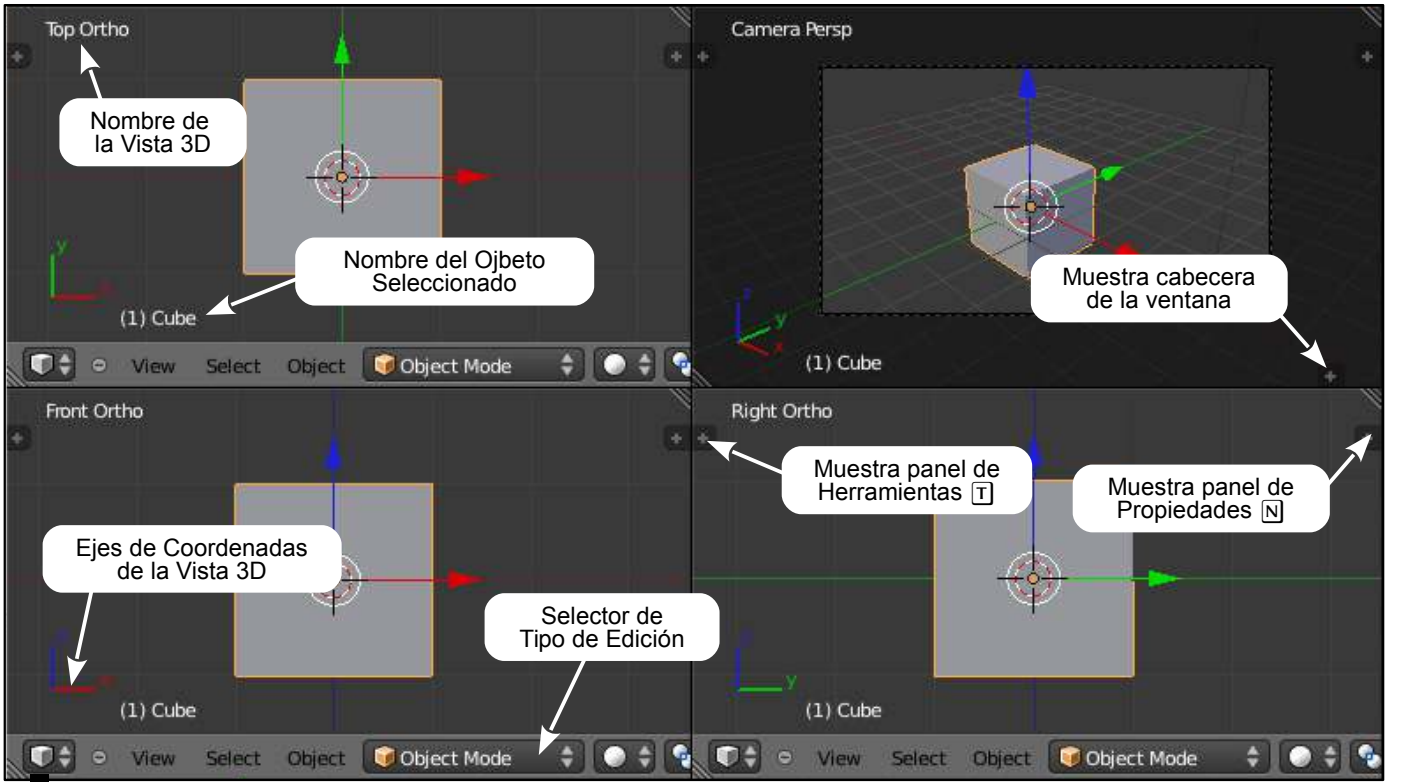


Figura 11 Personalización de vistas 3D y elementos comunes de la interfaz.

III Transformaciones Rígidas

Veamos ahora algunas operaciones básicas que podemos realizar sobre los objetos. Las operaciones son dependientes del punto de vista sobre el que estamos trabajando. Es decir, si rotamos un objeto situando el puntero del ratón encima de la vista que muestra el alzado, rotaremos el objeto sobre el eje *Y* por defecto, mientras que si lo rotamos con el puntero del ratón encima de la vista correspondiente a la planta, lo rotaremos sobre el eje Z. En la parte inferior izquierda de cada ven tana 3D aparece un pequeño sistema de coordenas, indicando los ejes cartesianos (ver Figura 11).

- **Rotación.** Con el objeto seleccionado, pulsamos y soltamos R (del inglés *Rotation*). No hay que mantener la tecla pulsada; simplemente se pulsa y se suelta, y acto seguido desplazamos el ratón.
- **Desplazamiento.** Procedemos igual que con la rotación, pero usando la tecla G (del inglés, *Grab*).
- **Escalado.** Igual que los anteriores, pero con la tecla S (*Scale*).

Es conveniente aprender los atajos de teclado anteriores, ya que estas transformaciones se realizan con mucha frecuencia. Blender también dispone de unos controles visuales para realizar las tres operaciones anteriores.

Figura 13 Uso de manejadores. En este caso se utiliza el manejador de rotación.

En la cabecera de la ventana 3D, aparece un el icono de un pequeño sistema de coordenadas (ver Figura 13), con el que es posible mostrar u ocultar estos manejadores. Si no aparece el icono en la cabecera de la ventana 3D es porque estará oculto (probablemente a la derecha). Puedes desplazar la cabecera de la ventana 3D, pinchando sobre ella y arrastrando con . Existe un botón para cada transformación: Desplazamiento, Rotación o Escalado. En estos manejadores, la operación se puede realizar desde el sistema de coordenadas global, local o desde el punto de vista, realizando la elección mediante la lista desplegable «*Global*» de su derecha.

Las operaciones anteriores, realizadas mediante atajos de teclado o manejadores, permiten utilizar tres modificadores. Estos modificadores se activan después de haber indicado la operación a realizar.

- **Limitación de Eje.** Cualquiera de las tres operaciones anteriores (realizada mediante atajos de teclado) pueden limitar su eje de aplicación. Con la operación seleccionada (por ejemplo, R para rotar), si pulsamos a continuación alguna de las teclas X, Y, Z limitamos la operación a realizar sobre ese eje. Si queremos que la transformación se realice según el sistema de referencia local del objeto, simplemente se pulsará 2 veces el eje (por ejemplo, X X realiza la rotación sobre el eje X local).

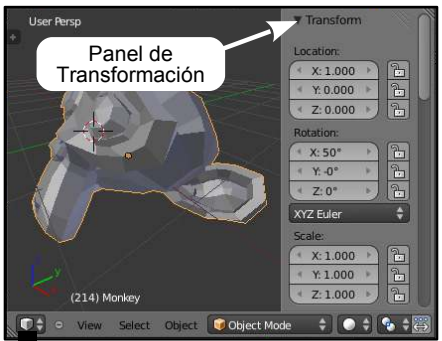


Figura 14 Uso del panel de Transformación para establecer un valor preciso.

- **Modo de Precisión.** Si mantenemos la tecla ⇧ (*Mayúsculas;* no confundir con *Bloq. Mayús.*) pulsada mientras realizamos alguna de las tres operaciones básicas, la ejecutaremos en modo de precisión. Si por el contrario, mantenemos la tecla Ctrl pulsada, lo haremos ajustándonos a un número de unidades dependiente de la operación; si estamos rotando lo haremos de 5 en 5 grados, si estamos desplazando o escalando, lo haremos ajustándonos a la rejilla. La información sobre el valor numérico que estamos aplicando, aparece en la cabecera de la ventana 3D.
- **Valor numérico exacto.** Se puede indicar igualmente un valor numérico; tras pulsar la tecla para la operación (por ejemplo R), podemos teclear directamente un valor numérico (90) y el objeto rotará 90 grados.

Se puede consultar y modificar el valor de las transformaciones anteriores en la pestaña de *Transformación* (ver Figura 14). Recordemos que esta pestaña también puede ocultarse o mostrarse con N. Los valores numéricos aparecen en botones delimitados por dos flechas. Este tipo de controles permiten introducir valores pinchando con y tecleando directamente el valor deseado.

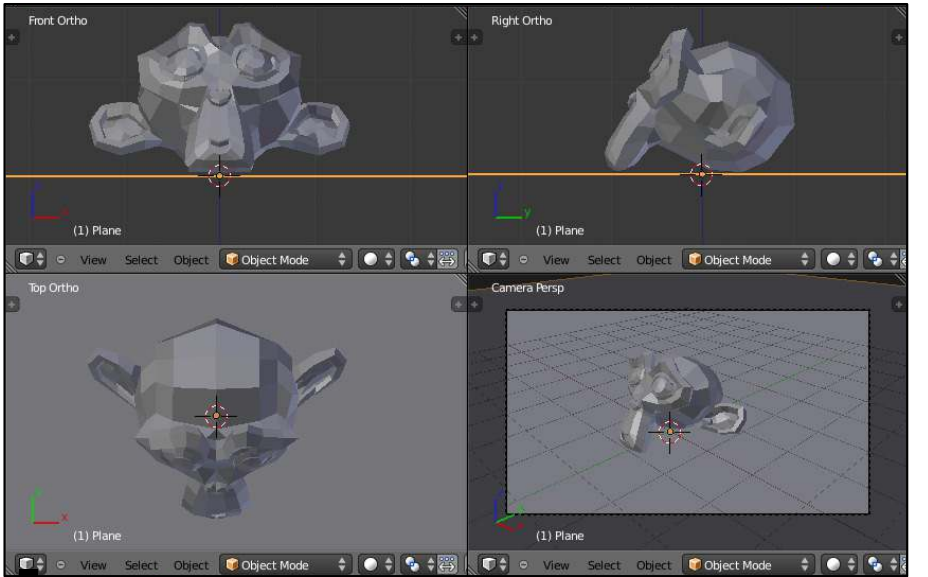


Figura 15 Situación del plano bajo la monita Suzanne y configuración de las vistas.

Para acabar este primer ejercicio, creamos una escena muy sencilla con un plano que sirva de "suelo" para Suzanne. Ayudándonos de las vistas de planta, alzado y perfil, situaremos el puntero 3D debajo de la cabeza del mono. Recordemos que la situación del puntero 3D se cambia pinchando con sobre la vista 3D.

Una vez situado correctamente el puntero, añadiremos un plano a la escena que servirá de suelo al objeto anterior. Para ello, pulsaremos ⇧ A ▷ **Add/ Mesh/ Plane**. Haremos el plano más grande (mediante S) para que cubra el suelo de la escena. Deberemos obtener una escena similar a la Figura 15.

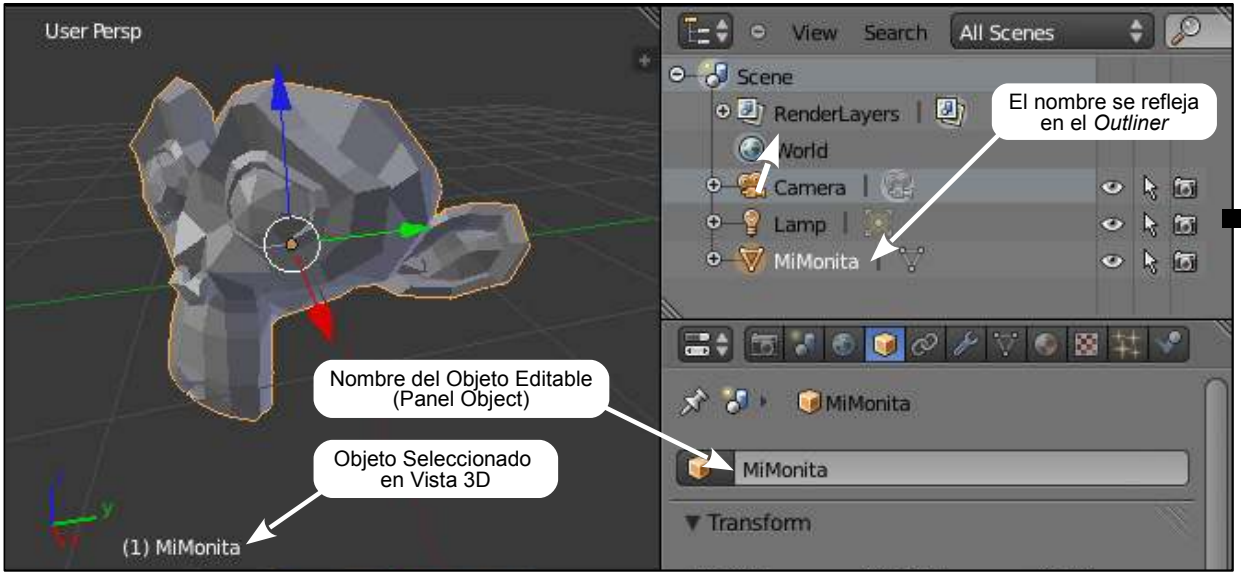


Figura 16

Nombrado de objetos. Al cambiar el nombre del objeto, aparecerá reflejado tanto en el *Outliner* como en la *Vista 3D*.

Nombrado de Objetos

Antes de pasar a la acción y crear nuestro primer videojuego, necesitamos conocer algunas utilidades básicas para organizar adecuadamente los objetos en la escena.

Cuando construimos una escena compleja, el número de objetos aumenta y es buena idea mantener una organización adecuada. Es conveniente renombrar los objetos que se van creando, para facilitar su posterior selección, importación, etc.

Blender requiere que cada objeto tenga un nombre único, ya que internamente emplea el campo del nombre como clave para referirse a él. Para renombrar un objeto, primero lo seleccionamos con y en la ventana de propiedades , el panel de objeto tiene un campo (por defecto, encima del grupo de propiedades Transform) para indicar el nombre del objeto (ver Figura 16). En esta Figura también aparece la herramienta Outliner, muy útil para seleccionar los objetos y editarlos.

A la derecha de cada objeto en el Outliner aparecen tres iconos que permiten:

- **Ojo** . Si está activo el objeto es visible en la escena. Sirve para ocultar temporalmente objetos.
- **Puntero** . Permite proteger objetos de ser seleccionados en la ventana 3D. Funciona como la típica herramienta de «candado» en software de diseño.
- **Cámara** . El objeto se tendrá en cuenta en la etapa de Render, para generar una imagen 2D final.

Figura 17 Diferentes elementos de la interfaz para el trabajo con capas.

Capas

Muchas aplicaciones de diseño permiten utilizar Capas (*Layers*) y disponer los objetos en ellas. Blender dispone de un cómodo sistema de 20 capas, que pueden ser activadas u ocultadas desde la cabecera de las ventanas 3D (ver Figura 17). A diferencia de otros comandos (como modificación de propiedades de sombreado en una ventana 3D), la activación de las capas es global a todas las vistas. Esto implica que si activamos una capa, los objetos visibles en todas las vistas serán aquellos que estén situados en esa capa. Los botones de acceso a las capas sólo están visibles en *Modo de Objeto*.

Si has utilizado alguna herramienta de diseño como *Gimp* o *Photoshop*, ya conoces la metáfora de las capas en interfaces de usuario: la escena final se compone de los elementos (objetos, cámaras, luces, etc...) que se encuentran en todas las capas que estén activas. Si una capa contiene ciertos objetos pero no está activa, la etapa de representación los ignorará. El uso de las capas es muy intersante para trabajar con escenas complejas. Es habitual tener ciertas capas para el escenario de una escena, otra capa para los personajes, otra para las luces...

Para activar una capa simplemente pincharemos sobre ella con .

Por defecto, al activar una capa, se desactivan el resto de capas. Podemos tener activas varias capas a la vez si mantenemos pulsado elegimos cada una de las capas con . Para mover un objeto a otra capa primero lo seleccionamos y después pulsamos M . Nos aparecerá una ventana (ver Figura 17) donde seleccionar la capa a la que se quiere mover el objeto. Las capas que contienen algún elemento aparecen dibujadas con un pequeño punto en el centro. Si tenemos objetos en varias capas, podemos saber en qué capa está un objeto porque al seleccionarlo se pinta de color naranja el punto central de la capa en la que está.

Illumina - Blender Game Engine
Low Poly Girl (Creative
Commons BY-NC by Rogper)

Capítulo 3
Blender Game Engine

En el capítulo anterior se estudiaron las órdenes esenciales en Blender para trabajar con objetos 3D y movernos en el Editor de Vista 3D. En este capítulo realizaremos el primer prototipo de videojuego empleando los Bloques Lógicos. Esta potente aproximación al desarrollo permite construir, sin necesidad de escribir ni una línea de código, aplicaciones relativamente complejas.

Partiendo de elementos previamente modelados, construiremos un videojuego de naves espaciales. Se dotará a las naves enemigas de un sencillo comportamiento, y finalmente se genenará un ejecutable para nuestra plataforma de trabajo. ¡Ánimo, que comenzamos con la parte más divertida de este manual!

Introducción

Como se ha visto en capítulos anteriores, Blender trae integrado un potente motor de videojuegos que le permite crear aplicaciones interactivas en 3D... ¡y generar ejecutables independientes de Blender!. El motor de juego de Blender (*BGE*) es una potente herramienta de alto nivel de programación. Su principal objetivo es el desarrollo del juego, pero se puede utilizar para crear cualquier software interactivo en 3D con otros fines, tales como visitas interactivas de artquitectura en 3D, libros electrónicos, etc.

El núcleo de la estructura del *BGE* son los bloques lógicos (*Logic Bricks*). El objetivo de los bloques lógicos es ofrecer un entorno visual fácil para diseñar aplicaciones interactivas sin necesidad de conocer ningún lenguaje de programación. Sobre cada objeto de juego se pueden definir tres tres tipos de bloques lógicos, *Sensores*, *Controladores y Actuadores*, además de *Propiedades* o *Atributos* (ver Figura 1).

La lógica en el motor de juegos de Blender está ligada a los Objetos de Juego (o *Game Objects*). Los objetos tienen un nombre único, y pueden ser desde objetos geométricos (como una nave espacial o un obstáculo), una cámara, una lámpara o un objeto sin representación gráfica (o *Empty*). Los objetos está listados por sus nombres y aparecen el la ventana de lógica cuando son seleccionados. Los bloques lógicos y propiedades de un objeto solo es visible cuando el objeto está seleccionado.

Propiedades

Las propiedades son como las variables en los lenguajes de programación. Sirven para guardar datos asociados a un objeto, como su puntuación, el número de vidas que le quedan, los tipos de armas que tiene, etc. En el caso de objetos que representen a una persona, las propiedades podrían ser su nombre, su edad, su altura, peso, número de DNI, etc.

Blender permite trabajar con los siguientes tipos de Propiedades:

- ***Timer***: Es un temporizador que sirve para contar siempre de forma creciente, desde el valor en el que se crea.
- ***Float***: Contiene números decimales en el rango de -10000.000 a 10000.000.
- ***Integer***: Números enteros entre -10000 y 10000.
- ***String***: Almacena un texto de hasta 128 caracteres.
- ***Boolean***: Valor lógico True/False.

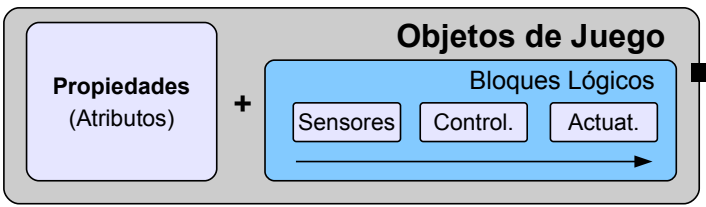


Figura 1

Diagrama general de los Objetos de Juego (Game Objects), bloques lógicos y propiedades.

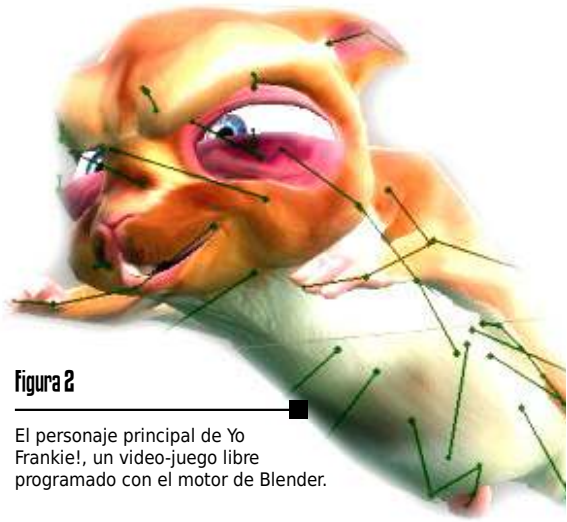


Figura 2

El personaje principal de Yo Frankie!, un video-juego libre programado con el motor de Blender.

Sensores, Controladores y Actuadores

Mediante estos tres tipos de bloques lógicos es posible definir el comportamiento de la aplicación.

Los ***Sensores*** comienzan todas las acciones lógicas. El sensor indica cosas como la cercanía de un objeto, la pulsación de una tecla, eventos programados, etc. Cuando un sensor es activado, un pulso es enviado a todos los controladores que estén conectados con él.

Los ***Controladores*** manipulan la lógica, evaluando los pulsos de los sensores y activando (si se cumple la condición que definen) los actuadores en respuesta.

Los ***Actuadores*** afectan a los objetos o al juego de alguna manera. Los actuadores cambian movimiento, sonido, propiedades, objetos, etc. Estos cambios pueden provocar eventos que sean capturados por sensores de otros objetos.

Referencia de Bloques Lógicos

En el Capítulo 4 puedes encontrar una descripción de los principales Sensores, Controladores y Actuadores disponibles en Blender. En este capítulo utilizaremos únicamente los necesarios para realizar los ejemplos... ¡Anímate y prueba otras combinaciones!

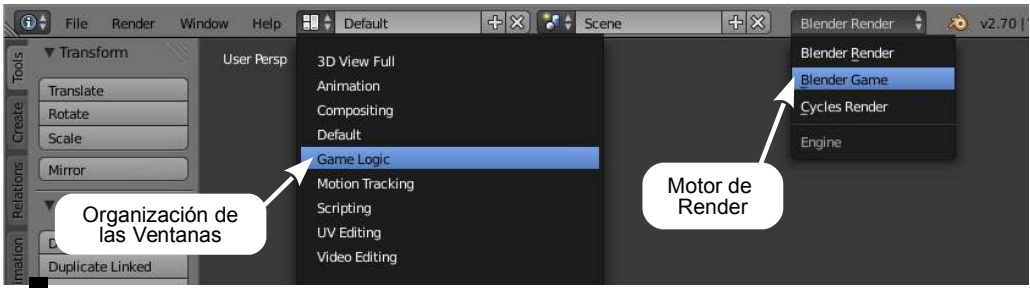


Figura 3 Selección del motor de Render y configuración de las ventanas.

Nuestro Primer Ejemplo

El primer paso para utilizar el motor de videojuegos de Blender es activarlo en el menú principal de la aplicación. El motor de render por defecto está en «*Blender Render*». Es necesario cambiarlo a «***Blender Game***» (ver Figura 3). Si no cambias el motor de Render, algunas opciones no aparecerán en el interfaz de usuario.

De igual modo, es recomendable cambiar la configuración de las ventanas a ***Game Logic*** (ver Figura 3). Como vimos en el capítulo anterior, esta configuración puedes personalizarla como más te guste, o hacerla de forma totalmente manual. De cualquier forma, ahora trabajaremos con esa configuración estándar porque con esas ventanas tendremos todo lo necesario para hacer nuestros primeros ejemplos.

Una vez elegida esa configuración de ventanas, tendremos una disposición como se muestra en la Figura 4. Además de la Vista 3D y el *Outliner* que fueron explicados en el capítulo anterior, existe una zona a la derecha para la edición de texto. Este editor permite programar el comportamiento de los objetos mediante código *Python*. En este curso no vamos a estudiar esta capacidad de Blender, por lo que directamente puedes eliminar esa zona de la pantalla (unir el editor con otro) o ignorarlo. Debajo del editor de texto está el editor de ***Propiedades*** (*Properties*), que permite definir propiedades de los objetos, como el nombre, propiedades físicas, etc...

Finalmente, el bloque principal del ***Editor de Lógica*** ocupa la parte inferior central e izquierda de la ventana, y permite añadir propiedades y bloques lógicos a los objetos de juego.

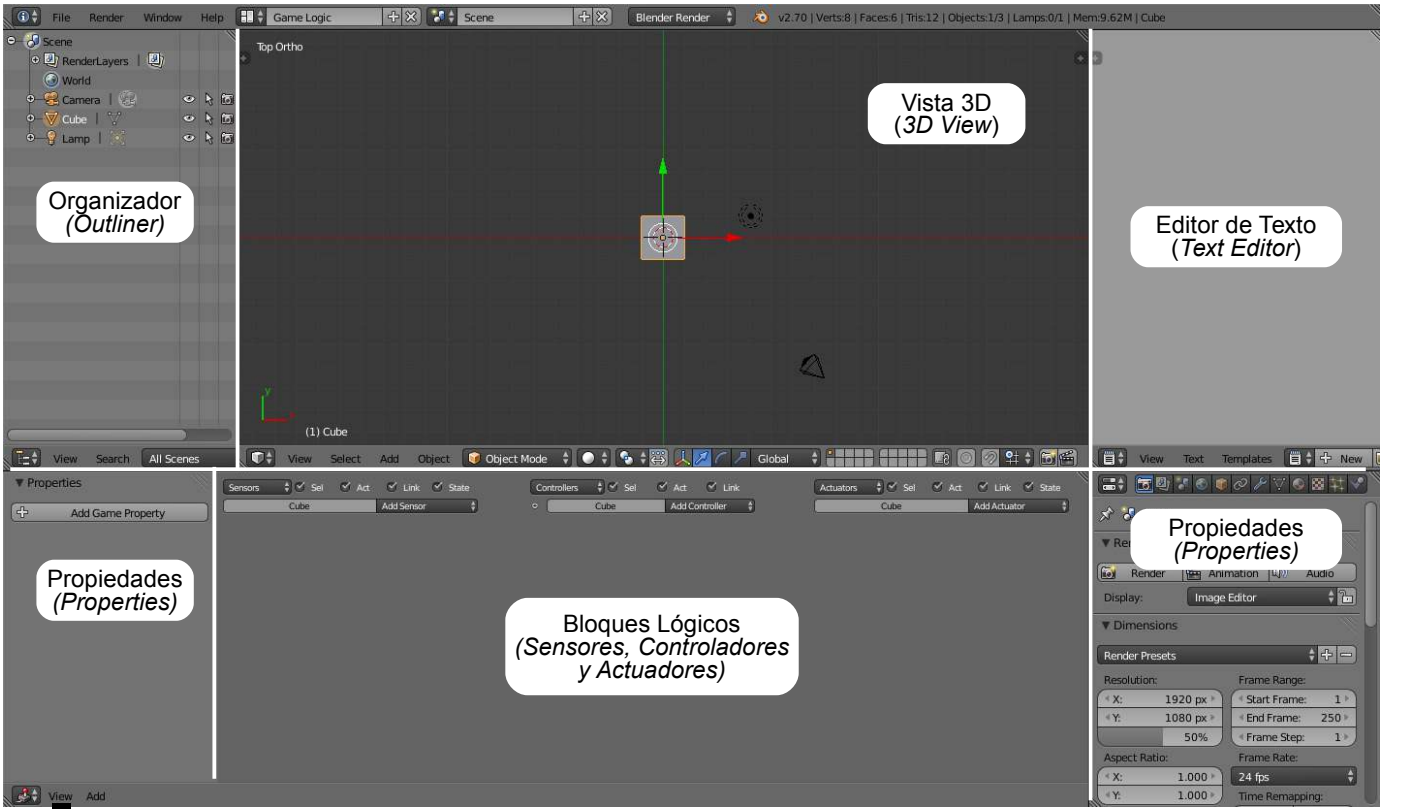


Figura 4 Ventanas (editores) existentes en la configuración por defecto de *Game Logic*.

Un Cubo que Rota... ¡Wow!

Veamos cómo funciona el editor de lógica con un ejemplo muy básico. Vamos a hacer que el objeto Cube, creado por defecto en la escena, rote con respecto de su eje X positivo cuando pulsamos la tecla A.

Para ello, con el *Cubo* seleccionado, añadimos un Sensor de tipo ***Keyboard*** (pinchando en *Add Sensor*). Definimos que la tecla es la A (parámetro ***Key***). Creamos un controlador de tipo AND y lo conectamos, de forma que cuando se pulse la tecla A, se transmita el pulso al controlador. Al ser un controlador AND con una única entrada, activará la salida siempre que la entrada esté activa. Finalizamos conectando un Actuador de tipo ***Motion***, que rote 1º con respecto del sistema de referencia local (el botón marcado con la letra ***L*** debe estar activado, como se muestra en la Figura 5).

Hecho esto, activamos el punto de vista de la cámara en la Vista 3D (pulsando 0 en el teclado numérico o desde la cabecera del 3D View en ⊳ **View/ Camera**). La vista debe ser algo similar a la Figura 6. Finalmente, para ejecutar la aplicación, pulsamos la tecla P (de *Play*). Ahora, cuando pulsemos la tecla A, el cubo debe rotar sobre su eje local X. Fascinante. Para parar el modo de juego y seguir trabajando en Blender, pulsaremos la tecla Esc.

Generando un Ejecutable

Si estás contento con este primer resultado, veamos cómo generar un ejecutable independiente de Blender. Sí, en cualquier momento del desarrollo puedes generar un ejecutable para los «usuarios» de tu aplicación que no necesitarán tener Blender instalado en su equipo. Para ello, debes activar un plugin de exportación.

En el menú principal, ⊳ **File/ User Preferences**, en la sección Addons, elige la categoría **Game Engine** y activa el único plugin de esa categoría «***Game Engine: Save As Game Engine Runtime***» (ver Figura 7). Hecho esto, puedes generar el ejecutable fácilmente en ⊳ **File/ Export/ Save As Game Engine Runtime**.

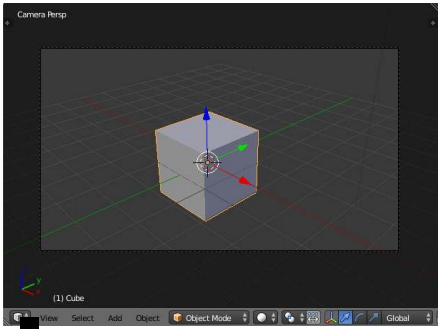


Figura 6 Vista 3D con la cámara activa.

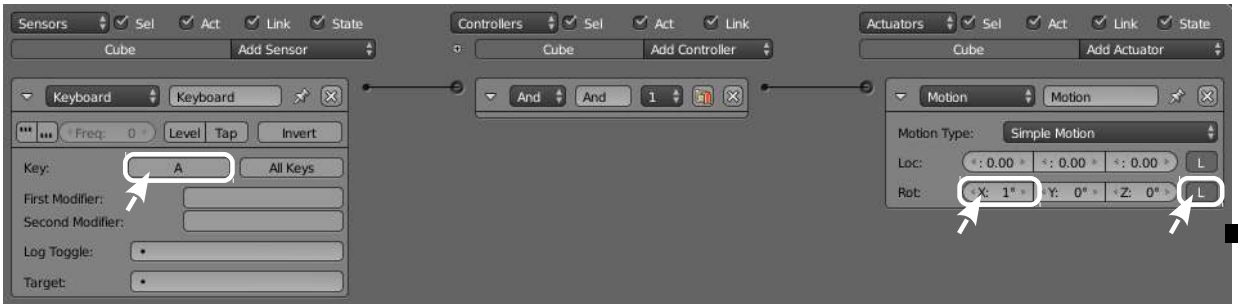


Figura 5

Bloques lógicos sobre el objeto "*Cube*"

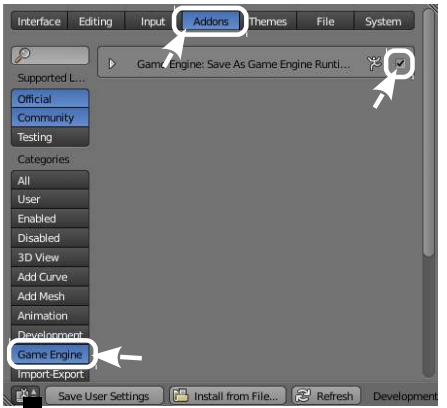


Figura 7 Activación del PlugIn para generar ejecutables desde Blender.

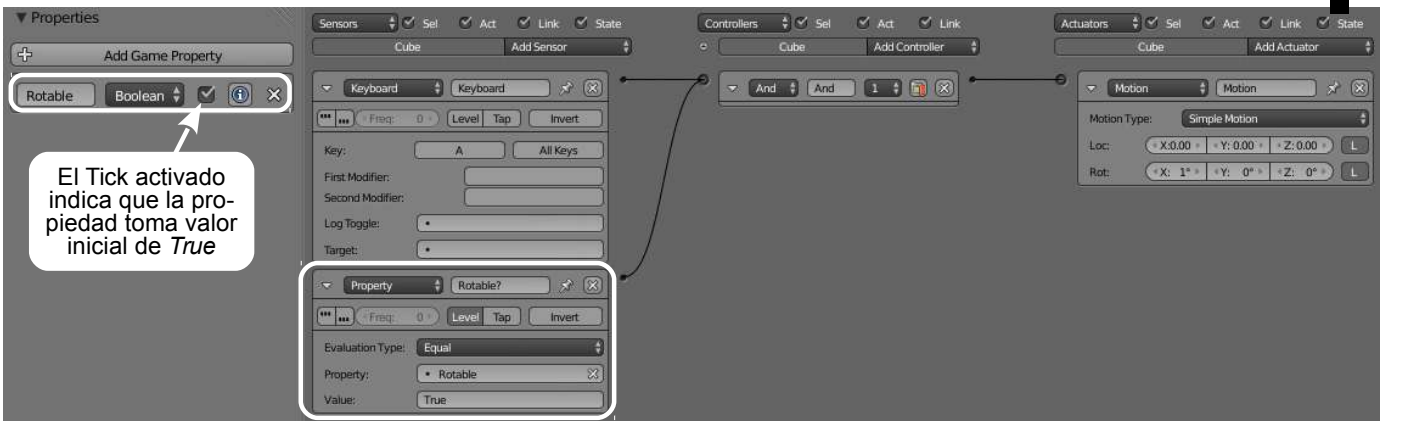
III Mejorando el Cubo Rotador

Ahora que sabemos cómo generar ejecutables de nuestros juegos, veamos cómo mejorar el ejemplo básico anterior añadiendo algo más de complejidad a los bloques lógicos.

Ahora vamos a añadir una propiedad al objeto de tipo *Boolean*, que tomará valores lógicos de Verdad o Falso (*True* o *False*). Esa propiedad la llamaremos ***Rotable***, y crearemos un nuevo sensor de tipo *Property* que preguntará por el valor de esa propiedad y lo conectamos al controlador anterior como se muestra en la Figura 8.

Con esta nueva disposición de bloques lógicos, el objeto podrá rotar o no dependiendo del valor inicial de la propiedad *Rotable*. Si lo inicializamos a verdadero (como es muestra en la Figura 8), el objeto podrá rotar. Sin embargo, si la propiedad la inicializamos a *False* (es decir, quitamos el *Tick* en la definición de la propiedad), el objeto nunca podrá rotar porque no se cumple la condición del controlador AND. Podemos «*leer*» el comportamiento de este controlador como «*Si la tecla A está pulsada Y la propiedad Rotable vale True, entonces ejecuto el actuador que tengo conectado a continuación*».

Segunda versión del ejemplo, con propiedad "Rotable" **Figura 8**



A partir de esta nueva versión podemos complicar un poquito el ejemplo, añadiendo un temporizador. Creamos una nueva propiedad *Timer* que llamaremos ***Clock*** (reloj), inicializada a 0. Las propiedades de este tipo van incrementando su valor como un reloj, indicando el número de segundos transcurridos desde el valor inicial que toman.

En la nueva versión del ejemplo (ver Figura 9) hemos creado el reloj y tres bloques lógicos nuevos de forma que cuando el reloj vale más de 5 (han transcurrido más de 5 segundos), la propiedad ***Rotable*** se establece a *False*, de modo que el objeto no podrá rotar.

Como se muestra en la Figura 9, es conveniente organizar los bloques lógicos asignándoles un nombre. De igual forma, con el triángulo situado en la esquina superior izquierda de cada bloque se puede «plegar» su representación, de forma que únicamente tengamos maximizados los bloques con los que estemos trabajando.

III Depurando la Aplicación

En programación es muy útil ver el valor que toman las propiedades de los objetos. Esto permite detectar errores (*bugs*) de nuestros programas. El editor de Blender permite mostrar el valor de las propiedades sobre la ventana del juego.

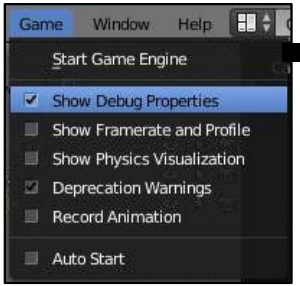


Figura 10 Activación de las opciones de depuración (debug) en el menú principal de Blender.

En el menú principal de Blender, debemos activar ⊳ **Game/ Show Debug Properties** (ver Figura 10). Debemos igualmente activar el botón de depuración de cada propiedad de la que queramos ver su valor (Figura 11).

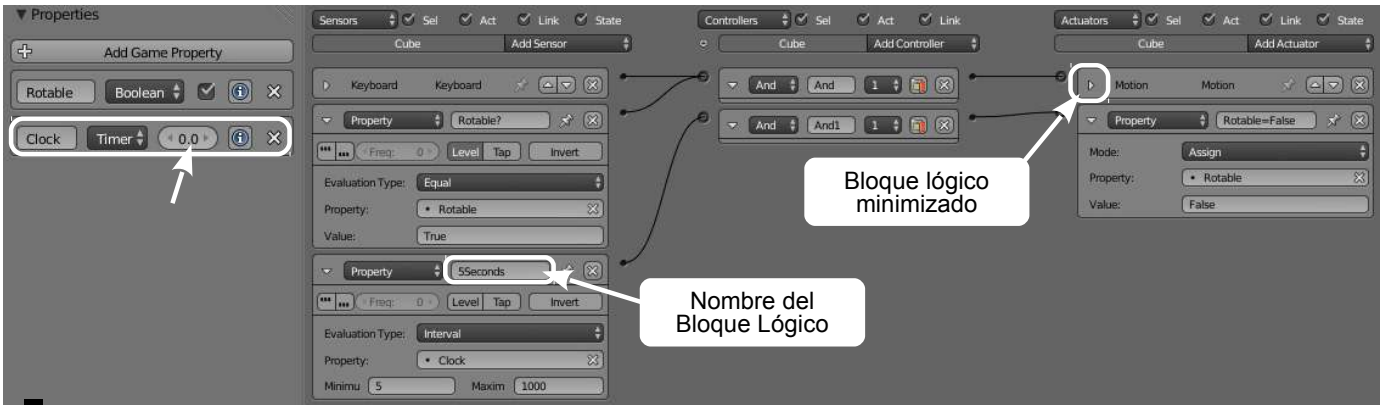


Figura 9 Añadiendo un reloj a la aplicación...

Figura 11 Activando las opciones de depuración sobre las propiedades

Una vez activadas las opciones de depuración, la ventana de la vista 3D mostrará la información del valor de las propiedades sobreimpresa, como se muestra en la Figura 12. ¡No olvides desactivar las opciones de depuración cuando vayas a generar el ejecutable final de tu videojuego!

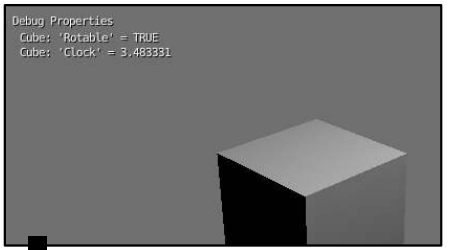


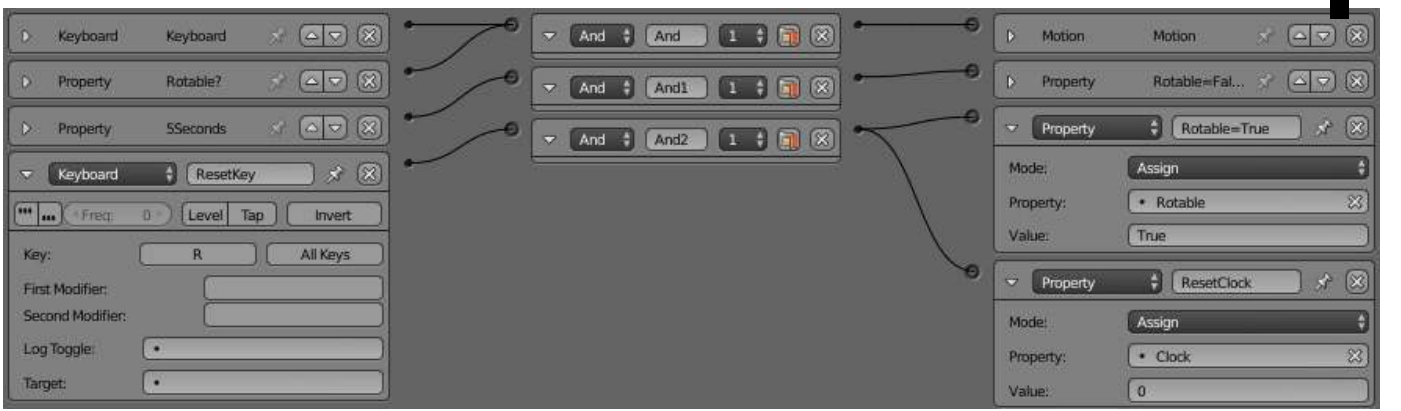
Figura 12 Debug en tiempo de ejecución

Supongamos que queremos habilitar una tecla adicional [R] de modo que, cuando se pulse tengamos otros 5 segundos en los que podamos rotar el objeto. ¿Qué tenemos que hacer? Añadir un nuevo sensor que active dos actuadores en este caso.

Un actuador establecerá la propiedad ***Rotable*** a *True*, y otro *reseteará* el reloj ***Clock*** a cero, para disponer de otros 5 segundos de rotación. El resultado de los bloques lógicos se muestra en la Figura 13.

Una vez que hemos entendido el funcionamiento de los bloques lógicos, vamos a *optimizar* el ejemplo utilizando controladores de tipo expresión. Se puede comparar el resultado que hemos construido hasta ahora completo en la Figura 14, frente al mismo resultado empleando expresiones en la Figura 15. La complejidad de la solución es mucho menor y mucho más fácilmente ampliable. El funcionamiento de la aplicación es exactamente igual.

Reset de tiempo... **Figura 13**



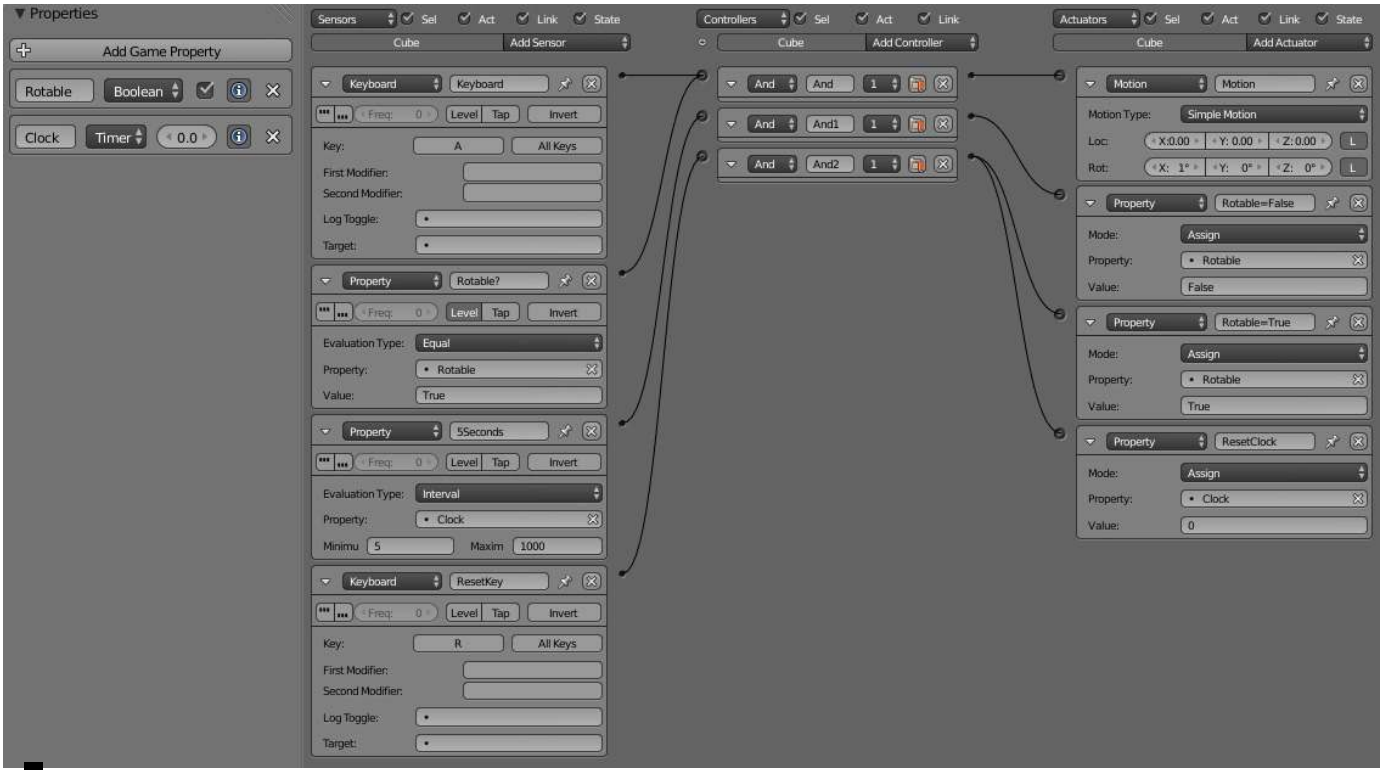


Figura 14 El ejemplo del cubo rotador con reloj completo, con todos los bloques lógicos expandidos

En las expresiones se pueden utilizar operadores matemáticos básicos, comparadores y operadores lógicos (ver Referencia en el Capítulo 4 de este manual). En estas expresiones se utilizan los nombres de las propiedades del objeto y el nombre de los sensores a los que está conectado el controlador. Es importante utilizar en los sensores nombres significativos. Así, en el ejemplo de la Figura 15 se ha renombrado el sensor de pulsación de la tecla *A* como ***AKey*** para utilizar ese nombre en la expresión del controlador.

Gracias al uso de esta expresión ya no es necesario el uso de la propiedad *Rotable*; sino que directamente podemos llamar al actuador *Motion* cuando la tecla *A* está pulsada y el valor del reloj es menor que 5. Vemos que el número propiedades se ha reducido a la mitad y el número de bloques lógicos de 11 a 6. En programación es habitual que existan múltiples caminos para resolver un mismo problema. Habitualmente la solución más sencilla suele ser la mejor. El uso de expresiones en este caso facilita enormemente la tarea.

Principio de diseño KISS

A modo de curiosidad, el principio KISS (del inglés *Keep It Simple, Stupid!*: «Mantenlo sencillo, ¡Estúpido!») es un acrónimo usado como principio de diseño. El principio KISS establece que la mayoría de sistemas funcionan mejor si se mantienen simples que si se hacen complejos; por ello, la simplicidad debe ser mantenida como un objetivo clave del diseño, y cualquier complejidad innecesaria debe ser evitada.

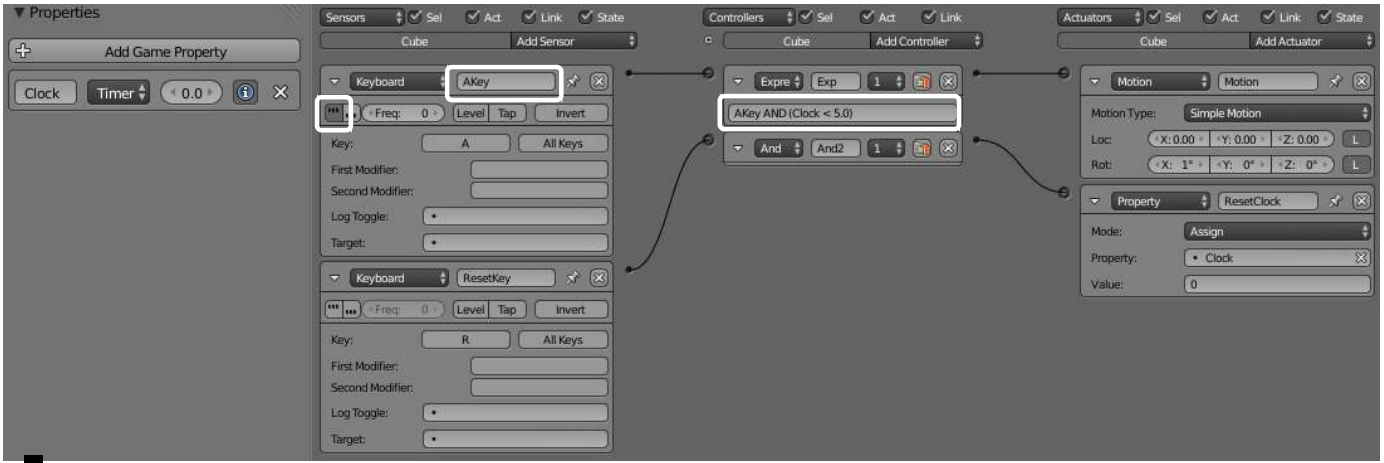


Figura 15 El ejemplo del cubo rotador con reloj completo empleando una única propiedad y una expresión como controlador.

Space Fighters

The Original Game :-P

III Introducción

A continuación desarrollaremos paso a paso un videojuego muy sencillo de batallas espaciales, poniendo en práctica los conceptos estudiados hasta ahora en el manual. Si tienes problemas para posicionar objetos en la escena o manejarte con las vistas 3D, es conveniente que repases el capítulo 2 donde se estudian los fundamentos del interfaz de usuario de Blender. De igual forma, también es recomendable que consultes el capítulo 4 con la referencia a los bloques lógicos soportados por Blender, porque en el este tutorial paso a paso no se explicará su uso. Finalmente, se asume que el lector ha trabajado con los conceptos explicados en las páginas anteriores de este capítulo, y es capaz de generar un ejecutable, activar el modo de debug, definir propiedades, cambiar el nombre a los bloques lógicos, etc...

Paso 1: Nave del Jugador

En este primer paso, comenzaremos cargando el modelo que servirá como nave del jugador. Comenzamos con la escena por defecto de Blender. Seleccionamos el cubo por defecto y lo borramos [Del].

Elegimos en el menú superior como motor de Render el Blender Game (ver Figura 16), y como configuración del entorno de trabajo Game Logic.

Ahora cargamos el modelo de la nave para el jugador. En el menú principal elegimos **File/ Append**, seleccionamos el archivo DarkFighter.blend y navegamos hasta la categoría de ***Object***, y seleccionamos el objeto llamado ***DarkFighter***. Pinchamos finalmente en el botón ***Link/Append from Library***.

Si activamos como modo de sombreado en la cabecera de la vista 3D «***Texture***», el resultado debería ser similar al mostrado en la Figura 16.

Si ahora vemos la escena desde el punto de vista de la cámara ([0] en el teclado numérico), tendremos una representación similar a la mostrada en la Figura 17. Vamos a cambiar la posición de la cámara para que vea la escena desde un punto de vista cenital. Lo más sencillo es establecer los valores numéricamente.

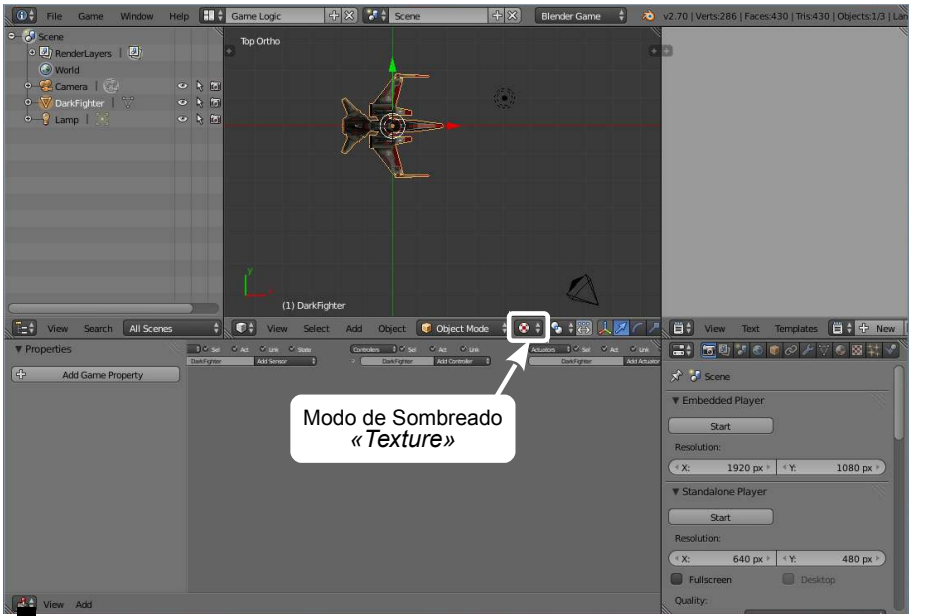


Figura 16 Tras importar el modelo del jugador principal, este debe ser el aspecto de la interfaz.

Con la cámara seleccionada, abrimos el panel de propiedades [N], y establecemos los valores como se muestra en la Figura 18 (posición y rotación a 0, salvo en el eje Z que «elevamos» la posición de la cámara 30 unidades para que vea la escena desde arriba).

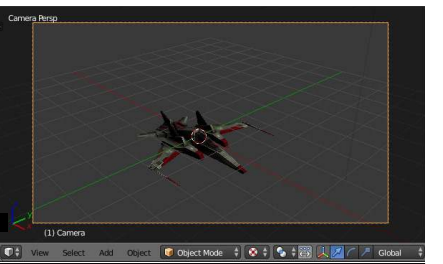


Figura 17

Vista desde la cámara

Ahora cambiaremos el nombre del objeto del jugador. En lugar de DarkFighter, lo llamaremos Player. De esta forma, el nombre es más representativo con vistas a crear los bloques lógicos. Con la nave seleccionada, en el menú de Propiedades (***Properties***), en la sección de ***Object*** cambiaremos el nombre a ***Player*** (ver Figura 19).

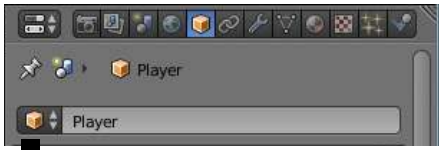


Figura 19 Cambio de nombre del objeto

Para finalizar con este primer paso, ajustaremos el tamaño del jugador a uno un poco menor (escalaremos con la tecla S) y su rotación (pulsando R, y seguidamente teclearemos -90). Nos aseguraremos que el objeto está posicionado en la posición 0 del eje Z (para que los enemigos que añadiremos a continuación estén perfectamente alineados en altura en el mismo eje). El resultado de este paso debe ser similar al mostrado en la Figura 19.

Cuando tengamos estos cambios realizados y la nave esté apuntando en el eje positivo de las Y, pulsaremos Ctrl A ***Apply Rotation & Scale*** para que el objeto aplique internamente esos cambios que hemos realizado.

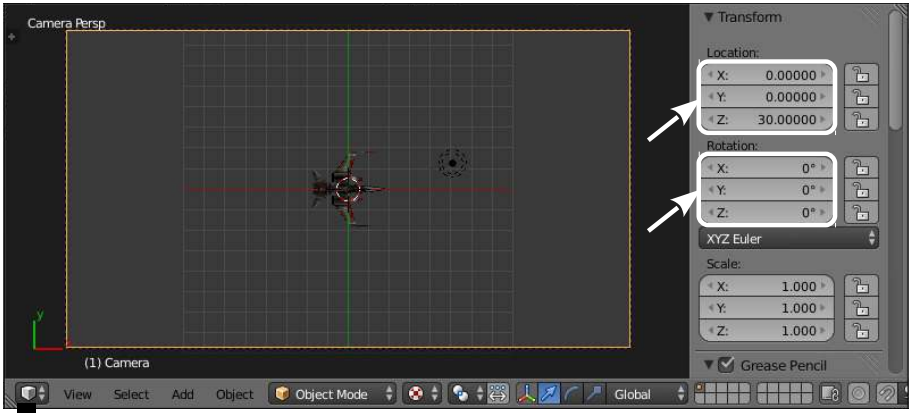


Figura 18 Situación de la cámara tras ajustar valores de Location y Rotation

Figura 19 Posicionando la nave del jugador adecuadamente

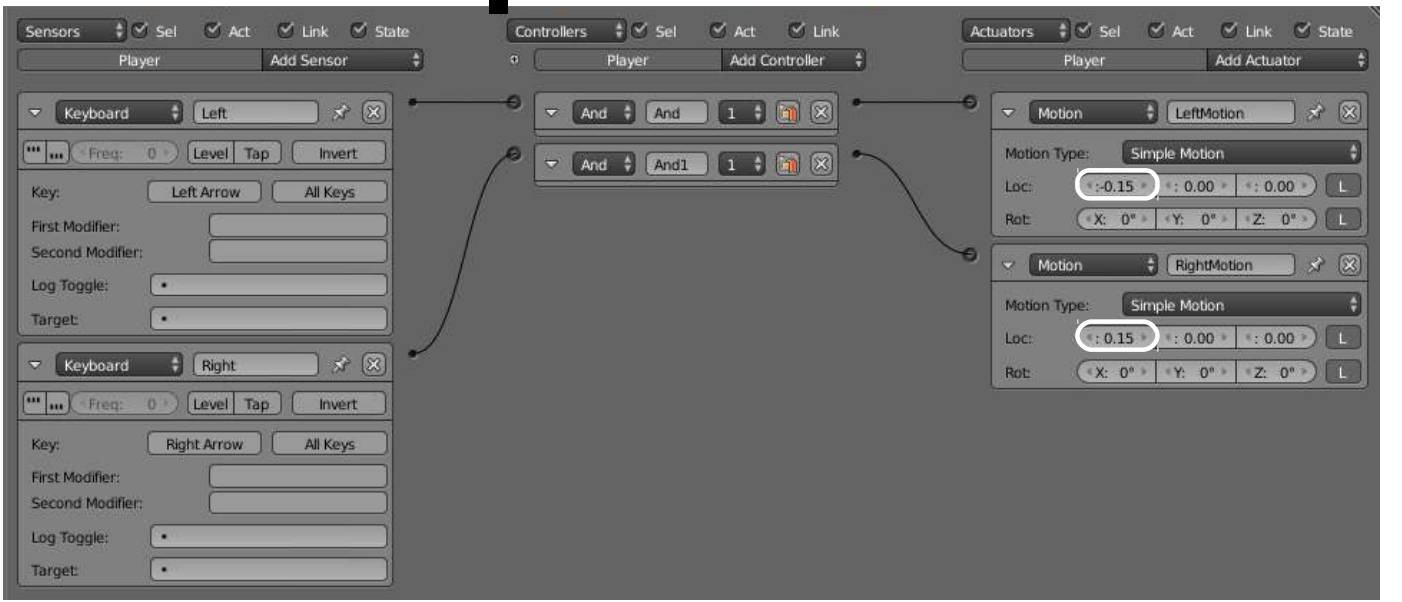
III Paso 2: Moviéndonos y Disparando

A continuación añadiremos los bloques lógicos para que la nave del jugador pueda moverse en el espacio. Simplemente añadiremos unos manejadores de teclado de forma que, mediante los cursores ←→, la nave realice desplazamientos con respecto del eje X global. El resultado de los bloques lógicos para este sencillo comportamiento puede verse en la Figura 20.

Para añadir disparo a la nave necesitamos dos elementos; por un lado el objeto a disparar. Crearemos una esfera de un color sólido que servirá como elemento de disparo. Por otro lado, un actuador que permita añadir objetos a la escena de forma dinámica. Si consultamos la referencia de actuadores en el capítulo 4, hay uno que sirve para ese propósito: ***Edit Object***.

El *problema* de *Edit Object* es que añade un objeto con respecto del centro del objeto que lo ejecuta. El centro de la nave está en su interior, y si añadimos objetos ahí, podrían colisionar con la propia nave. Una solución está en crear un objeto vacío "*Empty*" y emparentarlo con la nave, de modo que siga su movimiento. Es la solución que vamos a utilizar en este ejemplo.

Figura 20 Bloques lógicos para permitir el movimiento horizontal de la nave principal (Player).



Recordemos que los nuevos objetos en la escena se añaden con respecto a la posición del puntero 3D. Podemos cambiar la posición del puntero 3D numéricamente (en el panel de Propiedades) o directamente pinchando con en la vista 3D. Cuando el puntero 3D esté correctamente posicionado en la punta de la nave (ayudándonos de las vistas frontal, superior y lateral) añadimos un objeto Empty con ⇧ A ***Add/ Empty/ Arrows***. El resultado debe ser similar al de la Figura 21.

Igual que hicimos con el jugador, cambiamos el nombre del Empty a "***Empty-Player***" (ver Figura 22).

Figura 22 Cambiando el nombre al Empty

Finalmente, emparentamos el objeto *EmptyPlayer* con el objeto *Player*. Para ello, seleccionamos primero el objeto hijo (*EmptyPlayer*) y con pulsado, elegimos después el padre (el *Player*) y pulsamos Ctrl P y elegimos hacer padre al objeto (***Set Parent to Object***). Ahora, si desplazamos la nave, el objeto EmptyPlayer lo seguirá.

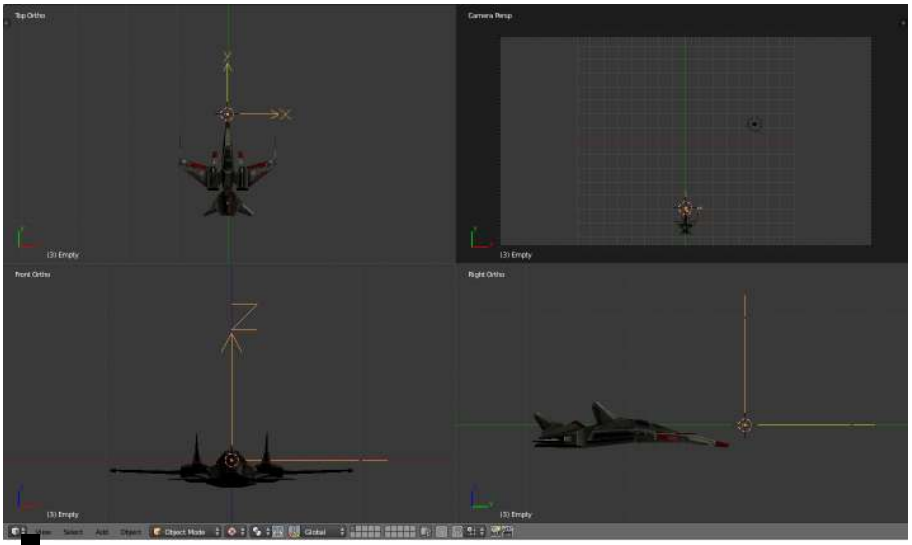


Figura 21 Añadiendo un Empty para que sirva como generador de disparos al objeto Player.

A continuación añadiremos el objeto que servirá como disparo. Pulsamos de nuevo ⇧ A ***Add/ Mesh/ UV Sphere.*** Ajustamos el tamaño de la esfera (con S) a algo similar a lo mostrado en la Figura 22. Cambiamos el nombre de la Esfera a "***Shot***". Creamos un nuevo material (pinchamos en New cuando accedamos a las propiedades de materiales) y definimos como propiedades de material un color blanco sólido y activamos ***Shadeless***, para que sea sin sombra (ver Figura 23).

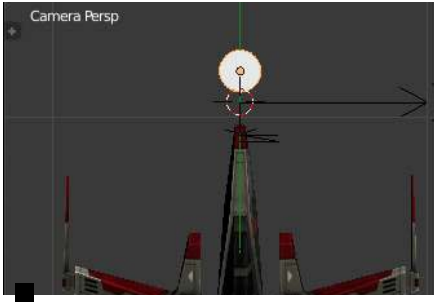


Figura 22 Objeto «Shot»

Figura 23 Propiedades del Material para el objeto de disparo «Shot»

Vamos a aplicar el tamaño internamente al objeto ***Shot***. Igual que hicimos con la nave, con el objeto Shot seleccionado, elegimos Ctrl A ***Apply Rotation & Scale***.

Ahora movemos a la segunda capa el objeto Shot (pulsamos M) y dejamos únicamente la primera capa activa. De esta forma, el objeto existe pero está en la 2ª capa oculto.

Hecho esto, vamos a añadir disparos cuando se pulse la barra espaciadora. Sobre el «*emisor*» de disparos (es decir, el objeto *EmptyPlayer*), conectamos con un *AND* los bloques lógicos de la Figura 24.

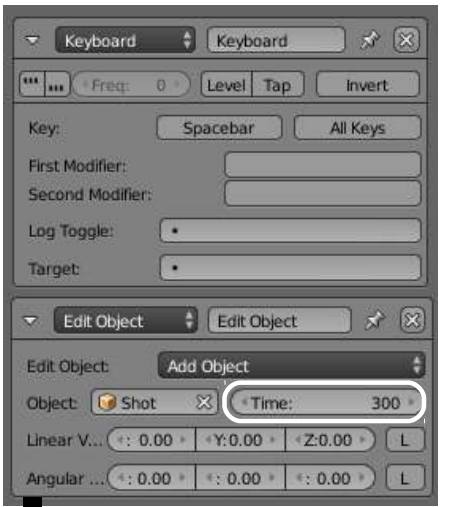


Figura 24 Bloques de EmptyPlayer

El campo ***Time*** del actuador *Edit Object* (ver Figura 24) indica el número de frames que permanecerá vivo el objeto desde su creación. Como el juego se ejecuta a 60 frames por segundo, un valor de 300 implica que el objeto creado se destruirá transcurridos 5 segundos. La destrucción de objetos es útil para evitar que se creen múltiples copias de un objeto y el sistema se colapse.

Si probamos ahora el juego comprobaremos que cuando pulsamos la barra espaciadora, se añade un disparo pero éste no se mueve. La solución es muy sencilla, basta con añadir un sensor de tipo ***Always*** sobre el objeto ***Shot*** que se encargue de actualizar la posición mediante un actuador de tipo ***Motion***. La descripción completa de las conexiones se muestra en la Figura 25.

Como se ha remarcado en la Figura 25, queremos que el movimiento se realice con respecto del *Sistema de Referencia Global*, por lo que desactivamos el botón ***L*** situado a la derecha de ***Loc***.

Si probamos ahora el juego, vemos que la nave ya es capaz de disparar múltiples disparos (que se destruyen automáticamente transcurridos 5 segundos). Parece que nuestro juego pide a gritos tener naves enemigas a las que poder disparar, ¿no te parece?. Vamos a añadir un poco de emoción al juego...

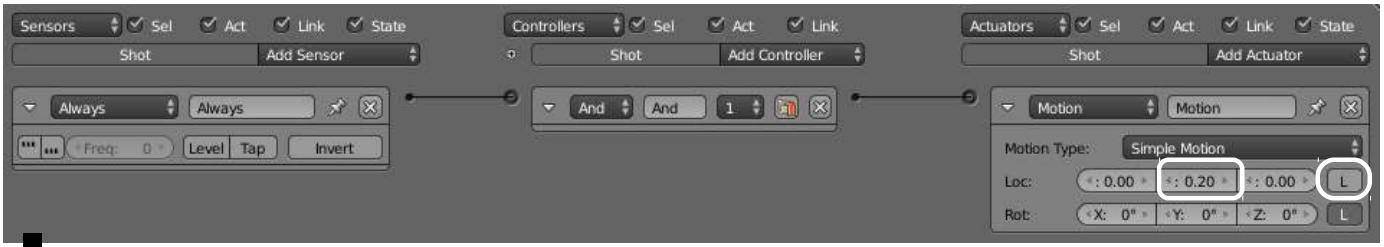


Figura 25 Actualización de movimiento sobre el objeto *Shot*.

Paso 3: Los primeros Enemigos...

En este tercer paso vamos a añadir un nuevo modelo de naves enemigas. Igual que hicimos con la nave principal, añadimos **File/ Append** del archivo *SpaceCruiser.blend* el objeto principal que se llama igual (SpaceCruiser). Ajustamos el tamaño [S] y la rotación [R], y le cambiamos el nombre a ***Enemy1***. Cuando tengamos el tamaño adecuado (similar al de la Figura 26), aplica los cambios internamente con [Ctrl] [A] ***Apply Rotation & Scale***.

Movemos el nuevo objeto a la capa 3 (pulsando la tecla [M]). Ahora tenemos en la única capa activa al jugador, en la capa 2 tenemos el disparo y en la capa 3 el primer tipo de enemigo.

Creamos un nuevo Empty con [⇧] [A] ***Add/ Empty/ Arrows***. Lo nombramos como ***EmptyEnemy1***, y lo situamos en la parte superior de la pantalla, fuera del campo de visión de la cámara (ver Figura 27).

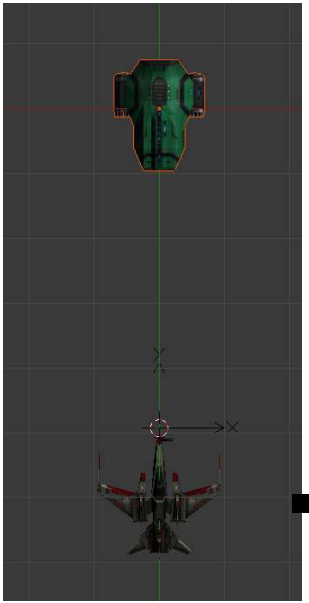


Figura 26 Ajuste de tamaño y rotación de Enemy1

Figura 27 Posición de EmptyEnemy1

Como se muestra en la Figura 27 es muy importante comprobar dos cosas en este nuevo «*Generador de Enemigos*». Primero que está fuera del campo de visión de la cámara (fuera del área sombreada), y segundo que el valor en Z es exactamente 0, para que esté perfectamente alineado con los disparos del jugador.

Figura 28 Bloques lógicos para el movimiento de los objetos de tipo Enemy1

Figura 29 Generación de enemigos aleatoria, con intervalos mínimos de 1 segundo (60 frames)

Igual que hicimos con el objeto *Shot* de disparo, añadiremos bloques lógicos sobre el ***Enemy1*** para que actualice su posición. En este caso, se realizará un movimiento en vertical, según el eje Y negativo (ver Figura 28). Se añade una rotación sobre el mismo eje, para dotar al objeto de mayor interés visual.

El generador de enemigos (***EmptyEnemy1***) creará instancias de ese tipo de forma aleatoria en el tiempo.

Como se muestra en la Figura 29, el emisor hace uso de un sensor de tipo ***Random***, que se lanzará aleatoriamente en intervalos de un mínimo de 60 frames (cada segundo o más), creando objetos mediante el actuador ***Edit Object*** de tipo *Enemy1*, que tendrán un tiempo de vida de 500 frames.

Con estos bloques lógicos conseguimos que se creen enemigos, pero falta que éstos puedan chocar con los disparos... ¡vamos a hacerlo en el siguiente paso!

III Paso 4: Colisionando con las Balas

A continuación definiremos los bloques lógicos para detectar la colisión con los disparos del personaje principal. Sobre el objeto de disparo ***Shot*** definiremos una propiedad que se llamará ***shot*** de tipo Booleano (ver Figura 33). El tipo no es importante, pero sí el nombre, porque nos servirá para filtrar las colisiones sólo con objetos que tengan esa propiedad. Así, los objetos enemigos utilizarán un sensor que detectará si han chocado con algún objeto que tenga esa propiedad (ver Figura 30).

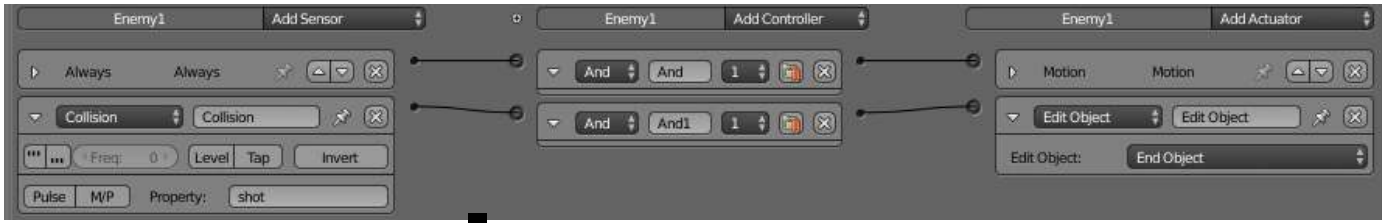


Figura 30 Destruyendo enemigos cuando chocan con un disparo

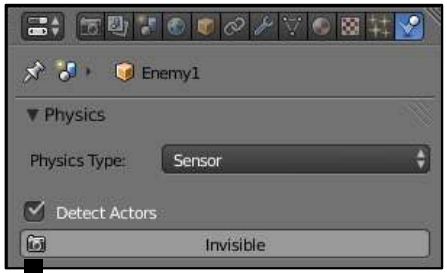


Figura 31 Colisión Física para *Enemy1*

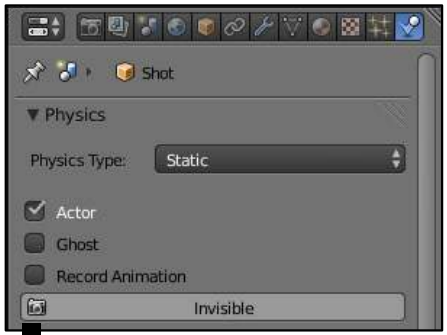


Figura 32 Colisión Física para *Shot*

Figura 33 Propiedad para disparo *Shot*

Además, hay que definir sobre los objetos dos perfiles de colisión (en las opciones de colisión física, como se muestra en las Figuras 31 y 32). Por un lado, el enemigo se definirá de tipo ***Sensor***, habilitando la opción de ***Detect Actors*** para que otros objetos puedan detectar su cercanía. Por otro lado, el disparo se define de tipo ***Static*** y ***Actor***, para que pueda colisionar con otros objetos.

Hecho esto, el enemigo que colisione con un disparo será eliminado de la escena. Podríamos ajustar el tiempo de generación de enemigos y de disparo para que la dificultad del juego sea mayor.

III Paso 5: Añadiendo un Marcador

Vamos a crear un marcador de puntuación para el juego. Para ello, añadimos un objeto de tipo Texto (⇧A ***Add/ Text***), y lo posicionamos como se muestra en la Figura 34. A este texto le cambiamos el nombre a "***Score***", y creamos una propiedad de texto pinchando en el botón ***Add Text Game Property*** (ver Figura 35). Esto nos creará una propiedad especial llamada ***Text*** que dibujará cualquier valor que contenga.

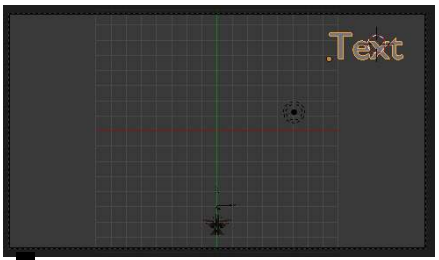
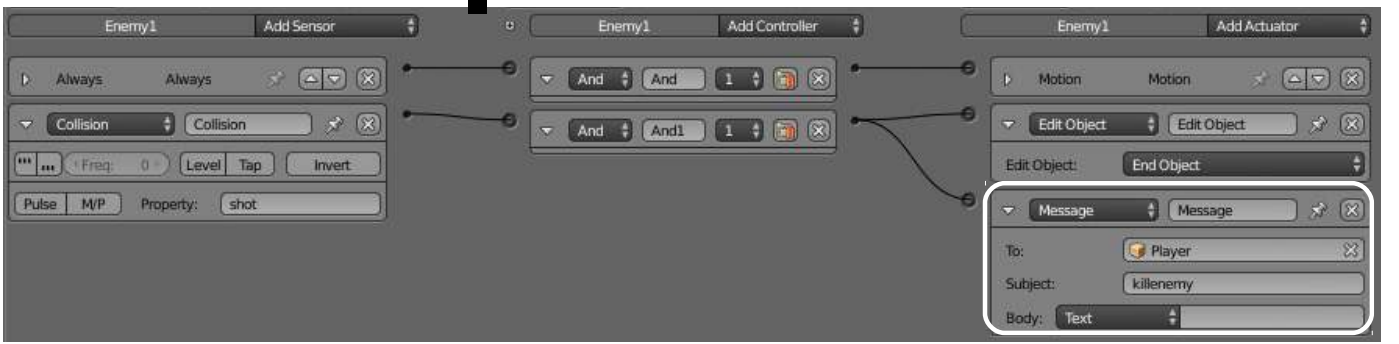


Figura 34 Posición del Marcador

Figura 35
Propiedad Text para el Marcador

Figura 36
Propiedad Text para el Marcador

Figura 37 Nuevos bloques lógicos para el Enemigo



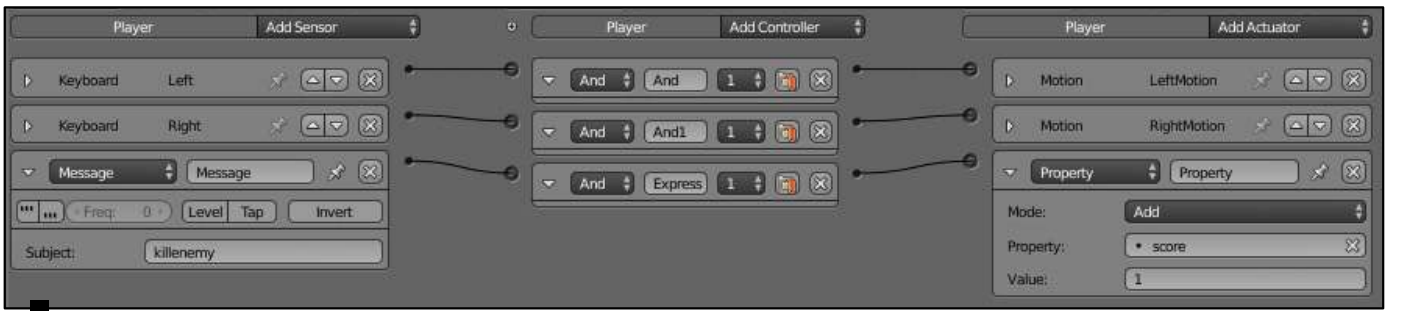


Figura 38 Actualización de la propiedad score del jugador

Figura 39 Bloques lógicos para el Marcador

El texto del marcador es una propiedad de tipo *String* (cadena de caracteres), que convertirá constantemente una propiedad de valor entero que contiene la puntuación. Así, puede verse el objeto ***Score*** como un simple representador de la puntuación. El que lleva la cuenta real de los puntos y decide cómo deben aumentarse o disminuirse es el ***Player***.

De esta forma, los bloques lógicos para el objeto ***Score*** se representan en la Figura 39. Por medio de un actuador de tipo ***Property*** se copia el valor de la propiedad «***score***» del jugador «***Player***» (ver Figura 36) en la propiedad «***Text***» del marcador.

Únicamente falta notificar al jugador cuando muera un enemigo. Es el propio objeto ***Enemy1*** el que manda un mensaje (mediante el actuador ***Message***) al ***Player***, indicando que ha muerto (ver Figura 37). El jugador, mediante tres bloques lógicos recibe la notificación e incrementa la puntuación en uno (Figura 38).

Paso 6: Cuidado con los Asteroides!!

Vamos a añadir otro tipo de objeto "enemigo" que no es sensible a los disparos, pero que debemos esquivar. Igual que hicimos con los modelos anteriores, añadimos ⊳ **File/ Append** del archivo *Asteroid.blend* el asteroide. Ajustamos el tamaño [S] y la rotación [R], y mantenemos el nombre de ***Asteroid***. Cuando tengas el tamaño adecuado (similar al de la Figura 40), aplica los cambios internamente con [Ctrl] [A] ***Apply Rotation & Scale***. Creamos un objeto ***EmptyAsteroid*** y lo posicionamos en la esquina superior izquierda, fuera de la pantalla. Movemos el *Asteroide* a la capa 4.

El código de generación del asteroide será similar al que utilizamos para generar los enemigos (ver Figura 29), pero generando objetos de tipo *Asteroid*.

Los Asteroides se moverán en *Zig-Zag*. Añadimos dos cubos [⇧] [A] ***Add/ Mesh/ Cube*** (escalados en vertical [S] [Y]) que servirán de límite lateral del mundo.

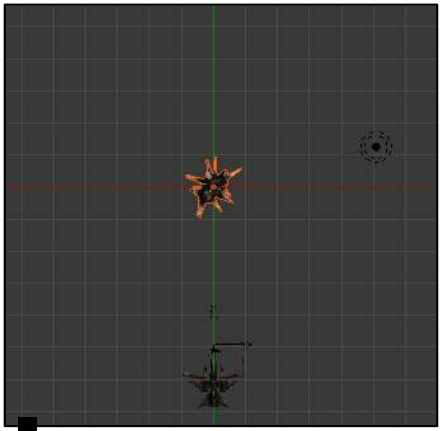


Figura 40 Asteroide

Llamaremos a estos cubos ***LimitRight*** y ***LimitLeft***, y los situaremos justo en el borde exterior de la cámara. Cuidado con la altura en Z de los objetos, que debe ser igual a 0 para que los asteroides puedan colisionar con ellos (ver Figura 41). Estos cubos deben estar en la misma capa que el objeto *Player*.

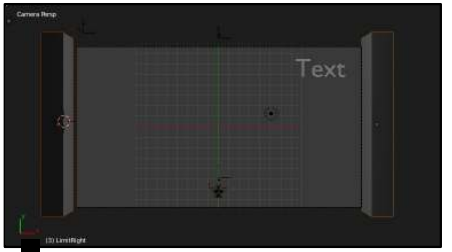


Figura 41 Posición de las cajas límite

Figura 43 Física del Asteroide

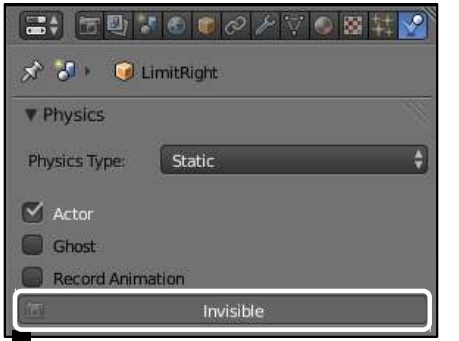


Figura 44 Física de los cubos Límite

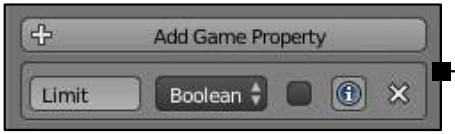


Figura 45

Propiedad sobre los Cubos Límite

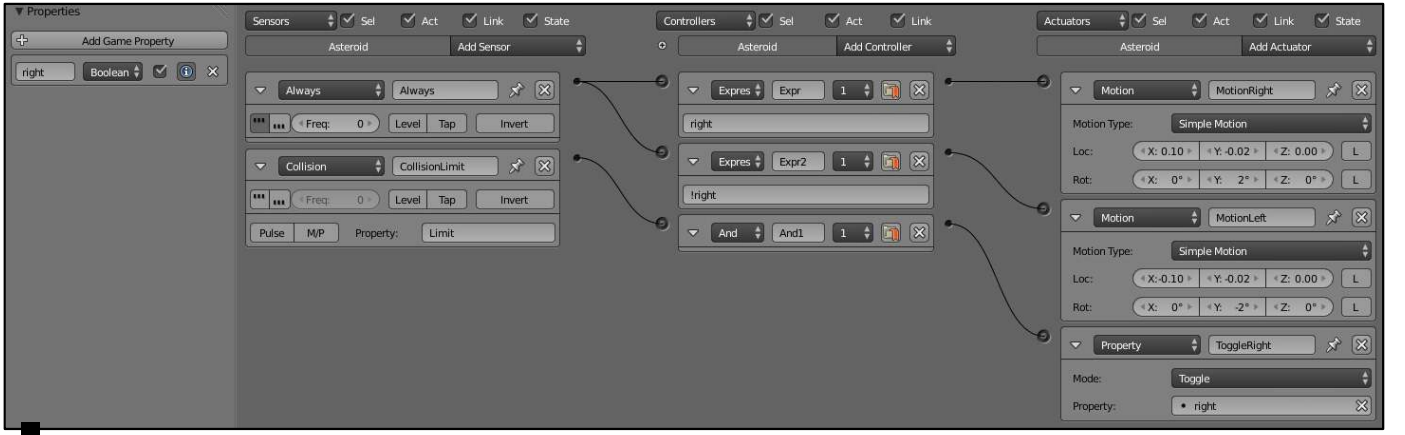


Figura 42 Bloques lógicos para el Asteroide

Las propiedades físicas de los cubos límite y del asteroide se definen de forma similar a como se hizo anteriormente con el enemigo y los disparos (ver Figuras 43 y 44). Cabe destacar que se ha activado la opción "***Invisible***" de los cubos límites para que no tengan representación gráfica; así, si no los hubiéramos añadido perfectamente en los límites de la cámara, no pasaría nada ya que no se dibujarán.

Los cubos límite tienen una propiedad definida llamada "***Limit***" (ver Figura 45), que nos permitirá filtrar y comprobar si hubo colisión con ellos por parte del asteroide.

La Figura 42 muestra los bloques lógicos para los asteroides. El asteroide mantiene una propiedad interna llamada ***right*** que indica si se desplaza hacia la derecha (si es cierta) o hacia la izquierda. Así, dependiendo del valor de ***right***, se ejecuta un actuador de movimiento u otro. Cuando el asteroide colisiona con una caja límite, cambia el valor de ***right*** (de *True* a *False* y de *False* a *True*), invirtiendo su sentido de desplazamiento. Lo que ocurre cuando un asteroide choca con el jugador se resume en las Figuras 46 y 47.

Para que funcionen los bloques lógicos, es necesario que el jugador sea ***Actor***, con una nueva propiedad ***player*** (ver Figura 48).

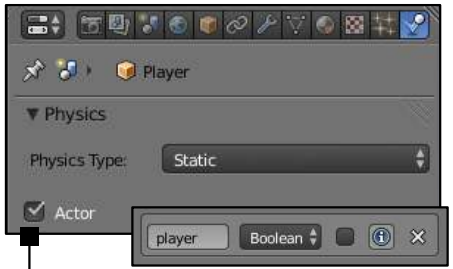


Figura 48 Nueva Propiedad y Física de Player

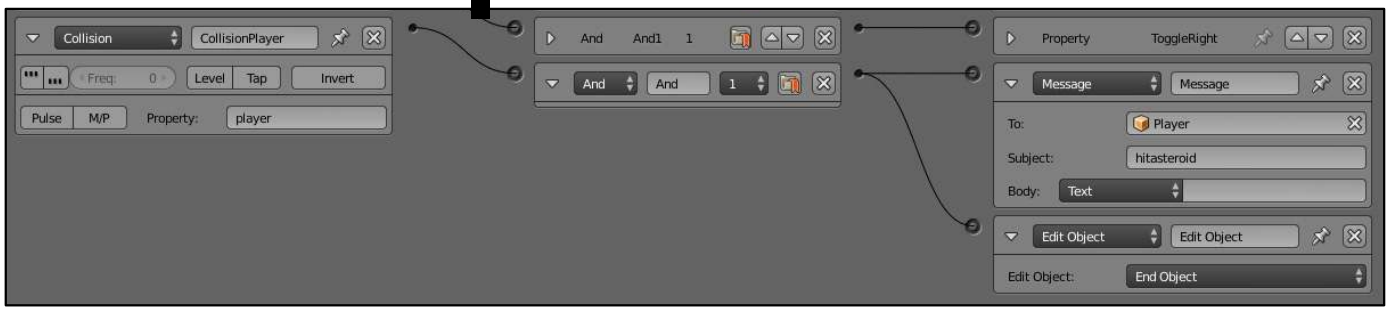


Figura 47 Nuevos bloques lógicos para *Player*

Figura 46 Nuevos bloques lógicos para *Asteroid*

Paso 7: Duplicando que es Gerundio

Para acabar este ejemplo, queda duplicar las fuentes de generación de enemigos y de asteroides.

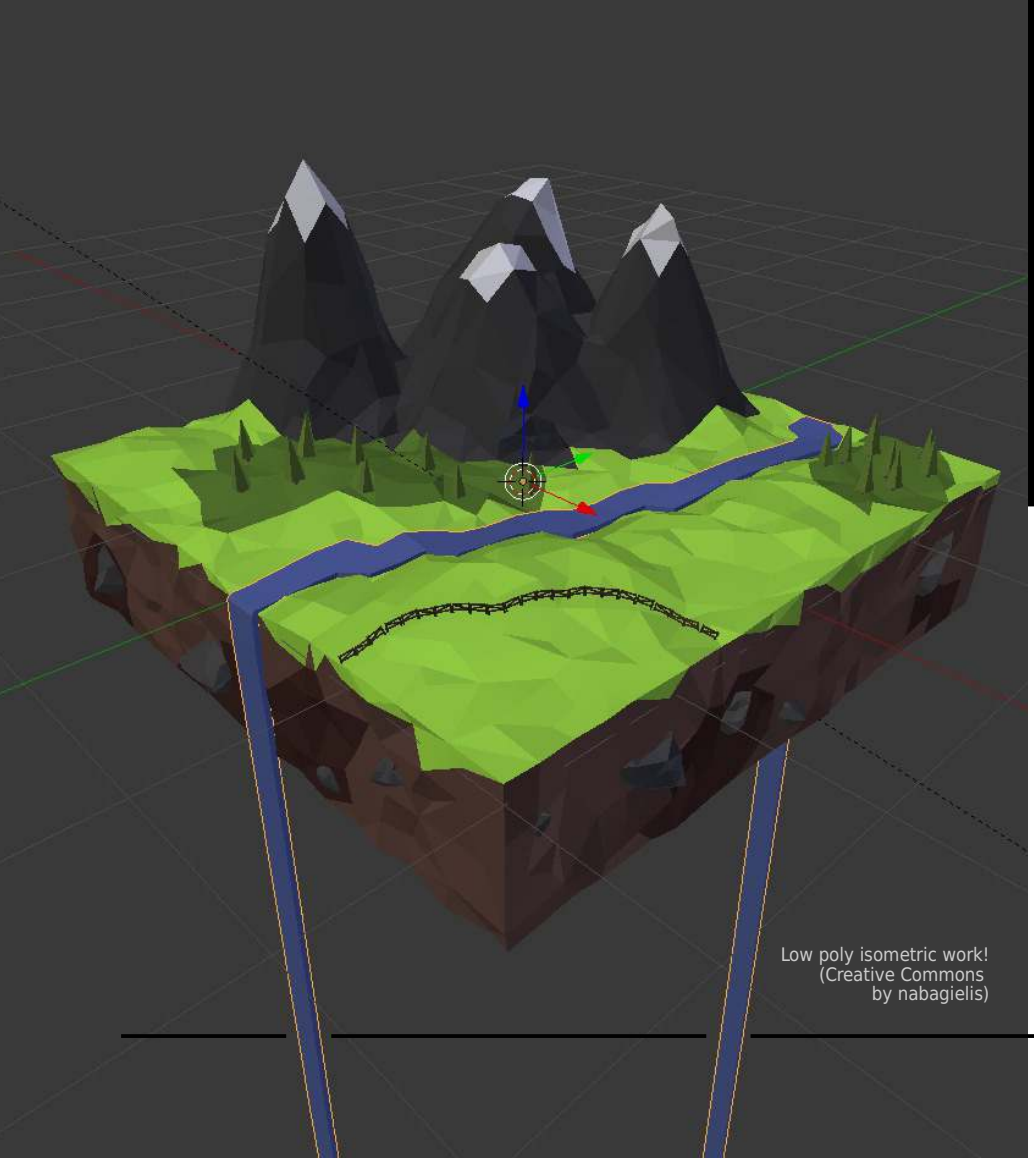
Crearemos otro emisor de asteroides en la zona derecha de la pantalla. Para ello, simplemente seleccionamos el ***Empty-Asteroid*** y pulsamos ⇧ D X. Con esto tendremos una nueva copia del objeto que desplazaremos únicamente en el eje X. Lo colocamos cerca del extremo superior derecho y... ya está. Tenemos otra fuente generadora de asteroides con la misma configuración.

La generación de nuevas fuentes de enemigos es igualmente sencilla. Elegimos el ***EmptyEnemy1*** y lo duplicamos con ⇧ D. Al final, podemos tener varias fuentes generadoras de enemigos y de Asteroides, como se muestra en la Figura 49.

Pasos 8, 9, 10 ... ¿Continuará?

¿Y ahora qué? Ahora queda que completes el juego como quieras. Te hemos proporcionado más modelos para que, si quieres, añadas nuevos tipos de enemigos (más resistentes a los impactos de disparos), que pongas nuevas condiciones de puntación (por ejemplo, si un enemigo se escapa por la parte inferior de la pantalla, penalice la puntuación, se puede hacer que la nave del jugador no pueda salirse de los límites de la pantalla, ajustar la velocidad de salida de los enemigos y asteroides, añadir sonidos mediante actuadores Sound, un contador de vidas del jugador... En definitiva, todo lo que te apetezca. Esperamos que este ejemplo haya despertado tu interés en este campo y que disfrutes mucho programando tus propios juegos con el Game Engine de Blender. ¡No olvides enviarnos tus creaciones!

Figura 49 Situación de las fuentes generadores de Enemigos y Asteroides.



Low poly isometric work!
(Creative Commons
by nabagielis)

Capítulo 4
Referencia de Bloques Lógicos

Este último capítulo de este librito resume los principales bloques lógicos del Logic Editor de Blender.

Se aconseja utilizar esta sección a modo de referencia rápida. Cualquier duda que no quede resuelta en este capítulo, muy probablemente lo estará en el *Wiki* oficial de Blender.

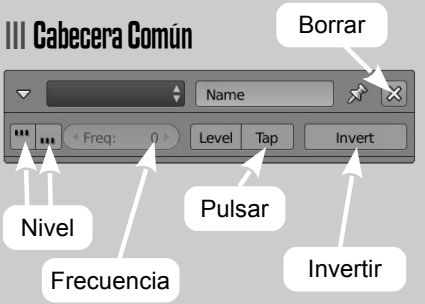
Queda fuera del propósito de este manual introductorio explicar la API de programación de Python, o estudiar algunos bloques lógicos avanzados, como los controladores para brazos robóticos o los actuadores de búsqueda de caminos. En los enlaces de la última sección del manual podrás encontrar algunas páginas donde profundizar en estos temas.

Sensores

Los sensores son los bloques lógicos que activan el resto de la lógica de la aplicación. Los sensores se encargan de dar una entrada cuando ocurre un evento determinado (por ejemplo, se pulsa una tecla, cuando ha pasado un determinado tiempo en un temporizador, o hay una colisión entre dos objetos que tienen una determinada propiedad). Cuando se activa un sensor, manda un pulso positivo a los controladores que tiene conectados.

La versión actual de Blender incorpora 14 tipos de sensores. A continuación estudiaremos, los principales sensores utilizados en programación de videojuegos.

Cabecera Común



El nivel (positivo o negativo) hace que el sensor se active según la frecuencia establecida a continuación (*Freq*). El campo de frecuencia (*Frequency*) establece el tiempo de espera entre la activación del sensor. Se establece en frames. Un valor 0 implica sin espera. La tasa por defecto (sin espera) es de 60Hz.

El botón de ***Tap*** hace que el sensor se active únicamente una vez, incluso si la entrada permanece activa. Esta opción es útil, por ejemplo, si el usuario mantiene una tecla pulsada y sólo queremos que se ejecute una vez el controlador.

Por último, la opción de ***Invert*** sirve para invertir la salida del sensor (se activa cuando no se cumple la entrada y viceversa).

Always

Este sensor se utiliza para tareas que necesitan realizarse en cada tick de reloj (o cada cierto número de ticks). Si sólo necesitas que se ejecute una vez al inicio de la aplicación (por ejemplo, en tareas de inicialización), activa el botón ***Tap***.

Delay

Este sensor pospone la inicialización de las acciones. En el campo ***Delay*** se indica el tiempo de espera para ejecutar la acción, con una duración indicada en ***Duration***. Si ***Repeat*** está activo, la acción se ejecutará cíclicamente.

Keyboard

Detecta la entrada por teclado de la aplicación. Puede guardar la pulsación en una propiedad de tipo String. Los modificadores (***Modifier***) especifican teclas adicionales que pueden pulsarse, para establecer combinaciones (como por ejemplo, detectar la pulsación de *Control + A*, o *Alt + V*...).

El campo ***Log Toggle*** es una propiedad booleana que sirve para determinar si las pulsaciones de tecla se guardan en la cadena de texto referenciada por ***Target***.

Mouse

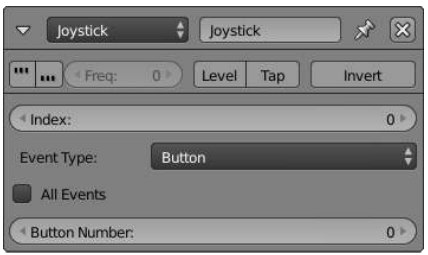
Detecta eventos de ratón. Estos eventos pueden ser los siguientes:

- **Mouse over any.** Es cierto (es decir, activa el controlador al que está unido) si el ratón se mueve sobre cualquier objeto.
- **Mouse over.** Es cierto si el ratón se mueve sobre el objeto que tiene definido este sensor.
- **Movement.** Devuelve cierto si hay algún movimiento de ratón.
- **Wheel Down.** Devuelve cierto si la rueda del ratón se mueve hacia abajo.
- **Wheel Up.** Devuelve cierto si la rueda del ratón se mueve hacia arriba.
- **Right Button.** Devuelve cierto si se pulsa el botón derecho del ratón.
- **Middle Button.** Devuelve cierto si se pulsa el botón del medio del ratón.
- **Left Button.** Devuelve cierto si se pulsa el botón izquierdo del ratón.

Actuator

Este sensor detecta cuando un determinado actuador se activa. En el campo ***Actuator*** se indica el nombre del actuador que queremos monitorizar.

Joystick



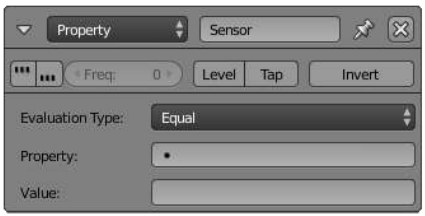
Activa el controlador cuando el Joystick se mueve. Empleando el campo ***Index*** se pueden controlar más de un Joystick.

Sensores

Collision

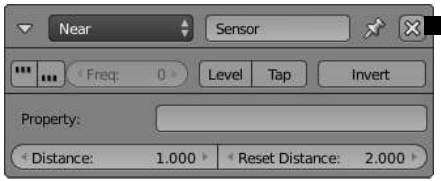
Envía un pulso positivo cuando el objeto choca con otro objeto. Mediante el campo ***M/P*** se puede definir si la colisión se detecta según un determinado material o una propiedad.

Property



Detecta cambios en las propiedades definidas en el objeto que lo contiene. El tipo de evaluación (***Evaluation Type***), que hace que se active el sensor, puede ser ***Changed***, ***Interval***, ***Not Equal***, ***Equal***.

Near



Detecta objetos dentro de un determinado rango de distancias.

Este sensor se activa cuando otro objeto se encuentra más cercano que una determinada distancia definida en el campo ***Distance***. Una vez que se ha activado, el sensor permanecerá en ese estado, enviando pulsos positivos, hasta que el objeto alcance una distancia mayor que la definida en ***Reset Distance***.

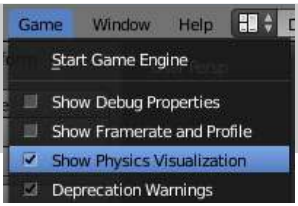
El campo ***Property*** permite filtrar únicamente aquellos objetos que tengan definida esa propiedad.

Sobre el uso del Sensor Near...

Este sensor detecta la cercanía de otros objetos, incluso atravesando objetos existentes. Es decir, aunque un objeto esté detrás de una pared, si cumple los rangos de distancia definidos en el sensor, será detectado. Los objetos a detectar deben ser «Actores»; es decir, deben tener la propiedad ***Actor*** activa en las opciones de simulación física.

¿Sabías que...?

Puedes activar en el menú principal ⊫ **Game/ Show Physics Visualization** para tener una representación del rango en el que se activará el sensor *Near*. Esto es muy útil para depurar el videojuego.



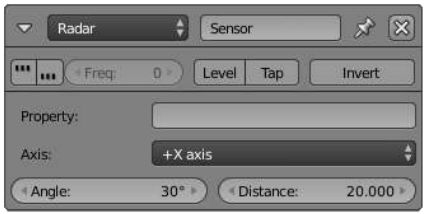
Message

Permite recibir mensajes de otros objetos. El campo ***Subject*** se puede dejar en blanco, de modo que recibe cualquier mensaje. Si se especifica una cadena de texto, solo se recibirán mensajes que tengan ese asunto concreto. El contenido del mensaje (***Body***) solo puede ser accedido mediante un controlador que ejecute código Python.

Random

Genera pulsos positivos de forma aleatoria. El campo ***Seed*** sirve para definir una semilla de números aleatorios.

Radar



Crea un cono de detección dentro del ángulo definido en el campo ***Angle***.

El comportamiento es similar al sensor de tipo *Near* estudiado anteriormente. En eje ***Axis*** determina la dirección positiva del cono de detección.

El ángulo ***Angle*** indica la apertura del cono (está definido entre 0 y 180 grados). La distancia definida en ***Distance*** inidica la altura del cono. Igual que se comentó en el sensor *Near*, resulta muy interesante activar la visualización de formas físicas para depurar el videojuego.

Ray

Este sensor lanza un rayo en una dirección, indicada en el eje ***Axis***. Este eje está definido en coordenadas locales. El campo ***Range*** sirve para definir la longitud del rayo. Igual que en el sensor *Collision*, es posible filtrar los objetos que intersecarán con el rayo por ***Material*** o Propiedad (***Property***).

El botón ***X-Ray Mode*** sirve para ver «a través» de objetos que no tienen la propiedad o material definida en el campo anterior.

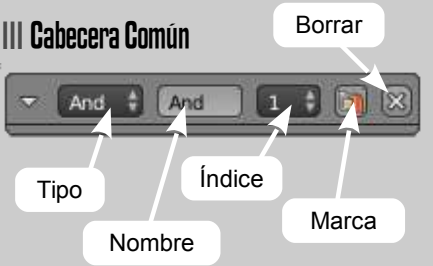
Controladores

Los controladores son bloques que recogen información enviada por los sensores y que especifican el estado en el cual operarán. Después de realizar las operaciones lógicas indicadas, envían un pulso de salida a los actuadores que tengan conectados.

Cuando un sensor es activado, envía un pulso positivo y cuando es desactivado, uno negativo. El trabajo de los controladores es verificar y combinar estos pulsos para disparar la respuesta apropiada.

Existen 8 tipos de controladores. En este manual nos centraremos en 7 de ellos, dejando el aprendizaje de los controladores de *Python* para un trabajo más avanzado.

Cabecera Común



Al igual que ocurría con los sensores, todos los controladores tienen una cabecera común. El campo de ***Marca*** permite forzar que la ejecución de el controlador se realice antes que el resto de controladores. Esto resulta de especial interés para la ejecución de scripts de inicialización.

El ***índice*** del controlador indica la capa de la máquina de estados a la que pertenece. Queda fuera del propósito de este manual el trabajo con capas de estados, por ser una característica avanzada del motor de videojuegos de Blender.

Operadores Lógicos

Blender permite trabajar con 6 operadores lógicos básicos. La Tabla 1 resume el comportamiento de las operaciones lógicas realizadas por los distintos tipos de controlador lógico. La primera columna (entrada), representa la cantidad de pulsos positivos enviados desde los sensores conectados. Las columnas siguientes representan la respuesta de cada controlador a esos pulsos. ***True*** significa que las condiciones del controlador han sido satisfechas y que los actuadores a los cuales se encuentra conectado serán activados; ***False*** significa que las condiciones del controlador no se alcanzaron y no se activará la salida.

Controlador Python

Una de las características avanzadas más interesantes del motor de videojuegos es el uso de scripts en Python. Queda fuera del ámbito de este manual profundizar en este tipo de controladores, pero si el lector ha utilizado algún lenguaje de programación, es muy recomendable estudiar la API del Motor de Videojuegos (ver referencia del manual oficial de Blender).

Sensores Positivos	Controladores					
	AND	OR	XOR	NAND	NOR	XNOR
Ninguno	✘ False	✘ False	✘ False	✔ True	✔ True	✔ True
Uno	✘ False	✔ True	✔ True	✔ True	✘ False	✘ False
Varios	✘ False	✔ True	✘ False	✔ True	✘ False	✔ True
Todos	✔ True	✔ True	✘ False	✘ False	✘ False	✔ True

Tabla 1

Controladores basados en expresiones lógicas

Expresión

Este tipo de controlador permite evaluar una expresión directamente escrita en su interfaz, generando un pulso positivo (que activa los actuadores que tenga conectados) cuando se cumpla que el resultado de la expresión es cierta.

La expresión puede estar formada por:

- **Variables.** Las variables pueden ser directamente nombres de ***sensores*** (serán ciertas cuando el sensor esté activo) o ***propiedades*** del objeto.
- **Operaciones.** Pueden ser operadores matemáticos básicos *, /, +, - y operadores lógicos <, >, >=, <=, ==, !=, AND, OR y NOT.

Ejemplo de uso

En el ejemplo de la Figura 1 se define una sencilla expresión sobre el controlador de tipo ***Expression***:

```
KilledByEnemy AND (lives>0)
```

Al utilizar el nombre de un sensor (***KilledByEnemy***), la expresión será cierta cuando la salida del sensor esté activa. En este caso, cuando se reciba un mensaje con asunto "***killed***". Además, en la expresión se realiza una comparación con una propiedad del objeto "*Player*", llamada "***lives***". Así, la salida del controlador será cierta (y por tanto, se activará el controlador que tenga conectado) cuando se reciba un mensaje con el asunto ***killed*** Y cuando el número de vidas (***lives***) del objeto sea mayor que cero. En cualquier otro caso, el actuador no será activado por este controlador.

Figura 1

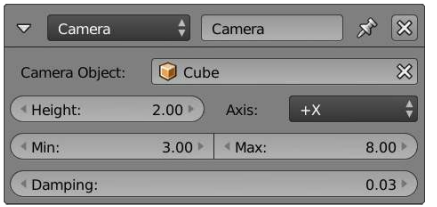
Ejemplo de uso de una expresión que utiliza la salida de un sensor y una propiedad

Actuadores

Los actuadores se encargan de ejecutar acciones sobre los objetos, como moverlos, crearlos en tiempo de ejecución, reproducir sonidos, etc... Los actuadores se ejecutan cuando reciben un pulso positivo desde los controladores a los que están conectados.

En la versión actual de Blender existen 16 tipos de actuadores. A continuación veremos algunos de los más útiles (omitiremos tres actuadores; *Action, State* y *Steering* por ser actuadores avanzados que escapan del ámbito de estudio de este pequeño manual.

Camera



Este actuador hace que la cámara (o el objeto sobre el que se aplique) siga a otro objeto, especificado en el campo ***Camera Object***. Este actuador hace que el objeto se mueva detrás del eje ***Axis*** del objeto de la cámara.

Los valores de ***Min***, ***Max*** y ***Height*** definen las distancias mínima y máxima de seguimiento, así como la altura.

El valor de ***Damping*** indica la fuerza de la restricción. A valores mayores el seguimiento será menos flexible.

Parent

Mediante este actuador se establece (con la opción ***Set Parent***) o se elimina (con ***Remove Parent***) la relación de parentesco entre objetos. Recordemos que mediante el parentesco, los objetos siguen siendo elementos independientes pero las transformaciones aplicadas al padre (traslación, rotación, escala...) son heredadas por el objeto hijo, pero no al contrario.

Parent Object especifica el nombre del objeto padre del que ejecuta este actuador.

Si ***Compound*** está activo, se añade la forma del objeto actual a la forma del objeto padre.

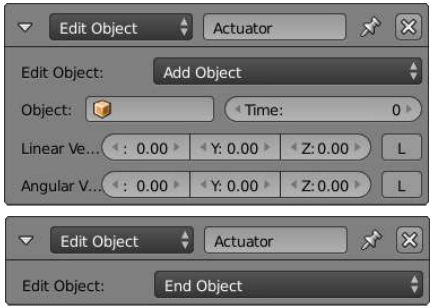
Ghost permite hacer el objeto "*fantasma*" mientras se emparenta.

Filter 2D

Actuador de filtro de imagen 2D que se aplica a la representación final de los objetos. Los filtros se aplican uno encima de otro en capas, igual que los filtros de programas de diseño gráfico 2D. El orden de aplicación se indica en el campo ***Pass Number***.

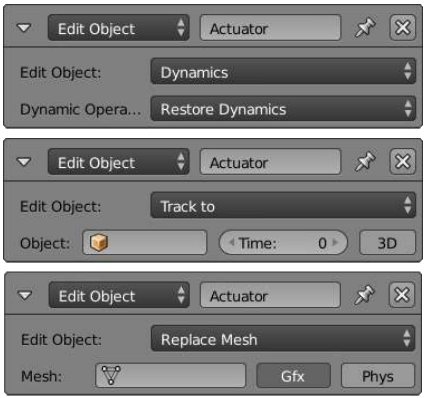
- ***Custom Filter:*** Define tu propio filtro 2D utilizando el lenguaje GLSL.
- ***Enable, Disable, Remove Filter:*** Los filtros son shaders que necesitan una gestión especial.
- ***Invert:*** Negativo de la imagen.
- ***Sepia:*** Colores más cálidos.
- ***Gray Scale:*** Escala de Grises.
- ***Prewitt:*** Filtro de detección de bordes.
- ***Sobel:*** Filtro de detección de bordes.
- ***Laplacian:*** Filtro detección bordes.
- ***Dilation:*** Brillo de píxeles cercanos.
- ***Erosion:*** Opuesto de *Dilation*.
- ***Blur:*** Desenfoque.
- ***Sharpen:*** Enfoque (Opuesto de *Blur*).
- ***Motion Blur:*** Desenfoque movimiento.

Edit Object



Edita las propiedades del objeto. Hay cinco opciones principales: *Add Object*, *End Object, Dynamics, Track To* y *Replace Mesh*.

- ***Add Object:*** Añade un objeto en la posición del centro del objeto que ejecuta este actuador. El parámetro ***Time*** indica el número de *frames* que el objeto creado permanecerá vivo en la escena (después será automáticamente eliminado). Es posible indicar la velocidad Lineal y Angular del objeto creado (***Linear Velocity*** y ***Angular Velocity***).
- ***End Object:*** Destruye el objeto actual.

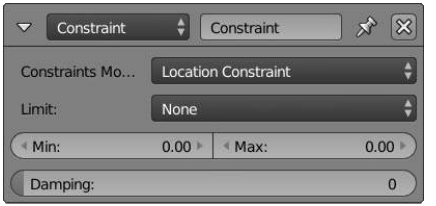


- ***Dynamics:*** Permite establecer propiedades dinámicas del objeto, como la masa, activar o desactivar la colisión de cuerpo rígido, o activar y desactivar la dinámica.
- ***Track To:*** Hace que el objeto apunte hacia otro objeto (en el espacio 2D o 3D).
- ***Replace Mesh:*** Tanto la malla como la forma de colisión física pueden ser reemplazadas utilizando este actuador.

Message

Permite enviar mensajes a otros objetos. El campo ***To*** permite especificar el objeto destinatario del mensaje. Si se deja en blanco, se enviará una copia del mensaje a todos los objetos del juego (*broadcast*). Mediante ***Subject*** indicamos el tipo de mensaje. Este campo resulta especialmente útil para filtrar mensajes cuando se reciben mediante el sensor *Message*. Finalmente, el campo ***Body*** sirve para indicar el cuerpo del mensaje. Este campo únicamente puede ser leído empleando un script en *Python*. El cuerpo puede ser cualquier cadena de texto (***Text***) o el valor de una propiedad (***Property***).

Constraint



Este actuador añade una restricción al objeto actual. La restricción puede ser de diferentes tipos:

- ***Location:*** Limita la posición del objeto en un determinado eje del mundo.
- ***Distance:*** Fuerza a que se mantenga una distancia determinada con respecto a una superficie.
- ***Orientation:*** Limita el rango de rotación del objeto en un determinado eje.
- ***Force Field:*** Crea un campo de fuerza a lo largo de un determinado eje del objeto.

Visibility

Cambia la visibilidad de un determinado objeto en tiempo de ejecución. Mediante el campo ***Visible*** establecemos si el objeto es o no visible. El campo ***Occlusion*** permite establecer si el objeto es ocultado por otros objetos. Finalmente el campo ***Children*** permite aplicar las mismas propiedades recursivamente a todos los objetos hijos del objeto que utiliza este actuador.

Game

Este actuador incorpora una serie de operaciones de alto nivel:

- ***Load/Save Global Dict:*** Permite cargar y guardar un diccionario global de Python, muy útil para compartir datos entre escenas.
- ***Quit Game:*** Salir del juego y finalizar la aplicación.
- ***Restart Game:*** Reiniciar el juego.
- ***Start Game from File:*** Carga un archivo .blend distinto.

Motion

Establece las propiedades de movimiento del objeto. Si el objeto no es estático (propiedad física), se pueden establecer también las propiedades dinámicas del mismo, como se muestra en la imagen.

- ***Loc:*** Permite indicar el número de unidades de Blender en las que se desplazará el objeto.
- ***Rot:*** Valor de rotación del objeto.
- ***L:*** Si está seleccionado, las coordenadas del movimiento son con respecto del sistema de referencia del objeto.

Scene

Gestiona las escenas del archivo.

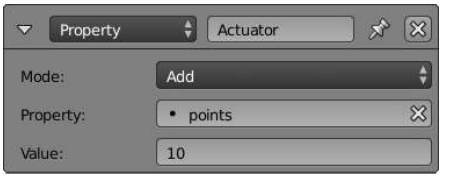
- ***Set Scene:*** Cambia como escena actual la seleccionada. .
- ***Set Camera:*** Establece como cámara la indicada como parámetro.
- ***Remove Scene:*** Elimina una escena.
- ***Suspend Scene:*** Pausa la escena.
- ***Resume Scene:*** Reanuda la escena previamente pausada.
- ***Add Overlay Scene:*** Añade otra escena que se dibujará encima del resto de elementos. Esta funcionalidad es especialmente útil para elementos de la interfaz de usuario.
- ***Add Background Scene:*** Dibuja otra escena detrás de todos los elementos de la escena actual.
- ***Restart:*** Reinicia la escena.

Sound

Este actuador permite reproducir música y efectos de sonido empleando la biblioteca *Audaspace*. El actuador permite varios modos de reproducción:

- ***Play Stop:*** El sonido se reproduce únicamente mientras el actuador está activo.
- ***Play End:*** El sonido se reproduce completo hasta que finaliza.
- ***Loop Stop:*** El sonido se reproduce en modo de bucle hasta que el actuador se desactiva.
- ***Loop End:*** Igual que *Loop Stop*, pero el sonido se reproduce hasta que finaliza (completo).

Property



Cambia el valor de una propiedad del objeto. Permite cuatro modos de actuación:

- ***Assign:*** Asigna un valor a la propiedad.
- ***Add:*** Añade el valor indicado en ***Value*** al valor actual de la propiedad ***Property***.
- ***Copy:*** Copias el valor de una propiedad de otro objeto en una propiedad del objeto actual.
- ***Toggle:*** Cambia el valor de 0 a 1.

Random

Este actuador permite generar números aleatorios en propiedades del objeto.

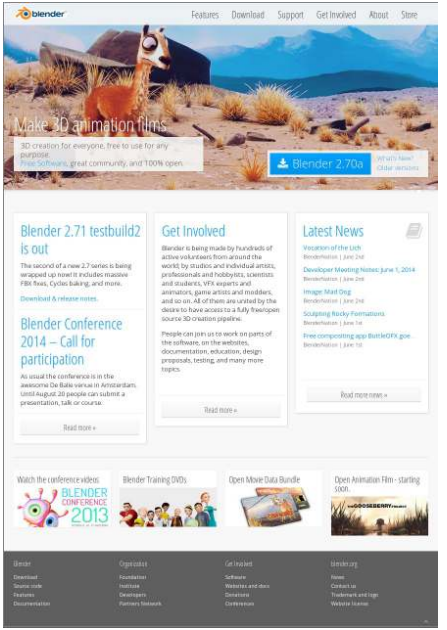
- ***Seed:*** Valor de la semilla de los números aleatorios.
- ***Property:*** Propiedad que recibe el valor aleatorio. Es importante elegir un método de distribución de los aleatorios (campo *Distribution*) que sea coherente con el tipo de la propiedad.
- ***Distribution:*** Existen diversos tipos de métodos distribución: *Float*, *Integer* y *Boolean*. Igualmente es posible devolver un valor constante.

Referencias

Enlaces Web para el Estudio Autónomo

Para finalizar, en esta sección se recogen algunos de los principales recursos disponibles on-line para el estudio autónomo. Para facilitar el estudio, se han elegido recursos en Castellano, aunque al final del listado se recopilan sitios con documentación en Inglés de gran calidad.

Esta recopilación está centrada en Blender como herramienta integral para el diseño 3D. En la actualidad, existen muchos lenguajes y alternativas para profundizar en el desarrollo de videojuegos, pero esa es otra historia que debe ser contada en otra ocasión. Disfruta del camino y no olvides divertirte en todo lo que hagas. Mucha suerte y *Happy Blending!*



Web Oficial de Blender (Inglés)

http://www.blender.org

Web oficial de Blender. En esta página puedes descargar tanto la última versión para tu sistema operativo, como todas las versiones publicadas desde hace 15 años. Como sitio oficial, contiene los enlaces al manual de usuario de la herramienta y la API del programador en Python.

Manual Oficial de Blender (Español)

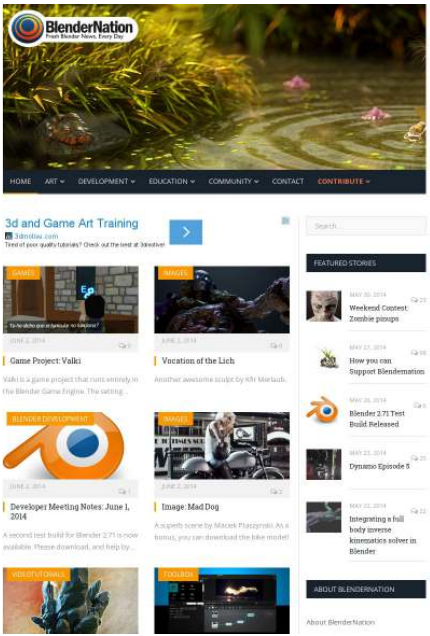
http://wiki.blender.org/index.php/ ↵ Doc:ES/2.6/Manual

Manual oficial de Blender, en formato Wiki. En la parte superior se puede elegir el idioma. La traducción al español está muy conseguida y es prácticamente completa, aunque en algunas secciones la página original (en Inglés) es más extensa.

Blender 3D en la Educación (Español)

http://www.ite.educacion.es/ ↵ formacion/materiales/181/cd/

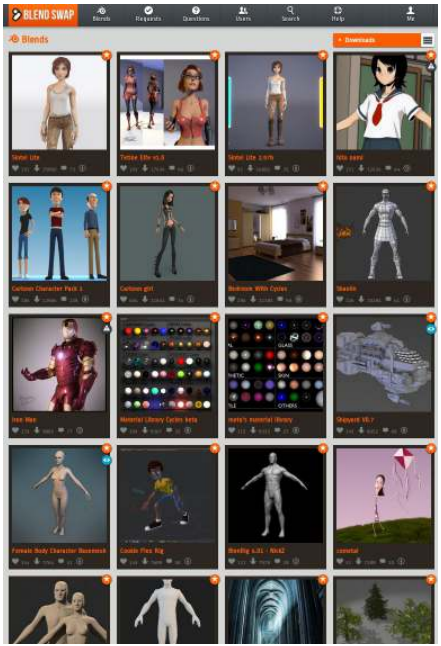
Curso totalmente en castellano creado por Joaquín Herrera Goás. Cubre los aspectos esenciales de Blender, con un enfoque basado en pequeños problemas. Es uno de los mejores cursos en castellano existentes para introducirse en la herramienta.



Blendernation (Inglés)

http://www.blendernation.com

Principal portal de noticias dedicado esencialmente a Blender. También se publican otras noticias de interés sobre Gráficos 3D, Imagen Digital y Videojuegos. Altamente recomendable para estar a la última en tu herramienta 3D favorita. :-)



Blendswap (Inglés)

http://http://www.blendswap.com/

Web de descarga de modelos libres de Blender de calidad profesional. En la licencia (habitualmente de tipo Creative Commons) se indica si pueden ser utilizados en proyectos comerciales. La web es en Inglés, por lo que utiliza el buscador con términos en la lengua de Shakespeare.

3D Poder (Español)

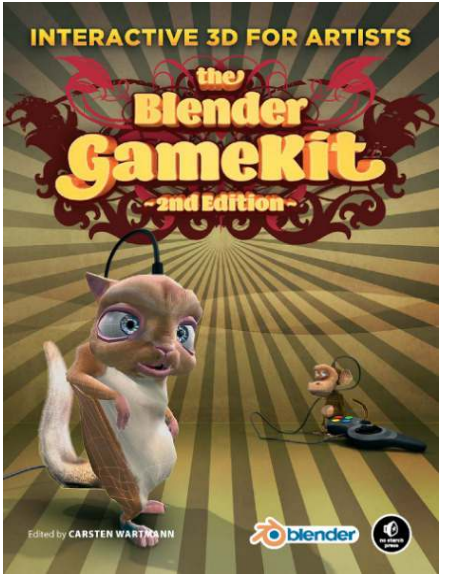
http://www.foro3d.com/

Este foro es probablemente uno de los mejores en castellano. Existen foros en lengua inglesa de alta calidad, pero hemos querido elegir un sitio donde puedas plantear tus dudas en castellano. Este foro tiene secciones específicas por programa. ¡Visita la de Blender y ayuda a los demas!

Blender Gamekit 1 (Inglés)

http://download.blender.org/documentation/gamekit1/

Libro oficial de la Blender Foundation completo en formato electrónico sobre el motor de videojuegos de Blender. Aunque emplea versiones antiguas de Blender, la mayoría de ejemplos y casos de estudio siguen siendo válidos.



Blender Gamekit 2 (Inglés)

http://download.blender.org/documentation/gamekit2/

Actualización del Blender Gamekit 1 empleando las últimas versiones de Blender. Al igual que el Gamekit 1, es posible descargar el PDF completo y los ejemplos discutidos en el libro.

Este manual fue compuesto con
OpenOffice en una máquina GNU/Linux
entre Mayo y Junio de 2014