liliana canalsova

kevin amores

javier blanco

laia pomar

TG2 – COMPARACIÓN TEÓRICA MOTORES DE VIDEOJUEGOS -

DESARROLLO DE TECNOLOGÍAS EMERGENTES

11 de abril de 2016

ÍNDICE

[1. Autores del trabajo, planificación y entrega 4](#_Toc448168037)

[1.1 Autores 4](#_Toc448168038)

[1.2 Planificación 4](#_Toc448168039)

[1.3 Entrega 4](#_Toc448168040)

[2. Descripción de las tecnologías 5](#_Toc448168041)

[2.1 Descripción de Unity 5](#_Toc448168042)

[2.1.1 Historia 5](#_Toc448168043)

[2.1.2 Descripción general 5](#_Toc448168044)

[2.2 Descripción de Unreal Engine 6](#_Toc448168045)

[2.2.1 Historia 6](#_Toc448168046)

[2.2.2 Descripción general 6](#_Toc448168047)

[3. Criterios de comparación 7](#_Toc448168048)

[3.1 DISEÑO 7](#_Toc448168049)

[3.1.1 Interfaz 7](#_Toc448168050)

[3.1.2 Recursos predefinidos 9](#_Toc448168051)

[3.2 CARACTERÍSTICAS GENERALES 12](#_Toc448168052)

[3.2.1 Multiplataforma 12](#_Toc448168053)

[3.2.2 Licencia 14](#_Toc448168054)

[3.3 RENDIMIENTO 15](#_Toc448168055)

[3.3.1 Optimización 15](#_Toc448168056)

[3.4 UTILIDAD 19](#_Toc448168057)

[3.4.1 Compatibilidad con programas externos 19](#_Toc448168058)

[3.4.2 Servicios 21](#_Toc448168059)

[3.5 PROGRAMACIÓN 23](#_Toc448168060)

[3.5.1 Lenguaje 23](#_Toc448168061)

[3.5.2 API 25](#_Toc448168062)

[3.6 GRÁFICOS 27](#_Toc448168063)

[3.6.1 Renderizado 27](#_Toc448168064)

[3.6.2 Materiales y shaders 29](#_Toc448168065)

[3.6.3 Animación 2D y 3D 31](#_Toc448168066)

[3.7 SONIDO 35](#_Toc448168067)

[3.7.1 Recursos propios dentro del entorno 35](#_Toc448168068)

[3.7.2 Programación del sonido dentro del juego 37](#_Toc448168069)

[3.8 RECURSOS 39](#_Toc448168070)

[3.8.1 Formación externa 39](#_Toc448168071)

[3.8.1.1 Formación externa en Unity 39](#_Toc448168072)

[3.8.2 Recursos multimedia externos 41](#_Toc448168073)

[3.9 HARDWARE 43](#_Toc448168074)

[3.9.1 Requisitos mínimos para ejecutar el entorno 43](#_Toc448168075)

[3.9.2 Requisitos mínimos para ejecutar el juego 44](#_Toc448168076)

[3.10 SOPORTE PROPIO 45](#_Toc448168077)

[3.10.1 Documentación 45](#_Toc448168078)

[3.10.2 Asistencia técnica 46](#_Toc448168079)

[4. Evaluación de los criterios por tecnología 48](#_Toc448168080)

[4.1 Evaluación de los criterios para Unity 48](#_Toc448168081)

[4.2 Evaluación de los criterios para Unreal Engine 49](#_Toc448168082)

[5. Comparación de las tecnologías 50](#_Toc448168083)

[6. Recomendaciones 51](#_Toc448168084)

[6.1 Situación 1 51](#_Toc448168085)

[6.2 Situación 2 52](#_Toc448168086)

# 1. Autores del trabajo, planificación y entrega

## 1.1 Autores

Se trata del grupo M1 y sus integrantes son:

* Laia Pomar
* Liliana Canalsova
* Javier Blanco
* Kevin Amores

## 1.2 Planificación

## <https://app.ganttpro.com/#!/app/home>

## 1.3 Entrega

https://github.com/laiapc/TG2

# 2. Descripción de las tecnologías

## 2.1 Descripción de Unity

### 2.1.1 Historia

La empresa **Unity Technologies fue fundada en 2004** por David Helgason (CEO), Nicholas Francis (CCO), y Joachim Ante (CTO) en Copenhague, Dinamarca después de su primer juego, GooBall, que no obtuvo éxito. Los tres reconocieron el valor del motor y las herramientas de desarrollo y se dispuso a crear un motor que cualquiera pudiera usar a un precio accesible.

El éxito de Unity ha llegado en parte debido al **enfoque** en las necesidades de los **desarrolladores independientes** que no pueden crear ni su propio motor del juego ni las herramientas necesarias o adquirir licencias para utilizar plenamente las opciones que aparecen disponibles.

El enfoque de la compañía es "democratizar el desarrollo de juegos", y hacer el desarrollo de contenidos interactivos en 2D y 3D lo más accesible posible a tantas personas en todo el mundo como sea posible.

La primera versión de Unity se lanzó en la Conferencia Mundial de Desarrolladores de Apple en 2005. Fue construido exclusivamente para funcionar y generar proyectos en los equipos de la plataforma Mac y obtuvo el éxito suficiente como para continuar con el desarrollo del motor y herramientas.

Unity 3 fue lanzado en septiembre de 2010 y se centró en empezar a introducir más herramientas que los estudios de alta gama por lo general tienen a su disposición, con el fin de captar el interés de los desarrolladores más grandes, mientras que proporciona herramientas para equipos independientes y más pequeñas que normalmente serían difíciles de conseguir en un paquete asequible.

La última versión de Unity**, Unity 5 lanzada a principios de 2015**, se anunció en Game Developers e incluye añadidos como Mecanim animation, soporte para DirectX 11 y soporte para juegos en Linux y arreglo de bugs y texturas. Desarrollado por creadores de juegos para mayor expectativa.

### 2.1.2 Descripción general

El motor gráfico utiliza **Direct3D** (en Windows), **OpenGL** (en Mac y Linux), OpenGL ES (en Android y iOS), e interfaces propietarias (Wii). Tiene soporte para mapeado de relieve, reflexión de mapeado, mapeado por paralaje, pantalla de espacio oclusión ambiental (SSAO), sombras dinámicas utilizando mapas de sombras, render a textura y efectos de post-procesamiento de pantalla completa.

El soporte integrado para **Nvidia** (antes Ageia), el motor de física **PhysX**, (a partir de Unity 3.0) con soporte en tiempo real para mallas arbitrarias y sin piel, ray casts gruesos, y las capas de colisión. El scripting viene a través de Mono. El script se basa en Mono, la implementación de código abierto de .NET Framework. Los programadores pueden utilizar UnityScript (un lenguaje personalizado inspirado en la sintaxis ECMAScript), C# o Boo (que tiene una sintaxis inspirada en Python). A partir de la versión 3.0 añade una versión personalizada de MonoDevelop para la depuración de scripts.

Unity también incluye **Unity Asset Server** - una solución de control de versiones para todos los assets de juego y scripts, utilizando PostgreSQL como backend, un sistema de audio construido con la biblioteca FMOD, con capacidad para reproducir audio comprimido Ogg Vorbis, reproducción de vídeo con códec Theora, un motor de terreno y vegetación , con árboles con soporte de billboarding, determinación de cara oculta con Umbra, una función de iluminación lightmapping y global con Beast, redes multijugador RakNet y una función de búsqueda de caminos en mallas de navegación.

## 2.2 Descripción de Unreal Engine

### 2.2.1 Historia

La primera versión de **Unreal Engine se desarrolló en 1998**, sus características principales integraban renderizado, detección de colisiones, IA, visibilidad, opciones para redes y manipulación de archivos de sistema en un motor bastante completo.

La siguiente versión apareció en 2002 con el desarrollo de “America’s army”. Se reestructuro todo el entorno, tanto a nivel del núcleo como el código completo y el motor de renderizado. Además, se incluyeron sistemas de física y soporte para las primeras consolas.

Dicha versión se actualizó a la 2.5 que mejoró el rendimiento y añadió física para vehículos y la edición del sistema de partículas. También se mejoró de cara a Xbox y se añadieron efectos de sonido EAX 3.0

En el 2006 apareció la tercera versión de Unreal Engine, preparado para DirectX 9-10, Xbox 360 y PlayStation 3. Para ello se añadió soporte para técnicas avanzadas de HDRR, mapping y sombras dinámicas. También se añadieron herramientas complementarias sobre las de anteriores versiones, modificando el sistema de física.

Esta versión sufrió varias adaptaciones y mejoras por distintos motivos. Se mejoró de cara a consolas como la Wii, se añadió el kit para desarrolladores amateur y se añadieron simuladores de construcción, conducción etc.

### 2.2.2 Descripción general

Unreal Engine es un motor de juego de PC y consolas creados por la compañía Epic Games. En sus inicios se utilizó para implementar el shooter llamado “Unreal” en el 1998, posteriormente a soportado algunas entregas de juegos conocidos como Unreal Tournament, BioShok, Star Wars, Batman etc.

El entorno está **escrito íntegramente en C++** y utiliza distintas librerías gráficas para abarcar cualquier temática del área de videojuegos. Su primer motor tiene versiones para Windows bajo DirectX, GNU/Linux y Macintosh bajo OpenGL, PlayStation2 y Dreamcast. A lo largo de su historia ha ido evolucionando, desarrollando el entorno para adaptarlo a OpenGL y DirectX9/10, posteriormente se desarrolló para poder programar para casi todas las consolas Xbox, Playstation 3, Game Cube y todos los sistemas de pc, Windows XP/Vista/7, Unix/Linux/Mac.

# 3. Criterios de comparación

Antes de desarrollar el trabajo queremos destacar que toda la documentación del mismo se basa en las páginas de documentación propias de los entornos, que son:

Unity: <http://docs.unity3d.com/es/current/Manual/>

Unreal Engine: <https://docs.unrealengine.com/latest/INT/>

Por ello solo se especificarán más URL’s en los apartados que hayan utilizado algún otro recurso.

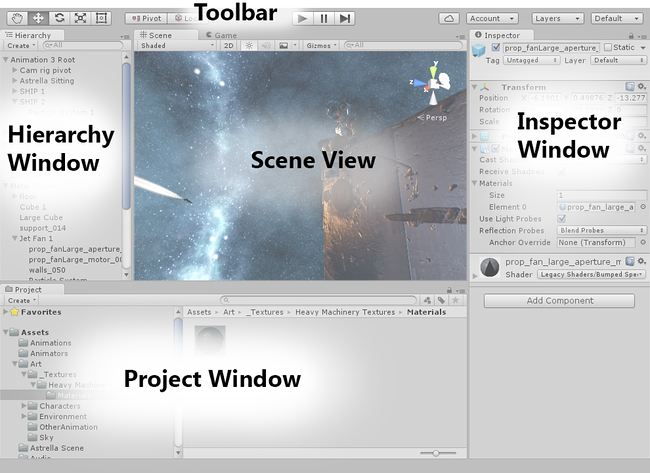
## 3.1 DISEÑO

### 3.1.1 Interfaz

* Nombre: interfaz
* Descripción: características de la interfaz del entorno
* Tipo de valor: texto

#### 3.1.1.1 Interfaz Unity

La interfaz de Unity es bastante sencilla, está compuesta por los siguientes elementos:



La **barra de herramientas** proporciona acceso a las principales funciones de Unity. A la izquierda contiene las herramientas más básicas para la manipulación de la escena y los. En el centro están los controles de reproducción. A la derecha se encuentran los accesos a la nube y a la cuenta de usuario, seguidos de un menú de visibilidad de las capas, y finalmente el menú Editor de diseño (que proporciona algunos diseños alternativos y permite guardar los diseños propios).

La **ventana de proyecto** muestra su biblioteca de activos o recursos disponibles para utilizar en el proyecto, ordenados por categorías. Todo lo que se guarde en la carpeta predefinida del proyecto se importará automáticamente.

La **vista de escena** es donde pondremos todos los elementos de nuestro juego. Permite editar la escena visual, ya sea desde una perspectiva 3D o 2D, dependiendo del tipo de proyecto que se está trabajando. Tanto los objetos como los escenarios se pueden arrastrar a la posición deseada. Contiene 2 pestañas: scene y game, cambiando a la pestaña game podremos ver cómo se comportaría el juego con los elementos y las acciones programadas en la vista de escena.

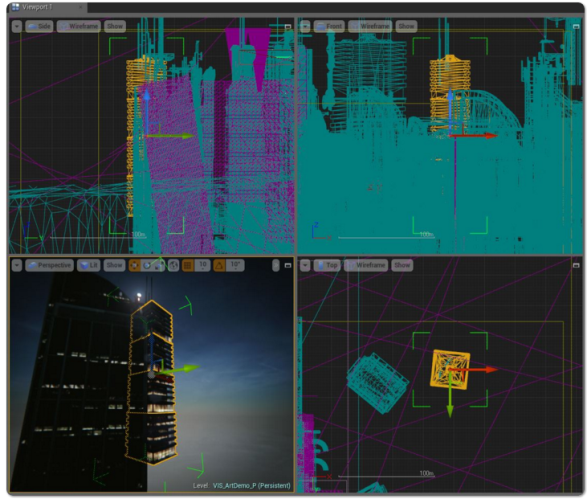
La **ventana de jerarquía** La jerarquía muestra la forma en que todos los elementos de nuestro juego están relacionados entre sí.

La **ventana de inspector** permite ver y editar todas las propiedades de objeto que se haya seleccionado. El contenido de esta ventana variará en dependencia del tipo de objeto seleccionado, ya que los diferentes tipos tienen diferentes propiedades.

#### 3.1.1.2 Interfaz Unreal Engine

**Editor de niveles**, esta parte del entorno permite construir los niveles y vistas. En él se modifican principalmente la colocación, la transformación y la edición de las propiedades de los actores. Permite definir el mundo a través de objetos geométricos ya sean luces, mallas o personajes. Reduciéndolo al proceso más básico, esta construcción consiste en colocar los objetos en el mapa y programar su comportamiento e integración.





Además, en este gestor de contenidos se pueden crear carpetas y subcarpetas que permitan organizar el contenido del juego (texturas, modelos, sonidos, blueprints, animaciones, esqueletos, etc.). En estas carpetas se nos permite cambiar el nombre o mover cualquier asset de sitio y el engine recalculará las dependencias en base a la nueva ruta, haciendo mucho más fácil el desarrollo. También resulta indispensable de cara a optimizar la carga y descarga de contenido en las etapas finales del desarrollo y poder reorganizar todos los recursos la carpeta “prototipo”.

### 3.1.2 Recursos predefinidos

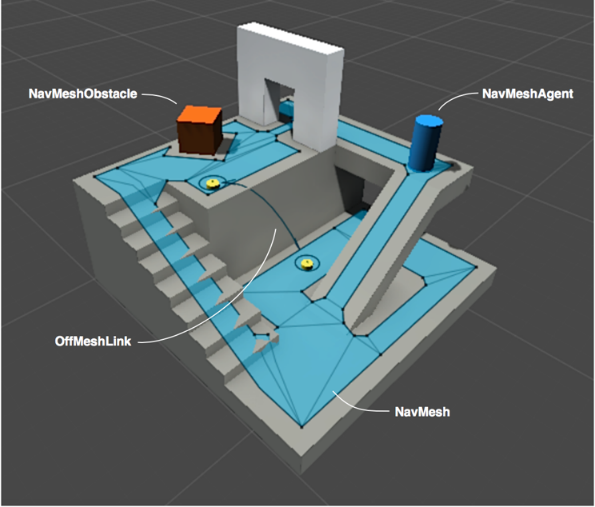
* Nombre: recursos predefinidos
* Descripción: recursos que el entorno provee de manera predefinida
* Tipo de valor: texto

#### 3.1.2.1 Recursos predefinidos Unity

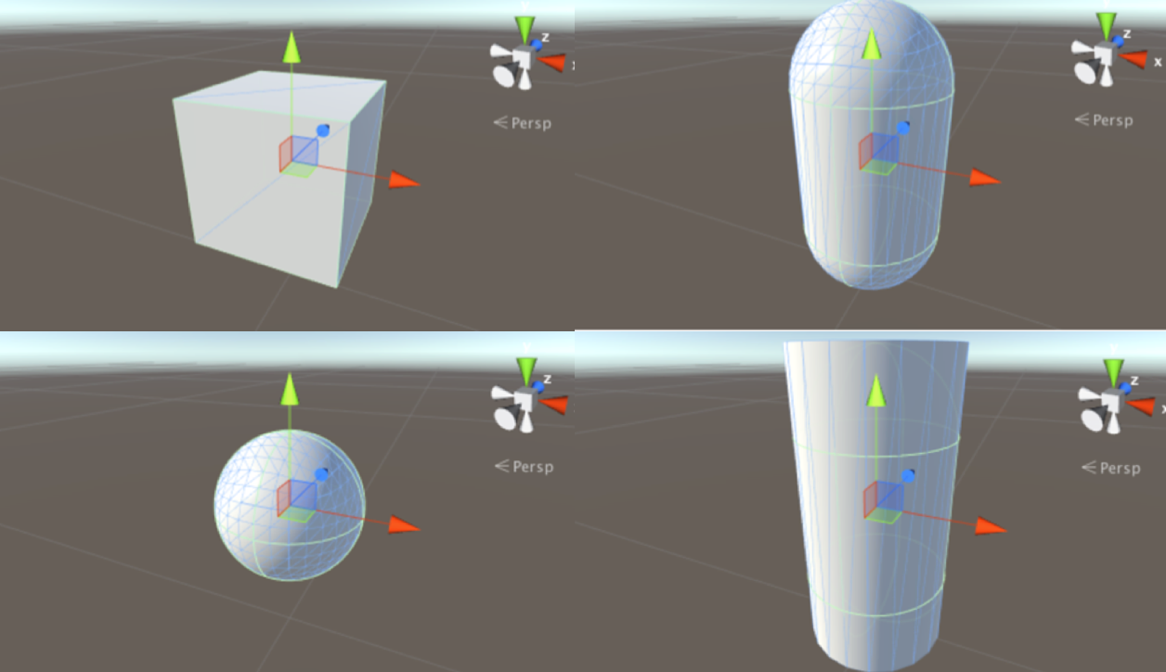
URL: <https://www.assetstore.unity3d.com/en/>

Con la descarga del programa, descargamos también una serie de assets predefinidos (Standard Assets). Estos son principalmente 2D, cámaras, personajes, efectos, entorno etc. Se pueden transferir entre proyectos usando paquetes Unity (Unity packages), que se pueden exportar e importar. Unity cuenta con una serie de componentes de interacción como son:

* Botones con eventos UnityEvents OnClick.
* Toggle: casilla de verificación **Is On** que determina si el toggle está actualmente apagado. Este valor es cambiado cuando el usuario haga clic . Éste también tiene un UnityEvent **OnValueCHanged** para definir lo que hará cuando el valor sea cambiado.
* Sliders (Deslizadores) para seleccionar valores mínimos y máximos.
* Barras de desplazamiento.
* Campos de texto.

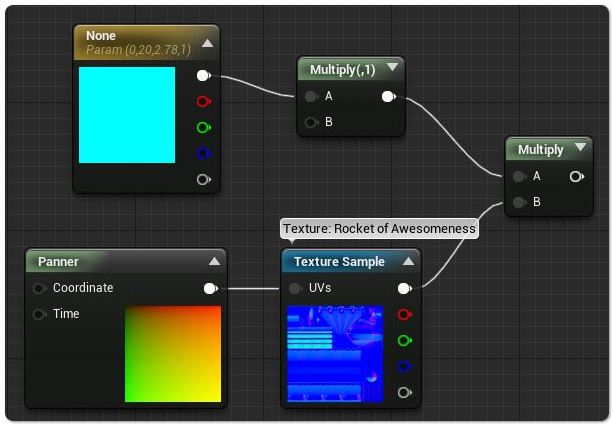
El sistema de navegación le permite crear personajes que se pueden mover de forma inteligente en el mundo del juego. El sistema de navegación utiliza mallas de navegación para razonar sobre el medio ambiente. Las mallas de navegación se crean automáticamente de su geometría de la escena. Los obstáculos dinámicos permiten alterar la navegación de los personajes en tiempo de ejecución, y fuera de la malla de vínculos le permiten construir acciones específicas, tales como la apertura de puertas, o saltando desde una cornisa. En esta sección se describe la navegación Unidad y la búsqueda de caminos en detalle.

Unity puede trabajar con modelos 3D de cualquier forma, se pueden crear con el software de modelado. Sin embargo, también hay una serie de tipos de objetos primitivos que pueden ser creados directamente dentro de Unity, por ejemplo cubos, esferas, cápsulas o cilindros.

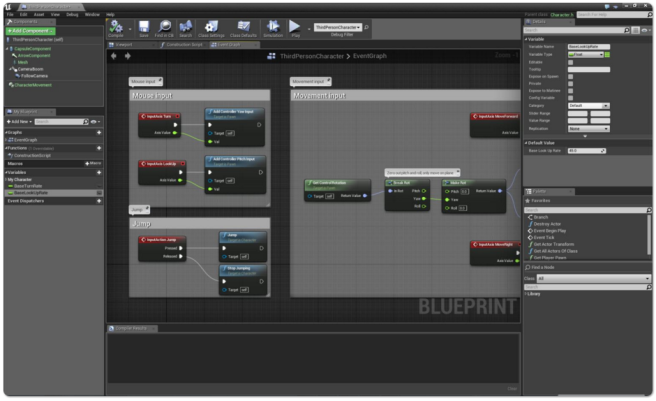


#### 3.1.2.2 Recursos predefinidos Unreal Engine

**Editor de materiales**, permite crear y editar los materiales que componen la escena del videojuego. Estos se definen como assets que pueden aplicarse a los objetos y terrenos, por ejemplo se puede crear el material “suciedad” y aplicarlo a los escenarios.

****

**Editor Blueprint**, es donde se puede trabajar y modificar planos especiales, estos se pueden utilizar para crear nuevos tipos de actores y eventos a nivel de guión, sin necesidad de escribirlos utilizando código C ++.

****

**Editor de comportamiento en árbol,** permite crear un script que programe la inteligencia artificial del videojuego mediante un sistema de nodos. Permite crear tareas, decorados, servicios etc.

**Editor de persona**, son las herramientas que permiten editar la animación de los personajes. Para ello se establecen distintas vistas, la de “esqueleto” para controlar el movimiento de sus huevos y articulaciones, la de “malla” para modificar los materiales que recubren el esqueleto. También tenemos la vista de “animación” para hacer los movimientos de los personajes y la vista “gráfica”, donde se editan los planos de animación y las máquinas de estado.



## 3.2 CARACTERÍSTICAS GENERALES

### 3.2.1 Multiplataforma

* Nombre: multiplataforma
* Descripción: plataformas distintas para las que se puede desarrollar el videojuego
* Tipo de valor: texto

#### 3.2.1.1 Multiplataformas en Unity

El número de plataformas sobre las que puede correr un videojuego desarrollado en Unity está creciendo constantemente.

Para dispositivos **móviles**:

* Despliegue en un clic para Android, iOS, Windows Phone y Tizen.
* Gran cantidad de optimizaciones gracias a características como la agrupación de assets.
* Herramientas y flujos de trabajo 3D y 2D dedicados y fáciles de usar.

Para **realidad virtual y realidad aumentada**:

* Pipeline de renderizado altamente optimizado.
* Capacidades de iteración rápida del editor de Unity.
* Native Oculos Rift, Gear VR and Playstation VR están disponibles para desarrolladores Unity. Procimamente lo estarán Microsoft HoloLens y Sream VR/Vive.

Juegos de **escritorio**:

* Posibilidad de crear juegos de escritorio de alta calidad gracias al shader físico de Unity.
* Realtime Global Illumination de Enlightnen, una simulación de cómo se transporta la luz físicamente. Significa que es una manera de simular cómo la luz se pasa entre las superficies en la escena 3D, lo que mejora el realismo visual en el juego. No sólo eso, sino que también puede ayudar en la ambientación y gameplay de tu juego. Los algoritmos de GI toman en cuenta la luz que viene directamente de una fuente de luz (iluminación directa) y también la luz que se refleja por las superficies en la escena con diferentes materiales (iluminación indirecta).
* Suministra soporte para despliegues con un clic en plataformas PC, Mac y Linux.

Juegos para **consola**:

* Más fácil que nunca para desarrolladores independientes publicar para plataformas de consolas.
* PS4, Xbox One, PlayStation Mobile, PlayStation Vita y Wii U de forma gratuita.
* Los procesos de aprobación varían según la plataforma, la comunicación hay que establecerla directamente con los propietarios para solicitar más información y para acceder a las compilaciones de despliegue en consolas de Unity.

Para **web**:

* Plug-in web gratuito.
* WebGL de Unity altamente optimizado que puede ofrecer un rendimiento a velocidades nativas. Ya ha sido ampliamente utilizado para crear numerosos títulos comerciales de éxito.

Para **Smart Tvs:**

* Desarrollo para tvOS , Android Tv y Samsung Smart Tv.

#### 3.2.1.2 Multiplataformas en Unreal Engine

Unreal engine 4 está preparado para cualquier plataforma actual de videojuegos. Se puede **desarrollar** desde pc tanto en **Windows** como en **Mac** y los juegos que se creen en ella pueden ser generados para funcionar, utilizando el mismo código, en iOS, OS X, Playstation 4, Xbox One, Linux, Android y HTML5.

El sistema de **blueprints** es capaz de detectar sobre que dispositivos se está desarrollando el juego y adaptar los controles para tal, lo cual facilita el desarrollo.

Además, la implementación para iOS soporta las nuevas librerías gráficas para procesadores de 64 bits de Apple, con lo que el rendimiento en los nuevos dispositivos con pantalla retina sin implementar nada nuevo.

Todas las herramientas y complementos son libres de descarga y uso, solo es necesario crear un usuario en la web de Unreal Engine y descargar el entorno para el sistema operativo deseado. Además, si al usuario se le asocia una cuenta de GitHub, nos permitirá acceder a los repositorios privados para poder ver y descargar el código fuente completo en C++ tanto de la herramienta para Windows o Mac como la de los complementos que generan los binarios en los diferentes sistemas operativos.

### 3.2.2 Licencia

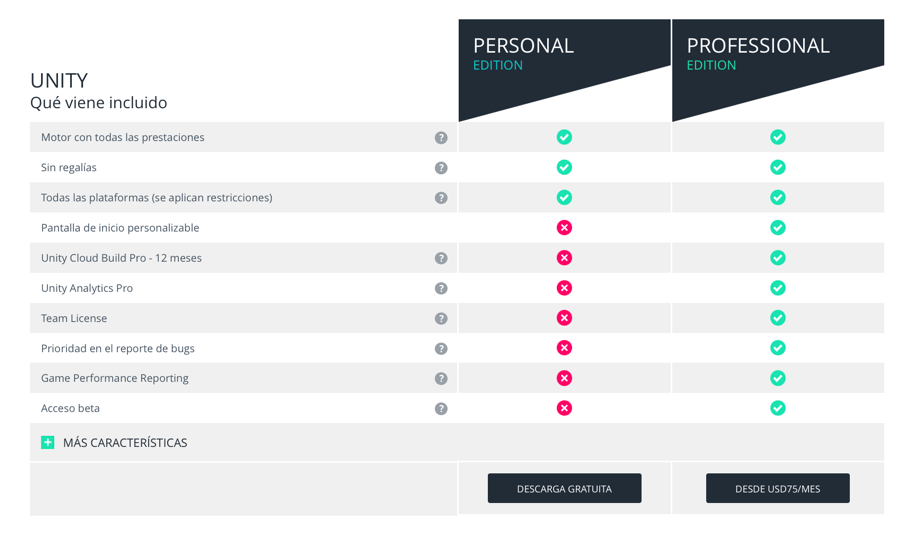
* Nombre: licencia
* Descripción: tipos de licencia del entorno
* Tipo de valor: texto

#### 3.2.2.1 Licencia en Unity

Unity ofrece diferentes tipos de licencia, con diferentes características:

Para **desarrolladores**:

* Licencia gratuita con una serie de características.
* Licencia de pago mensual con características más específicas.



Para **estudiantes:**

Acceso sencillo a la plataforma Unity a escuelas, estudiantes y docentes, todo ello con el conjunto completo de prestaciones profesionales del editor comercial de Unity. Además de la tecnología líder de la industria que es accesible a una amplia gama de usuarios, Unity suministra una amplia gama de tutoriales, ejemplos de proyectos de juego y recursos comunitarios.

* Descuentos especiales en cursos, certificados, contenidos y servicios para instituciones educativas
* Una biblioteca de aprendizaje de videos tutoriales y ejemplos de proyectos.
* Entradas con descuento para estudiantes y profesores a conferencias de desarrolladores de Unity.
* Una amplia comunidad en línea de usuarios comprometidos.
* Programa de licencias gratuitas a las instituciones educativas de primaria y secundaria y programas de aprendizaje.
* Kit de herramientas Unity.

Para **empresas**:

Las Soluciones Empresariales a la medida garantizan que Unity respalde plenamente las metas creativas y de negocios de la organización.

* Incluyen actualizaciones.
* Descuentos por volumen y de otro tipo.
* Licencias de códigos fuente.
* Soporte a medida bajo pedido.

#### 3.2.2.2 Licencia en Unreal Engine

Actualmente la licencia de uso de Unreal Engine 4 es **gratuita**, es decir, cualquier usuario puede descargarse el entorno con todas las librerías y herramientas disponibles.

Sin embargo, cuando se publica un juego en la tienda, Unreal cobra un 5% de los ingresos brutos, tras los primeros 3000$ obtenidos por el desarrollador, al trimestre. Esto implica que cuando un desarrollador publica un juego con Unreal Engine 4, una vez haya obtenido 3000$, deberá pagar a la empresa que está detrás de unreal un 5% de los ingresos brutos trimestrales.

## 3.3 RENDIMIENTO

### 3.3.1 Optimización

* Nombre: optimización
* Descripción: herramientas ofrecidas por el entorno para optimizar el videojuego
* Tipo de valor: texto

#### 3.3.1.1 Optimización en Unity

**Optimización del CPU**

Con el fin de que el CPU haga menos trabajo, es bueno reducir la cuenta de los objetos visibles. Unity cuenta con **Draw call batching**. Para dibujar un objeto en la pantalla, el motor tiene que emitir un draw call (llamado de dibujo) al API de gráficas (e.g. OpenGL o Direct3D). Los Draw Calls a menudo son costosos, con el API de gráficas haciendo un trabajo significante para cada draw call, causando una sobrecarga en el rendimiento por el lado del CPU.

Esto en su mayoría es causado por los cambios de estado hechos entre los draw calls (e.g. cambiando a un material diferente), que causa pasos de validación costosos y de translación en el driver gráfico. Unity utiliza varias técnicas para hacer frente a esto: Static Batching (combinar objetos estáticos en grandes mallas, y hacerlas de una manera más rápida) y Dynamic Batching (para pequeñas mallas, transforma sus vértices en la CPU, agrupa varios y dibuja una sola vez).

**Rendimiento de Iluminación**

La iluminación que no es computarizada para nada siempre es la más rápida. **Ligthmapping** permite computar una iluminación estática solo una vez, en lugar de computarla cada frame. El proceso para generar un ambiente lightmapped toma solo un poco más que simplemente colocar una luz en la escena en Unity, pero va a correr mucho más rápido (2–3 veces por 2 luces por-píxel) y se verá mucho mejor ya que el lightmapper puede suavizar los resultados de una iluminación global.

Optimizar una iluminación por-píxel salva ambos CPU y GPU: el CPU tiene menos draw calls por hacer, y el GPU tiene menos vértices para procesar y píxeles por rasterizar para todos estos renderizadores de objetos adicionales. Unity cuenta con las Quality Settings que sirven para modificar cuántas luces terminan como luces de píxel y cuántas como luces de vértice. Cada luz calcula su importancia basada en qué tan lejos está desde el mesh y qué tan intensa su iluminación es. Adicionalmente, algunas luces son más importantes que otras puramente del contexto del juego. Por esta razón, cada luz tiene un ajuste Render Mode que puede ser establecido a Important o Not Important; las luces marcadas como Not Important van a típicamente tener una carga menor de renderización.

**GPU: Compresión de Textura y Mipmap**

Utilizar **Compressed Textures** va a disminuir el tamaño de las texturas (resultando en unos tiempos de carga más rápidos y huellas de memoria más pequeñas) y también puede aumentar el rendimiento de renderización. Las texturas comprimidas utiliza solo una fracción del ancho de banda de memoria necesitado para texturas. De la misma forma, habilitar Generate Mip Maps para texturas utilizadas en escenas 3D puede ayudar limitar la cantidad de datos de textura transferida cuándo el GPU está renderizando, una textura mip mapped va a habilitar el GPU para utilizar una textura de baja-resolución para triángulos pequeños.

**LOD y Distancias de Eliminación Por-Capa (Per-layer Cull Distances)**

**Level of Detail** permite que elementos pequeños puedan hacerse invisibles en largas distancias, mientras que los objetos más grandes todavía serían visibles. De esta forma se reduce la carga de CPU y GPU.

La eliminación selectiva de oclusión (**Occlusion Culling**) es una característica que desactiva la renderización de objetos cuando estos no están vistos por la cámara ya que son oscurecidos por otros objetos.

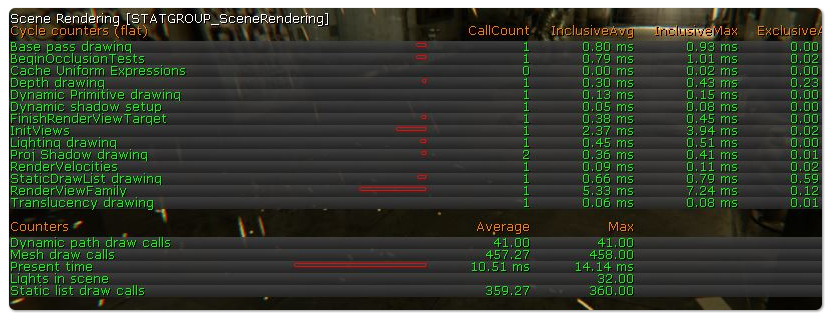
#### 3.3.1.2 Optimización en Unreal Engine

Cualquier videojuego tiene muchos objetos que se ejecutan de manera concurrente y que constan de múltiples características como material, iluminación, colisión, actualización etc. Es necesario optimizar y renderizar el juego lo máximo posible con el fin de que funcione correctamente. Para ello Unreal consta de diversas herramientas de las cuales explicaremos las más destacadas a continuación.

**CPU Profiling**

El rendimiento de la CPU puede verse comprometido cuando hay muchos objetos que tratar, por ello es necesario optimizar los hilos de procesamiento.

Además la CPU debe mandar los comandos de la GPU para tratar los gráficos. Para ayudar al desarrollador a disminuir todos estos posibles problemas Unreal contiene una consola de renderizado donde el desarrollador podrá ver todas las llamadas y características de la escena con el fin de optimizarlos.

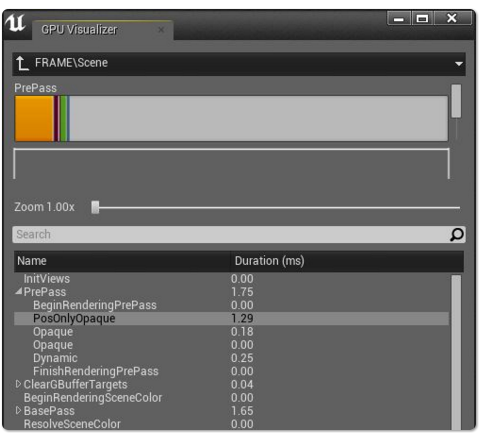


**Configuración para escalabilidad**

Unreal tiene un editor para la configuración de la escalabilidad mediante atajos que incluyen las configuraciones más comunes. Dentro de dicho editor se diferencian distintos apartados que se pueden configurar independientemente, vamos a destacar algunos. Se puede configurar la **resolución**, permitiendo definir los gráficos del videojuego en distintas calidades con el fin de que consumir menos recursos de cara a equipos con características bajas. Algo importante también es lo que se denomina **distancia de la vista,** en base a la posición del espectador del juego se pueden eliminar objetos que no estén dentro de la vista con el fin de reducir la necesidad de recursos gráficos. Otras características configurables son el **suavizado**, las **texturas** o los **efectos.**

**GPU Profiling**

La GPU tiene muchas unidades trabajando en paralelo por lo que se destina cada una a una parte de la estructura distinta. Normalmente los cuellos de botella se producen en el tratamiento del escalado de los elementos y la gestión de las sombras entre otros. Con la consola de GPU se permite al desarrollador ver cual provoca más costes y, por tanto, supone un cuello de botella para el juego ya que atasca la ejecución de otros elementos. Una vez detectados los problemas se pueden modificar los gráficos para conseguir disminuir estos costes de recursos y, por tanto, intentar eliminar los atascos.



## 3.4 UTILIDAD

### 3.4.1 Compatibilidad con programas externos

* Nombre: compatibilidad
* Descripción: tipos de archivos aceptados para importar al entorno y trabajar en el proyecto
* Tipo de valor: texto

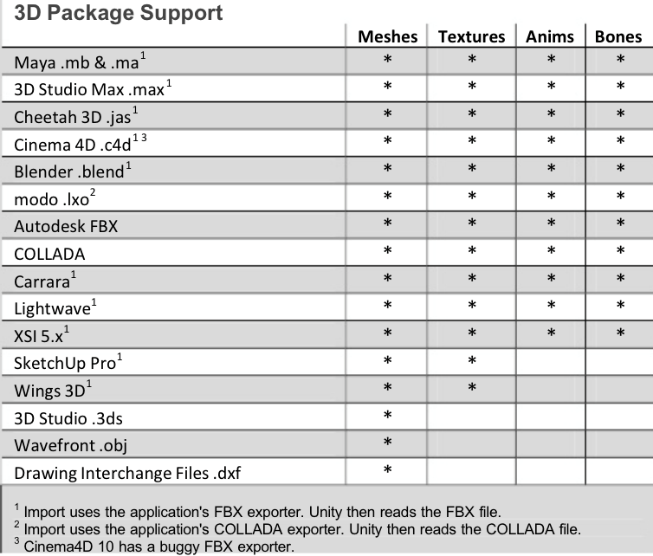
#### 3.4.1.1 Compatibilidad en Unity

<http://unityspain.com/topic/10633-exportar-modelo-de-unity-a-programa-3d-externo/>

<http://www.pablomarcos.net/siempre-es-mejor-utilizar-fbx-antes-que-obj-en-unity3d/>

Unity admite diferentes formatos de mallas, OBJ, FBX, Blend, MA, MB, MAX… Y por supuesto cada modelador trabajará con una aplicación 3D distinta, ya sea Maya, Blender, Strata, Lightwave, 3DS Max, Cinema4D y muchos más.

Para importar una malla Propietaria en Unity (como MA, MB, MAX o Blend) se necesita tener instalado el software 3D apropiado a la hora de importarla. Para archivos MA y MB se necesita tener Maya instalado, para archivos MAX se necesita 3DS MAX instalado, para archivos Blend, Blender, y lo mismo con el resto de formatos de archivo Propietarios. Esto es porque el importador de mallas de Unity lo que realmente hace es exportar el archivo Propietario a un archivo FBX sin que nos enteremos. Durante la importación, Unity carga el software de modelado 3D asociado a ese archivo y utiliza su exportador de FBX interno para obtener una versión de ese archivo en FBX. Por lo tanto, para Unity las mallas Propietarias crean una dependencia con el software que han sido modeladas ya que Unity necesita ese software para importar la malla. Si el software no está presente durante la importación, entonces la importación fallará. Una vez importada la malla, el software con el que ha sido modelada ya no es necesario.

A continuación se muestran los formatos soportados:

**Imágenes:** Photoshop .psd y .tiff se importan con capas acopladas de forma automática. También soporta JPEG, PNG, GIF, BMP, TGA, IFF, PICT y otros muchos formatos.

**Audio y vídeo:** MP3 y archivos de audio Ogg Vorbis .Ogg son compatibles de forma nativa, dependiendo de la plataforma. En las plataformas móviles, el audio se convierte en MP3 para tener un uso completo de la descompresión de hardware. AIFF, WAV y otros formatos de audio también son compatibles, ideal para los efectos de sonido. En Unity Editos se puede configurar una compresión . Son totalmente compatibles los formatos MOD, IT, archivos de seguimiento S3M, XM. Theora .ogg es soportado de manera nativa. Los formatos de vídeo MOV, AVI, ASF, MPG, MPEG, archivos MP4video se recodifican.

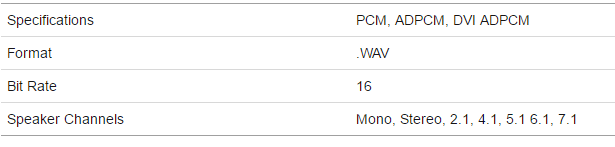
**Otros formatos soportados:** archivos XML y .txt pueden ser referenciados en tiempo de ejecución. Cualquier otro tipo de archivo, como RTF y DOC, se pueden utilizar para las notas de proyectos y listas de tareas pendientes.

#### 3.4.1.2 Compatibilidad en Unreal Engine

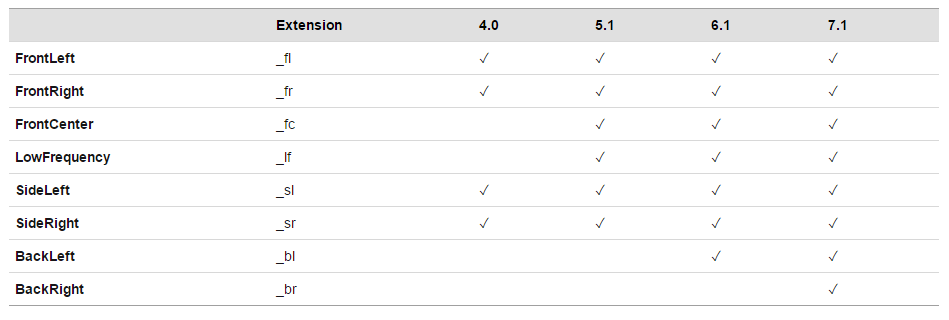
Hablamos de compatibilidad en el sentido de poder importar archivos de programas externos al entorno de Unreal para poder utilizarlos como componentes dentro del videojuego. Se pueden importar distintos tipos de archivos que vamos a explicar a continuación.

En cuanto a los modelos 3D se debe importar en formato fbx, dicho formato lo exportan la mayoría de sistemas de modelado 3D por lo que no resulta complicado. Unreal tiene un navegador de contenido para importar que detecta automáticamente el tipo de archivo importado. Una vez abierto se despliega un menú en base al tipo de objeto diferenciando entre, malla estática, malla esqueleto o animación. Dentro de estos menús se pueden configurar luces, texturas, colisiones etc.

Para los archivos de audio se pueden importar pequeños archivos de sonido de 16 bits sin comprimir en cualquier frecuencia de muestreo, aunque se recomiendan frecuencias de muestreo de 44100 Hz o 22050 Hz.



También se pueden importar sonidos base con distintas características, según la siguiente tabla:



### 3.4.2 Servicios

* Nombre: servicios
* Descripción: servicios de análisis ofrecidos por el entorno
* Tipo de valor: texto

#### 3.4.2.1 Servicios en Unity

Unity trae una creciente gama de servicios gratuitos para ayudar a los desarrolladores hacer juegos y participar, retener y rentabilizar el público. Algunos de estos servicios son:

* **Unity Ads** es una herramienta de monetización para los juego. Es una red de publicidad móvil que ofrece beneficios líderes en el mercado. La red sólo muestra los anuncios de videojuegos a gamers, lo que los hace mucho más relevantes. Eso es muy bueno para los anunciantes que buscan impulsar las descargas de su juego y para los editores que buscan generar ingresos con la venta de publicidad.
* **Unity Analytics** es una sencilla pero potente plataforma de datos que proporciona análisis para los juego de Unity, averiguar quiénes son los jugadores y su comportamiento en el juego. Plataformas compatibles: iOS, Android, Windows Phone 8.1, Windows Store 8.1 (escritorio), Windows Store 10.0 (Escritorio), Mac, PC, Linux, Web Player (Sólo disponible con el SDK), WebGL.
* **Unity IAP** hace que sea fácil de implementar compras en su aplicación a través de las más populares tiendas de aplicaciones. Las plataformas de apoyo incluyen: App Store de iOS, Mac App Store, Google Play y Windows Store.

#### 3.4.2.2 Servicios en Unreal Engine

En Unreal Engine se pueden configurar distintos tipos de herramientas de análisis según el tipo de videojuego o las estadísticas que se quieran obtener.

**Blueprint analytics plugin**

Esta API está integrada en el entorno de Unreal engine pero debe ser habilitada como complemento, permite ejecutar análisis sobre el juego durante el desarrollo. El plug-in traduce las llamadas y los envía al proveedor de análisis por defecto que se haya registrado para el proyecto. Una vez iniciado el análisis para una sesión de usuario se pueden registrar los eventos que se producen y ver su frencuencia, característica, atributos etc.

Para utilizar la API anterior debe escogerse un proveedor de los cuatro ofrecidos por Unreal, se diferencian en el tipo de plataforma sobre la que se va a ejecutar el juego:

* Apsalar, es el analizador para IOS. Comenzó como un servicio de análisis gratuito que permite reportar publicidad a Facebook, aunque recientemente se ha pasado a tener una cuota. Lo más destacado es que se puede medir el retorno de la inversión para los anuncios para móviles de Facebook, algo poco común. Para utilizarlo, el desarrollador se debe registrar, obtener una clave de aplicación y la contraseña que la identifica de forma exclusiva. Después descargar las bibliotecas que se compilan en el plugin Apsalar y colocarlas en el directorio habilitado.
* Flurry analytics provider, es otro analizador gratuito para IOS que se utiliza para comparar los datos con otros proveedores.
* File logging analytics provider, se utiliza para escribir las llamadas de la API de análisis en el disco en formato JSON, permitiendo depurar el proceso de análisis. Se escribe los datos en archivos propios en la carpeta Analytics permitiendo comparar los datos con los producidos por los eventos de consola.
* Multicast analytics provider, permite utilizar varios proveedores a la vez sin necesidad de ejecutarlos manualmente. Esto permite tener todas las cosas buenas de cada uno de los proveedores.

## 3.5 PROGRAMACIÓN

### 3.5.1 Lenguaje

* Nombre: lenguaje
* Descripción: lenguajes de programación que se pueden utilizar en el entorno
* Tipo de valor: texto

#### 3.5.1.1 Lenguaje en Unity

<http://academiaandroid.com/scripts-lenguajes-programacion-unity/><http://developeando.net/unity3d/>

Podemos controlar el comportamiento de los GameObjects en Unity asignándoles componentes predefinidos y configurando sus propiedades a través de una interfaz gráfica. A pesar de su versatilidad, es posible que estos componentes sean insuficientes si queremos crear una mecánica de juego más personalizada o compleja.

Para conseguir ese objetivo, Unity nos ofrece la posibilidad de implementar *scripts*. Un *script* es un pequeño programa que podemos asociar a un GameObject como si se tratara de un componente más, proporcionándole una funcionalidad concreta que definimos nosotros. Podemos escribirlos en alguno de los lenguajes que Unity soporta de forma nativa: C#, UnityScript y Boo.

**C#**: Es el lenguaje que prima el propio motor de juegos Unity, ya que le da un soporte especial y basa toda su documentación en él.

* Es el más utilizado, con mucha diferencia, por la comunidad de usuarios de Unity, lo que facilita luego el aprovechamiento de scripts creados por terceros en nuestros propios videojuegos.
* Su similitud con Java, lo hacen muy fácil de utilizar por aquellos que trabajan en el entorno Android.

**Boo**: Lenguaje de programación orientado a objetos, de tipos estáticos para la Common Language Infrastructure, con una sintaxis inspirada en Python y un énfasis en la extensibilidad del lenguaje y su compilador. Ha sido descontinuado a efectos prácticos en las nuevas versiones de Unity, ya que no pueden crearse scripts en este lenguaje directamente en el editor. Esta falta de soporte, debida según Unity al pequeño porcentaje de usuarios que lo utilizan, hace que sea mejor descartarlo.

 **Unity Scripting** es una ‘versión’ particular de Javascript. Aunque más utilizado que Boo, está lejos de las cifras de C# y la falta de documentación oficial es una señal a tener en cuenta.

#### 3.5.1.2 Lenguaje en Unreal Engine

El lenguaje de programación utilizado para escribir el código del juego es C++, dicho lenguaje nos permite controlar de forma muy minuciosa la gestión de memoria dentro del desarrollo de nuestro juego. No obstante, es un lenguaje mucho más complicado que otros lenguajes orientados a objetos como pueden ser C# o Java.



Unreal Engine nos permite utilizar código C++ plano, sin depender de las librerías que nos proporciona el motor, no obstante en caso de que decidamos utilizar estas, la programación se simplificará enormemente debido a la cantidad de métodos que ya trae incorporados. Dichas librerías se explicarán en un punto posterior.

En adición al lenguaje de programación C++, Unreal Engine, nos trae un sistema de scripting visual llamado Blueprint Scripting, el cual nos permite realizar tareas de programación desde un entorno visual con solo clickar y arrastrar nodos. Aunque este no sea un lenguaje de programación al uso, debemos hacerle mención ya que suele ser usado ampliamente para facilitar las tareas de los programadores al realizar tareas más sencillas o bien los propios programadores pueden programar scripts para ser usados en este lenguaje y facilitar que el resto del equipo pueda “programar” ciertas partes de la aplicación de forma autónoma.

### 3.5.2 API

* Nombre: API
* Descripción: descripción de la API básica disponible en el entorno
* Tipo de valor: texto

#### 3.5.2.1 API de Unity

Unity cuenta con una serie de clases predefinidas, además de las que podamos crear. Estas clases aportan funcionalidades que facilitan la creación de videojuegos. Algunos ejemplos de estas clases son:

* **MonoBehaviour**: Clase base para todos los nuevos scripts. Proporciona una lista de todas las funciones y eventos que están disponibles para scripts estándar adjuntos a Game Objects. Importante para cualquier interacción o control sobre objetos individuales en el juego.
* **Transform**: Cada Game Object tiene una posición, rotación y escala en espacio (ya sea 3D o 2D), y esto es representando por el componente Transform. Además de proporcionar esta información, tiene muchas funciones útiles que pueden ser utilizadas para mover, escalar, girar, re-apadrinar y manipular objetos, al igual que convertir coordenadas desde un espacio a otro.
* **Rigidbody/ Rigidbody2D:** Para la mayoría de elementos del gameplay, el motor proporciona un conjunto de herramientas más fácil para mover objetos, detectando triggers y colisiones, y aplicando fuerzas. La clase Rigidbody (o su equivalente en 2D, Rigidbody2D) proporciona todas las propiedades y funciones necesarias para jugar con la velocidad, masa, fricción, fuerza, colisión, etc.

Además de estas clases Unity ofrece una serie de APIs que facilitan otras funciones del desarrollo de un videojuego, como son la integración con redes sociales, la jugabilidad o las capacidades multijugador.

**API Social**

Es el punto de acceso de Unity para características sociales, tal como:

* Perfiles de usuario
* Lista de amigos
* Logros
* Estadísticas / Clasificaciones

Proporciona una interfaz unificada a diferentes back-ends sociales, tal como XBox Live o el GameCenter, y es utilizado principalmente por programadores en el proyecto del juego.

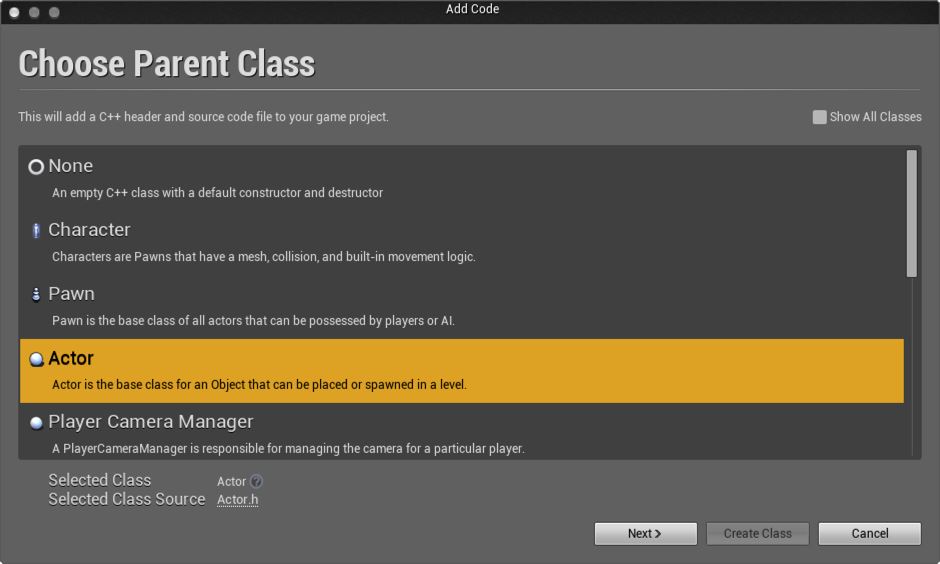
**Playable API**

Esta nueva API proporciona una forma de crear herramientas, efectos u otros mecanismos de juego mediante la organización y la evaluación de las fuentes de datos en una estructura de árbol.

**API de Alto Nivel**

(HLAPI) es un sistema para construir capacidades multijugador para juegos de Unity. Maneja las tareas comunes que son requeridas para juegos multijugador.

#### 3.5.2.2 API de Unreal Engine

El API de Unreal Engine nos proporciona una gran variedad de clases predefinidas las cuales podemos extender, además de poder crear una clase en C++ vacía, también tenemos la posibilidad de crear clases con distinta funcionalidad base pre implementada para que después nosotros podamos modificarlas o extenderlas

Entre las clases predefinidas podemos destacar las siguientes:

* Actor: es la clase base de objetos, se usa para cualquier objeto que pueda ser colocado en un nivel, es decir, objetos con un cuerpo físico no son clases lógicas.
* Pawn/Peón: es la clase básica de objetos que van a usarse, ya sean por medio del jugador o por una IA.
* Character/Personaje: es una clase que desciende de la clase peón pero que tiene preconstruido una malla de renderizado, un detector de colisiones y un sistema de movimiento.
* Player Camera Manager: es una clase que es la responsable de gestionar la cámara a la que está asociada.
* Game Mode: es una clase que nos permite programar y gestionar la lógica del juego.
* HUD: esta clase nos permite programar la interfaz que se va a mostrar al usuario durante el juego.

Además de las distintas clases precreadas podemos destacar una serie de métodos que son transversales a todas las clases:

* AMyActor(); (sirve para poner valores por defecto al actor en cuestión)
* Virtual void BeginPlay() override; (se usa para establecer lo que queramos que haga cuando el objeto entra el juego o cuando se carga la partida)
* Virtual void Tick (float DeltaSeconds) override; (la función se llama a cada frame del juego)

## 3.6 GRÁFICOS

### 3.6.1 Renderizado

* Nombre: renderizado
* Descripción: herramientas de renderizado proporcionadas por el entorno
* Tipo de valor: texto

#### 3.6.1.1 Renderizado en Unity

La palabra renderización proviene del inglés render, y no existe un verbo con el mismo significado en español, por lo que es frecuente usar las expresiones renderizar o renderear.  El término **rendering** también es usado para describir el proceso del cálculo de los efectos en la **edición de archivos de videos (pre-renderizado)** para producir una salida final de video.

Sin embargo, el uso en los videojuegos se habla de renderización en tiempo real, ya que los movimientos que pueda hacer el jugador no son predecibles y por esto se deberán generar en tiempo real con ayuda de una gpu para soportar la alta complejidad y cantidad de cálculos. Existen distintos **modos** de renderizado en Unity que permiten elegir si el objeto utiliza la transparencia, y si es así, qué tipo de mezcla va a utilizar.

* **Opaco**: Es el valor predeterminado, y es adecuado para objetos sólidos normales sin zonas transparentes.
* **Recorte**: Le permite crear un efecto de transparencia que tiene bordes duros entre las áreas opacas y transparentes. En este modo, no hay áreas semitransparentes, la textura es o bien 100% opaco, o invisible. Esto es útil cuando se utiliza la transparencia para crear la forma de materiales tales como hojas, o una tela con agujeros y los suelos.
* **Transparente**: Adecuado para la representación realista de materiales transparentes tales como plástico transparente o de vidrio. En este modo, el propio material se enfrentarán a los valores de transparencia (basado en el canal alfa de la textura y de la alfa del color de tinte), sin embargo reflexiones y una excelente iluminación permanecerá visible a plena claridad como es el caso de materiales transparentes reales.
* **Fade**: Permite que los valores de transparencia a desaparecer por completo un objeto, incluso de cualquier reflejos especulares o reflexiones que pueda tener. Este modo es útil si se desea animar un objeto aparición o desaparición paulatina. Se ISS no adecuado para la representación realista de materiales transparentes, tales como vidrio o plástico transparente debido a las reflexiones y pone de relieve también se desvanecieron.

#### 3.6.1.2 Renderizado en Unreal Engine

El sistema de renderizado de Unreal Engine está basado en DirectX 11 lo que incluye Sombreado diferido, Iluminación global, translucidad ligera y simulación de partículas mediante campos vectoriales postprocesados por la GPU.



**Sombreado diferido:**

Todas las luces son aplicadas en diferido en Unreal Engine 4. Los materiales tienen escritos sus atributos en el código lo que influye como la luz pasa a través de ellos e iluminan el escenario.

**Iluminación global:**

El sistema de Iluminado de Unreal Engine 4 se basa en varias herramientas que podemos utilizar para iluminar el mundo que varían en función de la calidad y el rendimiento que busque el desarrollador. Dichas luces pueden ser totalmente dinámicas, parcialmente estáticas y totalmente estáticas.

**Translucidad ligera:**

La translucidad emite sombras provenientes de su propio efecto y de otras iluminaciones además de garantizar el correcto fusionado de las distintas luces.

**Partículas simuladas por GPU:**

Unreal Engine 4 simula partículas en la GPU, comparado a la simulación tradicional que se carga sobre la CPU permite una simulación de cientos de miles de partículas comparadas con simplemente unas pocas miles de la simulación tradicional.

**Efectos de PostProcesado:**

Unreal Engine cuenta con varios efectos de post procesado que permiten a los diseñadores alterar el ambiente de la escena. Entre estos efectos se incluye: Oclusión Ambiental, Luces de Ambiente, Adaptación Ocular, Flashes, etc.

### 3.6.2 Materiales y shaders

* Nombre: materiales
* Descripción: materiales y shaders disponibles y su configuración
* Tipo de valor: texto

#### 3.6.2.1 Materiales y shaders en Unity

El renderizado en Unity se hace con **Materials**, **Shaders** y **Textures**.

Hay una relación cercana entre Materiales, Shaders y las Texturas en Unity.

**Materials** son definiciones acerca de cómo la superficie debería ser renderizada, incluyendo referencias a texturas utilizadas, información del tiling (suelo de baldosas), tines de color y más. Las opciones disponibles para un material depende en qué shader del materia está utilizando.

**Shaders** son scripts pequeños que contienen los cálculos de matemáticas y algoritmos para calcular el color de cada pixel renderizado, basándose en el input de iluminación y la configuración del Material.

**Textures** son imágenes bitmap. Un material contiene referencias a texturas, para que el shader del Material puede utilizar las texturas mientras calcula el color de la superficie de un objeto. Adicionalmente al color básico (albedo) de la superficie de un objeto, las texturas pueden representar otros aspectos de la superficie de un material tal como su reflectividad o rugosidad.

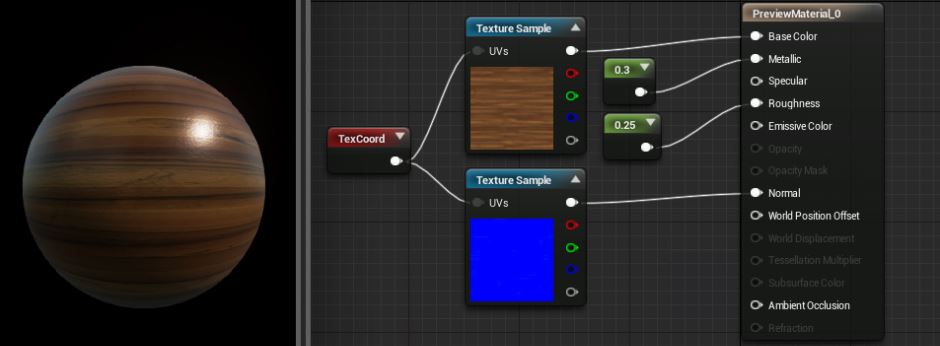
Un material especifica un shader en ser utilizado, y el shader utilizado determina qué opciones están disponibles en el material. Un shader especifica una o más variables de textura que espera utilizar, y el inspector del Material de Unity le permite a usted asignar sus propios assets de textura a estas variables de textura.

Para la mayoría del renderizado normal - entendiendo por tal personajes, escenario, entorno, objetos sólidos y transparentes, superficies suaves y duras etc., el [Standard Shader](http://docs.unity3d.com/es/current/Manual/shader-StandardShader.html) por lo usual es la mejor decisión. Este es un shader altamente personalizado el cual es capaz de renderizar muchos tipos de superficie en una manera bastante real.

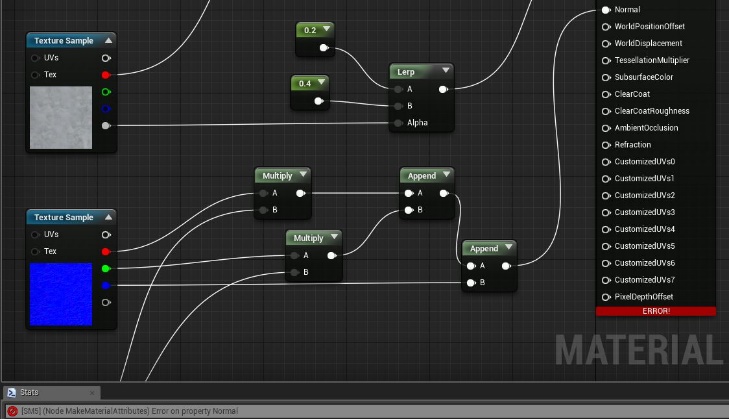
Hay otras situaciones dónde un diferente shader integrado, o incluso un shader escrito personalizado podría ser apropiado - tal como líquidos, follaje, vidrio de refracción, efectos de partícula, caricaturesco, ilustrativo u otros efectos artísticos, otros efectos especiales como visión nocturna, visión de calor, o visión de rayos x, etc.

#### 3.6.2.2 Materiales y shaders en Unreal Engine

Un material es un objeto que puede aplicarse a un modelo para controlar como se ve en la escena. A grandes rasgos podemos decir que un Material es como la pintura que se aplica a un objeto. Un material define el tipo de superficie de la cual nuestro objeto va a aparentar estar hecho. Entre sus propiedades puedes definir su color, su brillo, su transparencia, etc.



Cuando la luz de la escena impacta en la superficie del objeto, es el material lo que se usa para realizar los cálculos de cómo reaccionará la luz con la superficie. Esto se hace mediante expresiones matemáticas e imágenes.

Unreal Engine nos permite construer materiales mediante un Sistema de scripting visual en el que a traves de una red de nodos, dentro del editor de materiales, seleccionamos sus propiedades. Este sistema de programación visual se denomina HLSL, el cual nos permite seleccionar cada parámetro del material individualmente para construirlo justo como lo queremos. La única desventaja de este editor de materiales es su alta complejidad ya que para aprender a usarlo deberemos mirar el manual y algunos cursos acerca de esta funcionalidad.

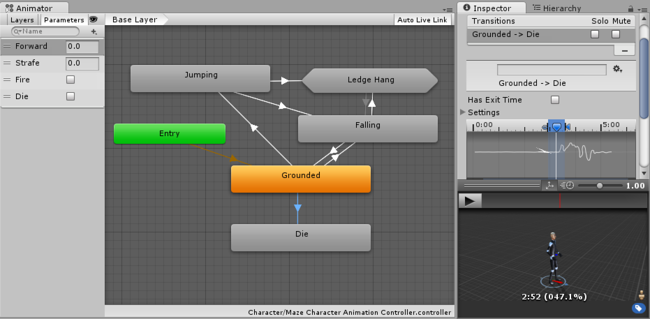
### 3.6.3 Animación 2D y 3D

* Nombre: animación
* Descripción: sistemas de animación ofrecidos por el entorno
* Tipo de valor: texto

#### 3.6.3.1 Animación 2D y 3D en Unity

En el caso de la animación haremos especial hincapié en la animación 3D ya que es la más compleja y en el caso de la animación 2D Unity nos proporciona la posibilidad de pasarle una hoja de scripts de un personaje 2D y automáticamente se crean las animaciones para los movimientos de este. Unity tiene un sistema de animación excelente y sofisticado llamado Mecanim que proporciona:

* Un flujo de trabajo y configuraciones de animaciones fácil para todos los elementos de Unity incluyendo objetos, personajes, y propiedades.
* Soporta para [animation clips](http://docs.unity3d.com/es/current/Manual/class-AnimationClip.html) importados y animaciones creadas dentro de Unity
* Animación humanoide [retargeting](http://docs.unity3d.com/es/current/Manual/Retargeting.html) - habilidad para aplicar animaciones del modelo de un personaje a otro.
* Un flujo de trabajo simplificado para alinear clips de animación.
* Una pre-visualización conveniente de clips de animación, transiciones e interacciones entre estos. Esto le permite a los animadores funcionar de manera más independiente a los programados, dar prototipos y pre-visualizar sus animaciones antes de que el código del juego es conectado.
* El manejo de interacciones complejas entre animaciones con una herramienta visual de programación.
* Animar diferentes partes del cuerpo con diferente lógica.
* Características de capas (layering) y de masking



**Flujo de trabajo de la Animación**

El sistema de animación de Unity está basado en el concepto de [Animation Clips](http://docs.unity3d.com/es/current/Manual/class-AnimationClip.html), los cuales contienen información acerca cómo ciertos objetos deberían cambiar su posición, rotación, u otras propiedades en el tiempo. Cada clip puede ser pensado como una sola grabación lineal. Los Animation clips de fuentes externas son creadas por artistas o animadores con herramientas de terceros como lo es Max o Maya, o proceden de estudios de captura de movimiento u otras fuentes.

Los Animation Clips (Clips de animación) son organizados entorno a un sistema con una estructura similar a la del diagrama de flujo llamado Animator Controller. El Animator Controller funciona como un “[State Machine](http://docs.unity3d.com/es/current/Manual/AnimationStateMachines.html)” que mantiene un seguimiento de qué clip debería actualmente estar reproduciéndose, y cuando las animaciones deberían cambiar o mezclarse juntas.

Un simple Animator Controller podría solamente contener uno o dos clips, por ejemplo para controlar un powerup de girar y rebotar, o para animar una puerta que se abre y se cierra en el tiempo correcto.

Un más avanzado Animator Controller puede contener docenas de animaciones humanoides para todas las acciones de los personajes principales, y pueden mezclarse entre múltiples clips al mismo tiempo para proporcionar un movimiento fluido a medida que el jugador se mueve alrededor de la escena.

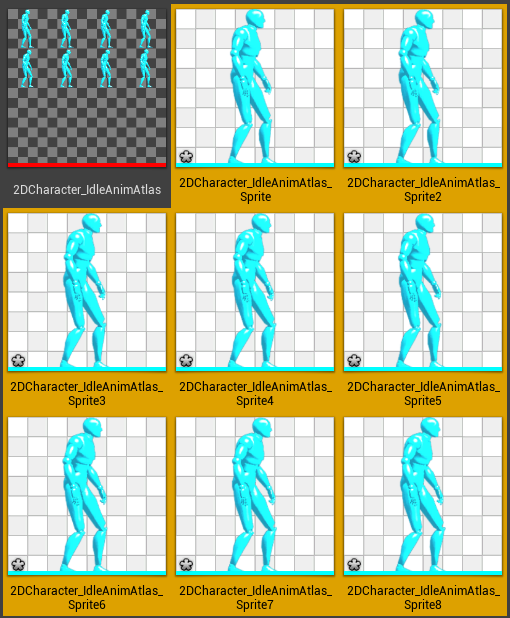
El sistema de animación de Unity también tiene un número de características especiales para manejar personajes humanoides los cuales le da a usted la habilidad de [retarget](http://docs.unity3d.com/es/current/Manual/Retargeting.html) una animación humanoide de cualquier fuente (Eg. captura de movimiento, el asset store, o alguna librería de animación de terceros) para su propio modelo del personaje, al igual que ajustar las [definiciones musculares](http://docs.unity3d.com/es/current/Manual/MuscleDefinitions.html). Estas características especiales están activadas por el sistema de [Avatar](http://docs.unity3d.com/es/current/Manual/class-Avatar.html) de Unity, dónde los personajes humanoide son mapeados a un formato interno común.

#### 3.6.3.2 Animación 2D y 3D en Unreal Engine

El Sistema de Animación de Unreal Engine, mezcla deformaciones basadas en esqueleto con deformaciones basadas en vértices para crear complejas animaciones. Este sistema puede hacer que movimientos básicos del jugador, se vean mucho más realistas reproduciendo y fusionando secuencias de animación, creando movimientos especiales personalizados como movimientos de escalada, aplicar efectos de daño e impacto e incluso crear expresiones faciales o controlar directamente la transformación de los huesos del modelo.



El motor nos permite controlar y configurar las animaciones de los personajes dentro del mismo, estableciendo los patrones de animaciones mediante programación con BluePrints o mediante un completo editor de animaciones. Dicho editor nos da las herramientas para crear nuestras propias animaciones a partir del modelo del personaje. Una vez que tenemos el modelo dentro del editor vamos moviéndolo y guardando las poses que queremos que compongan la animación, esto nos permite crear las animaciones de forma eficiente.

En cuanto a la animación 2D, Unreal nos permite crear animaciones mediante hojas de sprites. Dichas hojas contendrán todos los dibujos del personaje en las distintas posiciones, los cuales se unirán todos mediante el editor de sprites que nos proporciona unreal para crear la animación.

## 3.7 SONIDO

### 3.7.1 Recursos propios dentro del entorno

* Nombre: recursos propios
* Descripción: recursos propios referidos a sonido dentro del entorno
* Tipo de valor: texto

#### 3.7.1.1 Recursos propios dentro de Unity

El sistema de audio de Unity es flexible y poderoso. Puede importar la mayoría de formatos estándares de audio y tiene características sofisticadas para reproducir sonidos en un espacio 3D, opcionalmente con efectos como echo y filtración aplicadas. Unity también puede grabar audio de cualquier micrófono disponible de la máquina del usuario para uso durante el modo de juego o para almacenamiento y transmisión.

En la vida real, los sonidos son emitidos por objetos y escuchados por listeners. La manera en la que el sonido es percibido depende de una cantidad de factores. Un listener puede decir en qué dirección un sonido viene y también puede tener algún sentido de su distancia por su intensidad y calidad



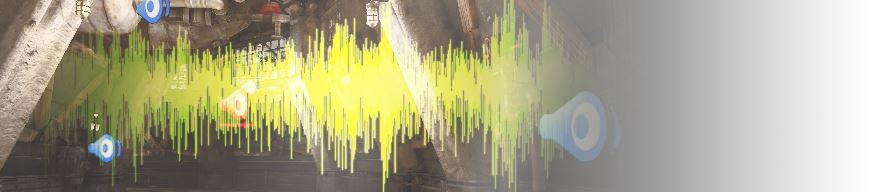
Para simular los efectos de la posición, Unity requiere que los sonidos sean originados de Audio Sources adjuntos a los objetos. Los sonidos emitidos luego son recogidos por un Audio Listener adjuntado a otro objeto, en la mayoría es la cámara principal. Unity puede simular los efectos de la distancia y la posición de una fuente del objeto listener y reproducirlos al usuario de acuerdo a esto.

Unity no puede calcular los echos solo con la geometría de la escena, pero usted puede simularlos al agregar Audio Filters a los objetos. Por ejemplo, usted puede aplicar el Echo Filter a un sonido que supuestamente viene dentro de una cueva. En situaciones dónde los objetos pueden moverse adentro y afuera de un lugar con un fuerte echo, usted puede agregar un Reverb Zone a la escena. Por ejemplo, su juego puede involucrar carros que se mueven a través de un túnel. Si usted coloca un reverb zone dentro del túnel, los sonidos de los motores de los carros va a comenzar hacer eco justo cuando entran y el echo morirá cuando ellos salgan al otro lado del túnel.

El Audio Mixer de Unity permite mezclar varias fuentes de audio, aplicarles efectos, y realizar una masterización.

#### 3.7.1.2 Recursos propios dentro de Unreal Engine

El Sistema de audio de Unreal Engine 4 está compuesto de varios componentes, cada uno de ellos trabajando en conjunto con todos los demás para construir la experiencia de audio que experimentará el jugador. Cuando importamos un archivo de audio en el motor y lo colocamos en el nivel, tenemos varias opciones para personalizar, desde las básicas como ajustar el volumen o el tono hasta algunas más complejas como la atenuación del sonido.



Además en Unreal Engine 4, podemos crear sonidos compuestos en forma de señales de audio mediante un editor de señales de audio que trae incorporado el propio motor. En el podemos combinar sonidos mediante nodos igual que hacíamos con los materiales para conseguir el resultado deseado.

En las últimas versiones de Unreal Engine, podemos encontrar que se ha añadido una mejora al sistema de audio. Dicha mejora consiste en la compatibilidad de sonido estéreo en los objetos del juego, antes cada objeto dentro del juego podía emitir solo un sonido. Con el nuevo sistema, un mismo objeto podrá emitir dos sonidos de forma simultánea los cuales se percibirán en estéreo por el jugador.



### 3.7.2 Programación del sonido dentro del juego

* Nombre: recursos propios
* Descripción: recursos propios referidos a sonido dentro del entorno
* Tipo de valor: texto

#### 3.7.2.1 Programación del sonido dentro del juego en Unity

El AudioMixer es un asset que puede ser referenciado por AudioSources para proporcionar un routing (direccionamiento) más complejo y mezcla de señales de audio generadas desde AudioSources. Hace esta categoria basándose en la mezcla vía la jerarquía del AudioGroup que es construido por el usuario dentro del Asset.

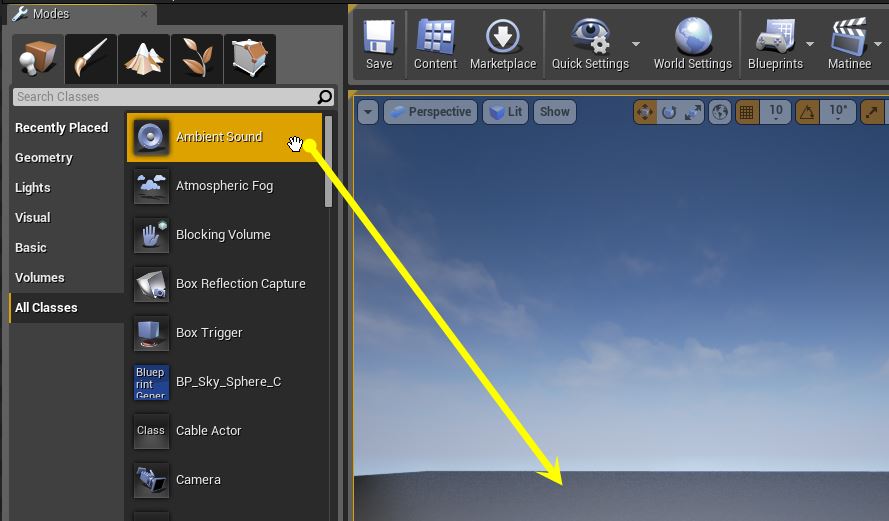
Efectos DSP y otros conceptos de masterización de audio pueden ser aplicados a la señal del audio como si fuera direccionado desde el AudioSource al AudioListener.

Con objeto de facilitar la comprensión de la explicación recientemente expuesta en el trabajo añadimos la definición del concepto enroutamiento:

“El Enrutamiento de Audio es el proceso de tomar un número de señales de audio de input y output.” El término señal aquí se refiere a un flujo continuo de datos de audio digitales, que se puede descomponer a canales de audio digitales (como estéreo o 5.1).

Internamente hay algo de trabajo en estas señales siendo hechas, tal como la mezcla, la aplicación de efectos, atenuación etc. Por varias razones que serán cubiertas, esto es un aspecto importante del procesamiento de audio y esto es lo que el AudioMixer está diseñado para permitirle a usted hacer.

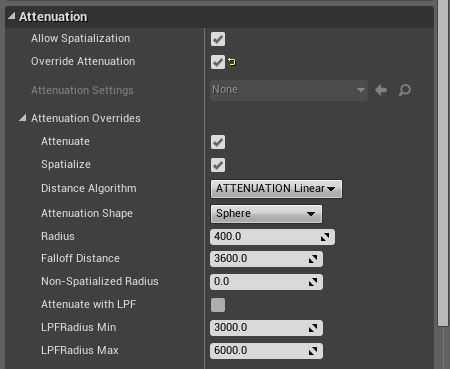
#### 3.7.2.2 Programación del sonido dentro del juego en Unreal Engine

Unreal Engine 4 maneja el proceso por cual podemos producir o modificar sonidos a traves del uso del Actor de Sonido. Cuando una pista de sonido o una señal de sonido son colocadas en el nivel, se crea un Actor de Sonido.

El actor de sonido toma un gran papel a la hora de transmitir al jugador ya que reproducirá el sonido con más o menos volumen en función de la distancia a la que nos encontremos de la fuente del sonido.

Una vez que colocamos el sonido podemos configurarlo de múltiples formas en función de nuestras necesidades:

* Si necesitamos que se empiece a reproducir nada más comience el juego, dejaremos la función de AutoPlay activada.
* También podemos hacer que el sonido se reproduzca en bucle en función de lo que hayamos establecido en la opción looping.
* Por último, podemos configurar una gran cantidad más de parámetros para que el sonido se adapte a nuestras necesidades.



## 3.8 RECURSOS

### 3.8.1 Formación externa

* Nombre: formación externa
* Descripción: formación externa a la empresa desarrolladora del entorno
* Tipo de valor: texto

### 3.8.1.1 Formación externa en Unity

Para este apartado haremos mención de los cursos que expusimos y analizamos sobre la tecnología del motor gráfico Unity en el primer trabajo grupal, por lo que si se desea más información acerca de estas opciones de formación externa solo tendrá que acceder a este documento.

<https://www.coursera.org/learn/desarrollo-videojuegos-unity>

Curso desarrollado por la universidad de los Andes, en este curso se desarrollaran las habilidades básicas para crear un juego en 2D.

<https://www.tutellus.com/tecnologia/videojuegos/creacion-de-videojuegos-con-unity-3d-3291>

Curso enfocado a personas con experiencia básica previa en este entorno, tiene como bases ayudarte a mejorar animaciones, programación, tratamiento de sonido y el exportado multiplataforma.

<http://3dmotive.com/series/intro-to-unity-5.html>

Este curso pertenece a la rama de los no gratuitos, esta impartido por 3DMotive y enfocado a usuarios novicios en este entorno y tecnología, su temario avanza desde lo más básico como puede ser la interfaz de usuario hasta tareas más complejas como puede ser la animación.

#### https://upload.wikimedia.org/wikipedia/en/b/b1/Official-company-logo-for-lynda.com_400x400px.png3.8.1.2 Formación externa en Unreal Engine



Como hemos indicado ya en otros trabajos, hay mucha formación en internet acerca de Unreal Engine, no obstante, cabe destacar que la mayor parte de esa formación es formación de pago, es decir, cursos de pago. Entre los proveedores que ofrecen cursos de pago para Unreal Engine, podemos destacar famosas empresas especializadas en formación como Digital Tutors, Udemy o Lynda.com.

Aunque haya mucha formación no gratuita acerca de Unreal Engine en Internet, este motor cuenta con una gran comunidad detrás y si buscamos un poco podemos encontrar foros, cursos en plataformas de videos como Youtube, blogs especializados, etc.

Aparte de la formación que podamos recibir online, en casi todos los másteres universitarios dedicados al desarrollo de videojuegos se suele aprender a manejar este motor debido a su alta popularidad en el mercado. Esto no solo se aplica a los másteres sino también a los grados especializados en el desarrollo de videojuegos siendo más notable su enseñanza en las academias privadas.

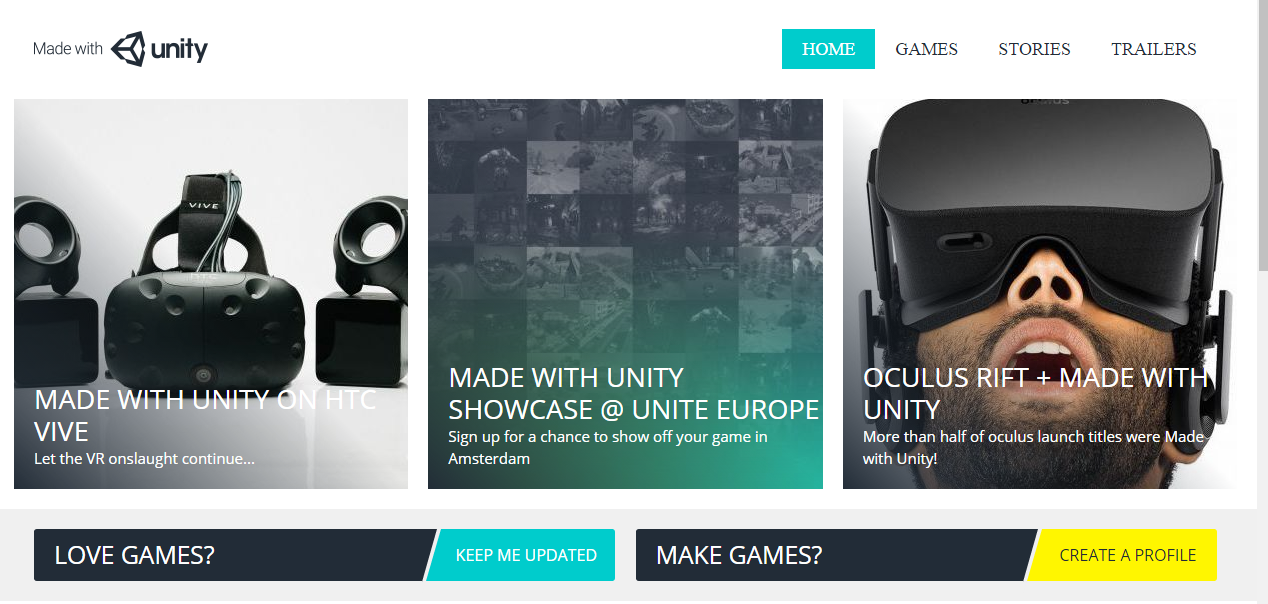
### 3.8.2 Recursos multimedia externos

* Nombre: recursos multimedia externos
* Descripción: recursos multimedia disponibles en medios externos al del entorno
* Tipo de valor: texto

#### 3.8.2.1 Recursos multimedia externos en Unity

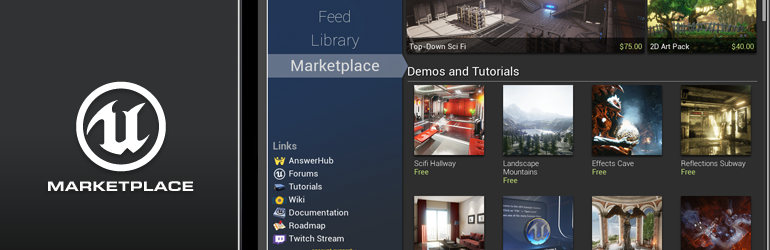
<https://www.assetstore.unity3d.com/en/#!/content/7065>

Unity posee un Marketplace de recursos multimedia desarrollados por la propia comunidad y profesionales del sector para poder comprar modelos 3D ya creados, escenarios, packs de texturas, etc…

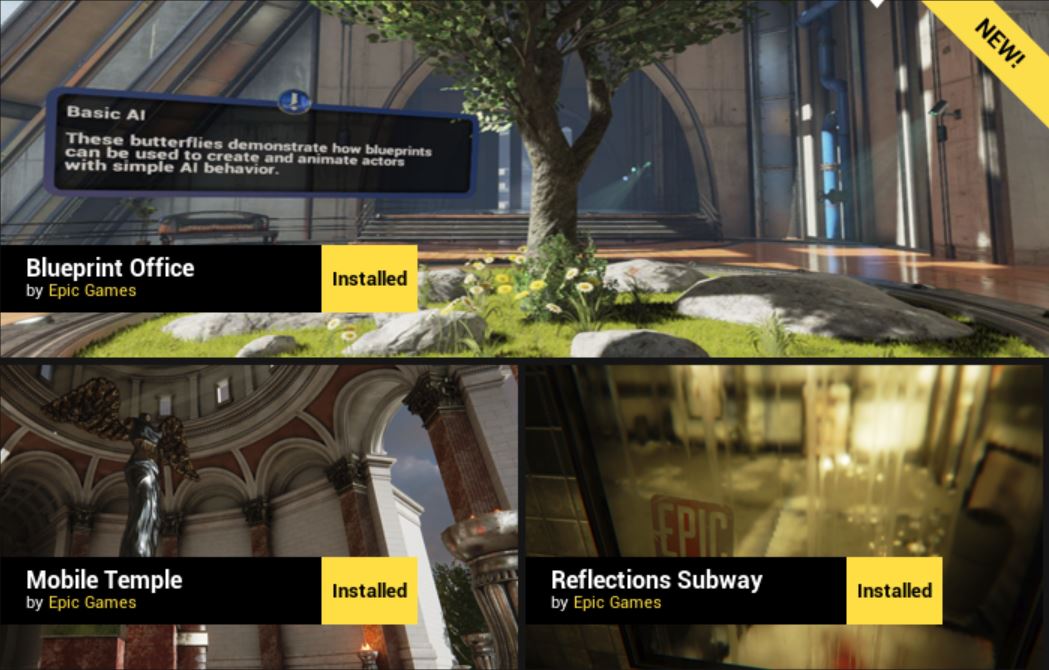


#### 3.8.2.2 Recursos multimedia externos en Unreal engine

Unreal Engine cuenta con su propio Marketplace donde la comunidad sube sus materiales para que cualquiera que esté interesado pueda comprar dicho material y añadirlo a su proyecto. En el Marketplace podemos encontrar material de todo tipo, desde objetos individuales hasta packs temáticos con una gran variedad de objetos agrupados. Los precios van desde artículos gratis hechos por la propia unreal o por la comunidad que los pone gratuitamente bajo licencias creative commons, hasta paquetes con muchos objetos a un precio moderado.



No obstante, no solo podemos encontrar recursos en el Marketplace que nos ofrece unreal, también podemos encontrar música de uso libre en sitios como youtube o personajes, fondos y demás piezas de arte en webs como opengameart.org.



## 3.9 HARDWARE

### 3.9.1 Requisitos mínimos para ejecutar el entorno

* Nombre: requisitos del entorno
* Descripción: requisitos necesarios en el pc para ejecutar el entorno
* Tipo de valor: texto

#### 3.9.1.1 Requisitos mínimos para ejecutar Unity

**OS**: Windows 7 SP1+, 8, 10; Mac OS X 10.8+.

Windows XP y Vista no son compatibles y las versiones de servidor de Windows & OS X no se han probado.

**GPU**: Capacidades de tarjeta de vídeo con DX9 (modelo de shader 2.0). Se presupone que cualquier tarjeta de video posterior al 2004 cumple con los requisitos suficientes.

**Requisitos adicionales para el desarrollo de plataformas:**

* iOS: Computadora Mac que ejecuta como mínimo la versión OS X 10.9.4 y Xcode 6.x.
* Android: Android SDK y Java Development Kit (JDK).
* Windows 8.1 Store Apps / Windows Phone 8.1: 64 bit Windows 8.1 Pro y Visual Studio 2013 Update 2+.
* WebGL: Mac OS X 10.8+ o Windows 7 SP1+ (solo editor de 64 bits)

#### 3.9.1.2 Requisitos mínimos para ejecutar Unreal Engine

Unreal Engine tiene algunos requisitos específicos de hardware y software para ejecutar el juego el editor para el desarrollo.

**Hardware Recomendado**

* Sistema Operativo: Windows 7/8 64-bit
* Procesador: Quad-core Intel or AMD, 2.5 GHz or faster
* Memoria: 8 GB RAM
* Tarjeta de video/DirectX Version: DirectX 11 compatible

**Requisitos Mínimos de Software**

A continuación, están los requisitos mínimos para ejecutar el motor o el editor.

**Ejecutar el Motor**

* Sistema Operativo: Windows 7/8 64-bit
* DirectX Runtime: DirectX End-User Runtimes (June 2010)

Los requisitos para los programadores que vayan a usar el motor para desarrollar están listados a continuación. Requisitos necesarios para 'Ejecutar el motor' (Instalados automáticamente)

* Visual Studio Versión Visual Studio 2015 Professional or Visual Studio 2015 Community
* Desarrollo iOS: iTunes 11 o superior

### 3.9.2 Requisitos mínimos para ejecutar el juego

* Nombre: requisitos juegos
* Descripción: requisitos necesarios en los dispositivos donde se va a ejecutar el juego desarrollado
* Tipo de valor: texto

#### 3.9.2.1 Requisitos mínimos para ejecutar los juegos en ambos

Para ambos motores los requisitos necesarios variaran en función de la complejidad del proyecto pero podemos exponer como requisitos generales los siguientes.

* Escritorio:
  + OS: Windows XP SP2+, Mac OS X 10.8+, Ubuntu 12.04+, SteamOS+
  + Tarjeta de vídeo: capacidades DX9 (shader modelo 2.0)
  + CPU: compatible con el conjunto de instrucciones SSE2.
  + Web Player (obsoleto): Requiere un navegador que soporte plugins, como IE, Safari y algunas versiones de Firefox
* iOS: requiere iOS 6.0 o versiones posteriores.
* Android: OS 2.3.1 o posterior; CPU ARMv7 (Cortex) con tecnología NEON o CPU Atom; OpenGL ES 2.0 o posterior.
* WebGL: Cualquier versión de escritorio reciente de Firefox, Chrome, Edge o Safari
* Windows Phone: 8.1 o posterior
* Windows Store Apps: 8.1 o posterior

## 3.10 SOPORTE PROPIO

### 3.10.1 Documentación

* Nombre: documentación
* Descripción: documentación propia facilitada por el desarrollador del entorno
* Tipo de valor: texto

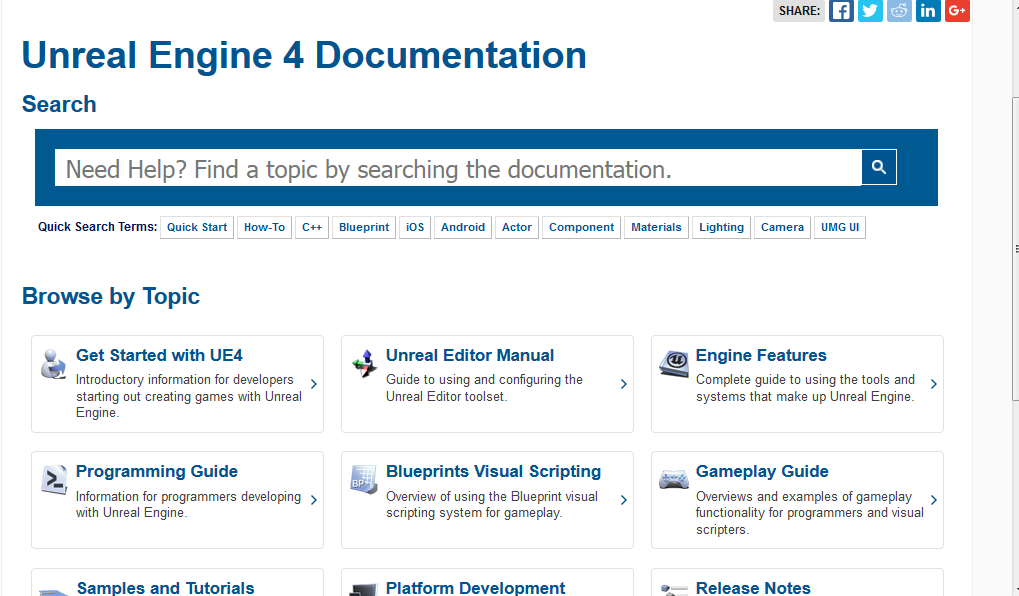
#### 3.10.1.1 Documentación de Unity

Unity cuenta con mucha documentación gratuita y de calidad en internet pero haremos inciso en la proporcionada por ellos mismos:

<http://docs.unity3d.com/Manual/index.html>

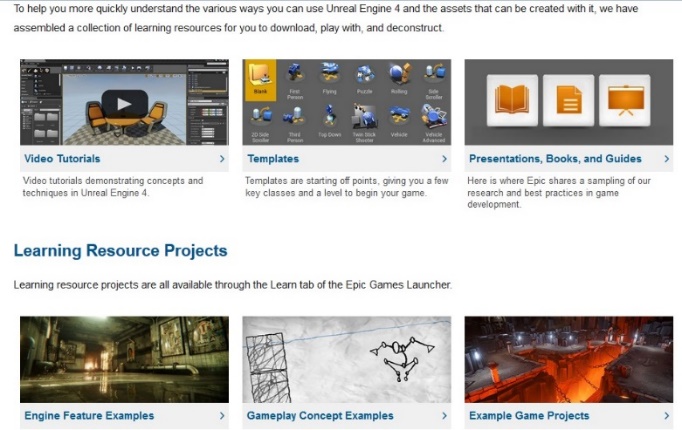
Podemos destacar de este manual su completitud y frecuencia de actualización pues con cada nueva versión del software este es revisado y actualizado.

#### 3.10.1.2 Documentación de Unreal Engine



La documentación que nos proporciona Epic Games sobre Unreal Engine es amplia y completa. Con solo leernos la documentación podemos aprender lo suficiente para utilizar el motor con eficacia. Dicha documentación es muy extensa ya que abarca en profundidad todas las características que posee el motor.

Además de toda la documentación detallada, podemos acceder a unas guías rápidas las cuales sirven para que si ya sabemos manejarnos un poco con versiones anteriores del motor o si por ejemplo venimos de haber trabajado con otro motor distinto (como Unity por ejemplo) podamos ponernos al día de forma rápida y sencilla sin complicarnos demasiado.

Por último, en la documentación podemos encontrar una serie de tutoriales guiados que nos indicarán como movernos por el motor y nos enseñarán a desenvolvernos por el de forma satisfactoria. Aparte de estos tutoriales también tenemos bastantes ejemplos funcionales de cómo podemos hacer proyectos para que los veamos y aprendamos de ellos.

### 

### 3.10.2 Asistencia técnica

* Nombre: asistencia
* Descripción: asistencia técnica facilitada por el desarrollador del entorno
* Tipo de valor: texto

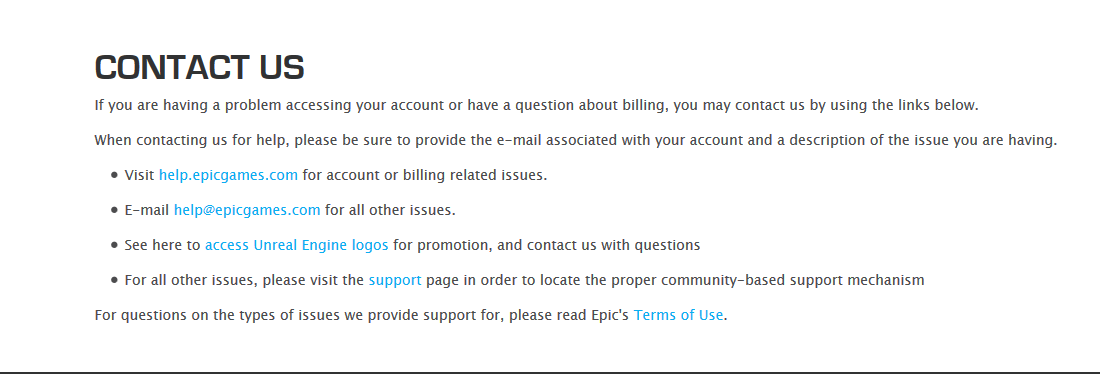
#### 3.10.2.1 Asistencia técnica de Unity

Unity presenta 2 versiones de su producto, la gratuita (Si el fruto de nuestro desarrollo genera más de X beneficios deberemos pagar royalties a Unity de igual manera ) el servicio técnico está más centrado a cuestiones de mal funcionamiento de la aplicación, temas de compras de licencias y la de pago los cuales se llevan a cabo en mensualidades y es esta versión Premium la cual nos ofrece un servicio de asistencia técnica profesional.

Este programa de asistencia técnica es vendido con la premisa o estategia comercial de que tu como usuario no tengas que preocuparte por los problemas derivados de un mal uso o desconocimiento del software, que ellos de forma rápida y eficaz te ayudaran con tu problema para que te centres en lo realmente importante que es la creación del videojuego.

Cabe destacar que prometen un servicio 24/7 con diferentes canales de contacto y diferentes paquetes de asistencia técnica en función de tus necesidades.

#### 3.10.2.2 Asistencia técnica de Unreal Engine



Además de todo lo mencionado en el tema de la documentación, Epic Games, cuenta con un equipo de soporte para Unreal Engine 4 que ofrece servicio a estudios que necesiten modificar el motor para algún proyecto específico o que necesiten algún tipo de licencia personalizada.

Para los usuarios normales, el soporte, se encarga de tareas relacionadas con la seguridad y el acceso a las cuentas de los usuarios, problemas de facturación, promoción y marketing, etc.

En caso de que tu consulta no esté entre lo que se menciona en la página de soporte te proporcionan un enlace hacia otra página para tratar tu solicitud de forma más especializada.

Por último, especifican que los tipos de incidencias a los que ellos proporcionan soporte se encuentran especificadas en los términos de uso.

## 4. Evaluación de los criterios por tecnología

## 4.1 Evaluación de los criterios para Unity

Debe incluir al menos una tabla con la siguiente estructura.

\*0 muy mal, 10 excelente

|  |  |
| --- | --- |
| **CRITERIOS** | **EVALUACIÓN** |
| **1.1:** **Diseño:** Interfaz | 8 |
| **1.2:** **Diseño:** Recursos predefinidos | 6 |
| **2.1:** **Características generales:** Multiplataforma | 9 |
| **2.2:** **Características generales:** Licencias | 9 |
| **3.1:** **Rendimiento:** Optimización | 7 |
| **4.1:** **Utilidad:** Compatibilidad con programas externos | 8 |
| **4.2:** **Utilidad:** Servicios | 7 |
| **5.1:** **Programación:** Lenguaje | 7 |
| **5.2:** **Programación:** API | 8 |
| **6.1:** **Gráficos:** Renderizado | 7 |
| **6.2:** **Gráficos:** Materiales y Shaders | 8 |
| **6.3:** **Gráficos**: Animación 2D y 3D | 8 |
| **7.1:** **Sonido:** Recursos propios dentro del entorno | 6 |
| **7.2:** **Sonido:** Programación del sonido dentro del juego | 6 |
| **8.1:** **Recursos:** Formación externa | 8 |
| **8.2:** **Recursos:** Recursos multimedia externos | 8 |
| **9.1:** **Hardware:** Requisitos mínimos para ejecutar el entorno | 9 |
| **9.2:** **Hardware:** Requisitos mínimos para ejecutar el juego | 8 |
| **10.1:** **Soporte propio:** Documentación | 9 |
| **10.2:** **Soporte propio:** Asistencia técnica | 6 |

## 4.2 Evaluación de los criterios para Unreal Engine

|  |  |
| --- | --- |
| **CRITERIOS** | **EVALUACIÓN** |
| **1.1:** **Diseño:** Interfaz | 8 |
| **1.2:** **Diseño:** Recursos predefinidos | 7 |
| **2.1:** **Características generales:** Multiplataforma | 7 |
| **2.2:** **Características generales:** Licencias | 8 |
| **3.1:** **Rendimiento:** Optimización | 7 |
| **4.1:** **Utilidad:** Compatibilidad con programas externos | 7 |
| **4.2:** **Utilidad:** Servicios | 6 |
| **5.1:** **Programación:** Lenguaje | 8 |
| **5.2:** **Programación:** API | 8 |
| **6.1:** **Gráficos:** Renderizado | 8 |
| **6.2:** **Gráficos:** Materiales y Shaders | 7 |
| **6.3:** **Gráficos**: Animación 2D y 3D | 7 |
| **7.1:** **Sonido:** Recursos propios dentro del entorno | 7 |
| **7.2:** **Sonido:** Programación del sonido dentro del juego | 6 |
| **8.1:** **Recursos:** Formación externa | 7 |
| **8.2:** **Recursos:** Recursos multimedia externos | 8 |
| **9.1:** **Hardware:** Requisitos mínimos para ejecutar el entorno | 8 |
| **9.2:** **Hardware:** Requisitos mínimos para ejecutar el juego | 8 |
| **10.1:** **Soporte propio:** Documentación | 9 |
| **10.2:** **Soporte propio:** Asistencia técnica | 7 |

## 5. Comparación de las tecnologías

|  |  |  |  |
| --- | --- | --- | --- |
| **CRITERIOS** | **UNITY** | **UNREAL** | **COMENTARIOS** |
| **1.1:** **Diseño:** Interfaz | 8 | 8 | Ambas son sencillas y completas. |
| **1.2:** **Diseño:** Recursos predefinidos | 6 | 7 | Unreal tiene mayor oferta de recursos predefinidos. |
| **2.1:** **Características generales:** Multiplataforma | 9 | 7 | Unity ofrece una mayor cantidad de plataformas a las que exportar. |
| **2.2:** **Características generales:** Licencias | 9 | 8 | La licencia gratuita de Unity ofrece todos los servicios y no cobra derechos hasta que haya un beneficio de 100.000€. Unreal cobra desde los 3.000€. |
| **3.1:** **Rendimiento:** Optimización | 7 | 7 | Ambos ofrecen recursos similares. |
| **4.1:** **Utilidad:** Compatibilidad con programas externos | 8 | 7 | Unity admite un mayor número de formatos para importar. |
| **4.2:** **Utilidad:** Servicios | 7 | 6 | Unity ofrece un mayor número de servicios. |
| **5.1:** **Programación:** Lenguaje | 7 | 8 | Unreal ofrece además un lenguaje visual como apoyo a la programación tradicional. |
| **5.2:** **Programación:** API | 8 | 8 | La de Unreal es más versátil y la de Unity es más fácil. |
| **6.1:** **Gráficos:** Renderizado | 7 | 8 | Unreal tiene más efectos de post-procesado. |
| **6.2:** **Gráficos:** Materiales y Shaders | 8 | 7 | En Unity es más fácil la creación de materiales. |
| **6.3:** **Gráficos**: Animación 2D y 3D | 8 | 7 | En Unity es más fácil la animación. |
| **7.1:** **Sonido:** Recursos propios dentro del entorno | 6 | 7 | Unreal permite la inclusión de fuentes stereo. |
| **7.2:** **Sonido:** Programación del sonido dentro del juego | 6 | 6 | Es bastante simple en ambas, y con pocas opciones. |
| **8.1:** **Recursos:** Formación externa | 8 | 7 | Unity tiene más cursos gratuitos. |
| **8.2:** **Recursos:** Recursos multimedia externos | 8 | 8 | Los recursos sirven para ambos motores. |
| **9.1:** **Hardware:** Requisitos mínimos para ejecutar el entorno | 9 | 8 | Unity tiene menos requisitos. |
| **9.2:** **Hardware:** Requisitos mínimos para ejecutar el juego | 8 | 8 | En ambos depende del proyecto. |
| **10.1:** **Soporte propio:** Documentación | 9 | 9 | Amplia documentación en ambos casos. |
| **10.2:** **Soporte propio:** Asistencia técnica | 6 | 7 | La de Unity requiere una licencia Profesional (de pago). |

## 6. Recomendaciones

### 6.1 Situación 1

#### 6.1.1 Descripción de la situación

*Un grupo de estudiantes recién graduados quiere crear una startup tecnológica. Como el sector del videojuego es un sector en alza han decidido montar un estudio de desarrollo de videojuegos. Su experiencia en el desarrollo de videojuegos es muy baja, dado que no se han interesado mucho por la industria durante la carrera. Por lo que tenemos un equipo con una escasa formación que quiere empezar a desarrollar un producto lo antes posible para ir generando ingresos para realizar proyectos más grandes en una vista a futuro.*

#### 6.1.2 Recomendación de tecnología a utilizar

La tecnología que proponemos para este caso es **Unity.**

Teniendo en cuenta que la mayor debilidad de este grupo es su inexperiencia en el sector, los criterios más importantes a tener en cuenta son los de la facilidad de uso del software. En este caso Unity ofrece mayores ventajas, ya que necesita menos experiencia y ofrece más facilidades, como se puede observar en la siguiente tabla:

|  |  |  |
| --- | --- | --- |
| **Criterios relevantes para la decisión** | **Ventajas Unity** | **Ventajas Unreal Engine** |
| Lenguaje | No necesita muchos conocimientos básicos ya que el lenguaje es más sencillo de aprender. | El lenguaje es más potente. |
| Licencias | Gratuita. Puedes ganar hasta 100.000€ sin necesidad de pagar nada. | Gratuita. Puedes ganar hasta 3.000€ sin necesidad de pagar nada. |
| Multiplataforma | Unity permite exportar a un gran número de plataformas, tanto las principales como otras más específicas, lo que permite llegar a un público mayor. | Unreal permite exportar a las principales plataformas. |
| Materiales | En Unity los materiales son muy fáciles de crear | En Unreal los materiales tienen un mejor acabado. |

### 6.2 Situación 2

#### 6.2.1 Descripción de la situación

*Un estudio independiente, debido al éxito de sus recientes proyectos ha crecido hasta un nivel moderado por lo cual el estudio ha decidido embarcarse en un proyecto de mayor envergadura que los realizados hasta ahora. El equipo del proyecto cuenta con cierta experiencia en el desarrollo de videojuegos ya que tienen varios títulos publicados por lo que pueden afrontar mayores retos sin que eso les suponga un coste demasiado alto. Dicho proyecto se requiere que disponga de un acabado más detallado debido al presupuesto disponible y al mayor número de ventas que se espera.*

#### 6.2.2 Recomendación de tecnología a utilizar

La tecnología que proponemos para este caso es **Unreal Engine.**

En este caso el grupo tiene experiencia, por tanto la facilidad no es un punto a tener en cuenta. El principal requisito es que el producto tenga la mayor calidad posible, por lo tanto recomendamos el motor que mayor calidad de producto terminado ofrece. A continuación una tabla con los criterios a tener en cuenta:

|  |  |  |
| --- | --- | --- |
| **Criterios relevantes para la decisión** | **Ventajas Unity** | **Ventajas Unreal Engine** |
| Lenguaje | No necesita muchos conocimientos básicos ya que el lenguaje es más sencillo de aprender. | El lenguaje es más potente, lo cual ofrece más posibilidades a la hora de desarrollar el juego |
| Renderizado | Consume menos recursos | Tiene un gran número de efectos de post-procesado. Esto permite conseguir un mejor resultado visual. |
| Animación | En Unity es más fácil | Unreal permite animaciones más realistas, lo que añade calidad al juego. |
| Materiales | En Unity los materiales son muy fáciles de crear. | En Unreal los materiales tienen un mejor acabado. |