Liliana Canalsova

Kevin Amores

Javier Blanco

Laia Pomar

TG3 - Motores de videojuegos: Unity y Unreal

DESARROLLO DE TECNOLOGÍAS EMERGENTES G1 Mañana

Índice

[1. Autores del trabajo, planificación y entrega 3](#_Toc450669540)

[1.1 Autores 3](#_Toc450669541)

[1.2 Planificación 3](#_Toc450669542)

[1.3 Entrega 3](#_Toc450669543)

[2. Requisitos del prototipo a implementar 4](#_Toc450669544)

[2.1 Requisitos funcionales 4](#_Toc450669545)

[Requisitos de juego: 4](#_Toc450669546)

[Requisitos del entorno: 4](#_Toc450669547)

[3. Criterios de comparación en la implementación 5](#_Toc450669548)

[3.1 Implementación 5](#_Toc450669549)

[3.1.1 Necesidades de ejecución del entorno 5](#_Toc450669550)

[3.1.2 Recursos ofrecidos por el entorno 6](#_Toc450669551)

[3.1.3 Creación de objetos 7](#_Toc450669552)

[3.1.4 Modificación de objetos 9](#_Toc450669553)

[3.1.5 Tiempo de desarrollo del proyecto 11](#_Toc450669554)

[3.2 Resultado 12](#_Toc450669555)

[3.2.1 Velocidad de ejecución del juego 12](#_Toc450669556)

[3.2.2 Renderizado 12](#_Toc450669557)

[4. Proyecto de implementación de un prototipo del videojuego utilizando Unity 13](#_Toc450669558)

[4.1 Documentación de diseño 13](#_Toc450669559)

[4.2 Documentación de construcción 14](#_Toc450669560)

[4.2.1 Creación de objetos y su programación para Unity 14](#_Toc450669561)

[4.2.2 Interfaz de usuario 15](#_Toc450669562)

[4.3 Documentación de pruebas 15](#_Toc450669563)

[4.4 Documentación de instalación 16](#_Toc450669564)

[4.5 Manual de usuario 17](#_Toc450669565)

[5. Proyecto de implementación de un prototipo del sistema utilizando Unreal 18](#_Toc450669566)

[5.1 Documentación de diseño 18](#_Toc450669567)

[5.2 Documentación de construcción 19](#_Toc450669568)

[5.2.1 Creación de objetos y su programación para Unreal 19](#_Toc450669569)

[5.2.2 Interfaz de usuario 21](#_Toc450669570)

[5.3 Documentación de pruebas 21](#_Toc450669571)

[5.4 Documentación de instalación 22](#_Toc450669572)

[5.5 Manual de usuario 22](#_Toc450669573)

[6. Comparación de las dos implementaciones 24](#_Toc450669574)

[6.1 Evaluación de los criterios en la implementación usando Unity 24](#_Toc450669575)

[6.2 Evaluación de los criterios en la implementación usando Unreal 24](#_Toc450669576)

[7. Comparación de la implementación de las tecnologías 25](#_Toc450669577)

[8. Conclusiones 26](#_Toc450669578)

# 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

Debido a la complejidad de los prototipos el desarrollo se ha realizado en equipo en reuniones presenciales. Dichas reuniones fueron los días 20, 27 de abril y 2 y 4 de mayo, en las que el equipo trabajó de manera conjunta en ambos prototipos y en el desarrollo del presente documento. Además, se hizo una reunión online el día 9 de mayo, en vista de la modificación de la fecha de entrega, con el fin de ultimar detalles.

## 1.3 Entrega

Enlace del trabajo: <https://github.com/laiapc/TG3>

# 2. Requisitos del prototipo a implementar

## 2.1 Requisitos funcionales

### Requisitos de juego:

| **REQ.** | **DESCRIPCIÓN** |
| --- | --- |
| RF01 | Los controles de la barra son flecha izquierda y derecha |
| RF02 | La pelota debe rebotar de manera natural automáticamente teniendo en cuenta el ángulo. |
| RF03 | Los bloques se rompen con la colisión de la pelota sumando puntos al marcador |
| RF04 | Cuando se pierde la bola en la pared inferior, se recupera una de la reserva. Para lanzarla hay que presionar la barra espaciadora |
| RF05 | La pelota rebota con las dos paredes laterales y el techo |

### Requisitos del entorno:

|  |  |
| --- | --- |
| RF06 | Se puede iniciar una nueva partida al perder todas las bolas, en la pantalla de “GAME OVER” |
| RF07 | Al empezar una partida hay que presionar la barra espaciadora |
| RF08 | Durante el juego se muestran las bolas restantes y los puntos acumulados |
| RF09 | Al romper todos los bloques se muestra “NIVEL FINALIZADO CON ÉXITO” y posteriormente se añadirían más niveles (no programados en este prototipo) |

# 3. Criterios de comparación en la implementación

## 3.1 Implementación

### 3.1.1 Necesidades de ejecución del entorno

#### 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)

#### Unreal

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**

**Ejecutar el Motor**

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

**Requisitos necesarios para 'Ejecutar el motor' (Instalados automáticamente)**

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

### 3.1.2 Recursos ofrecidos por el entorno

#### Unity

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 Unity 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.

#### Unreal

**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 ++.

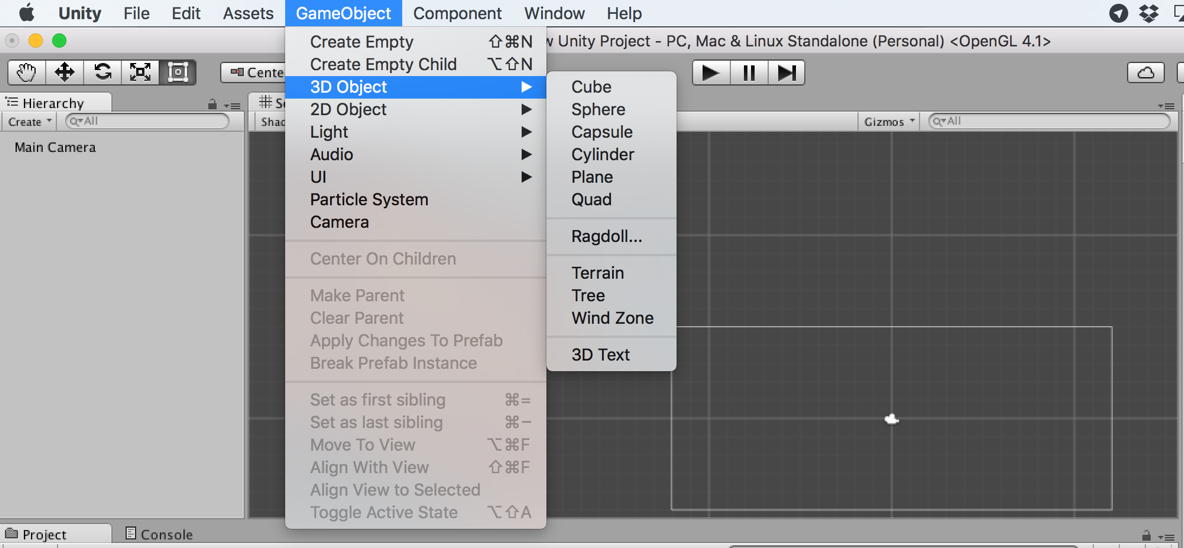
**Editor de comportamiento en árbol,** permite crear un script que programe la inteligencia artificial del videojuego mediante un sistema de nodos. Permite crear tareas, decorados, servicios etc.

**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.

### 3.1.3 Creación de objetos

Ambos programas tienen una serie de objetos predeterminados. Se puede elegir crear un objeto y pinchando sobre él se activarán una serie de propiedades modificables como el tamaño y la posición.

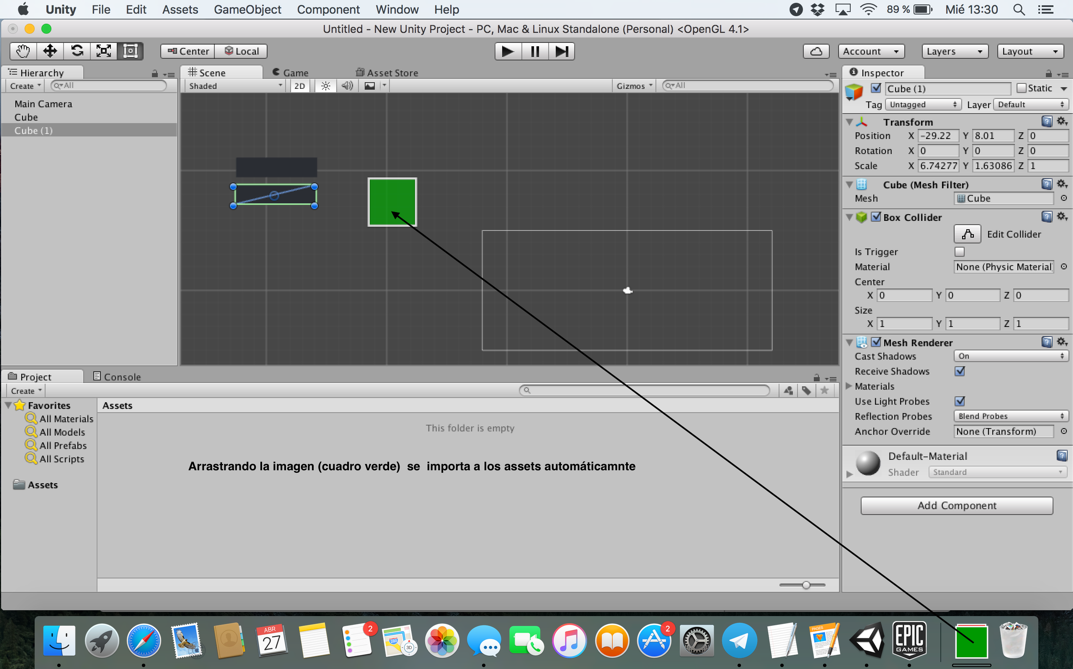
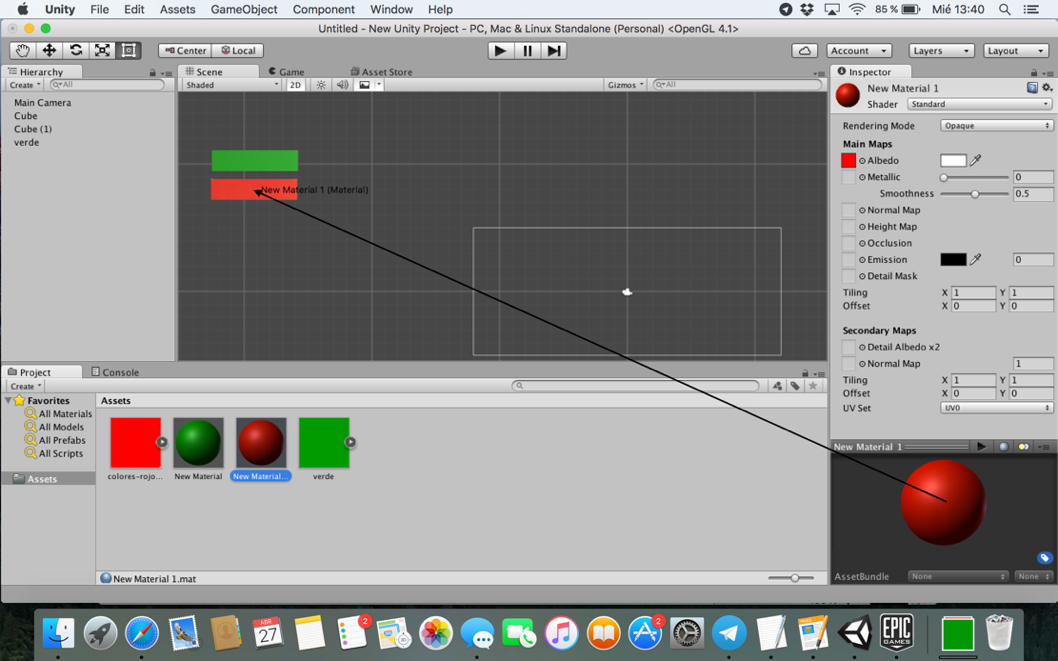
#### Unity



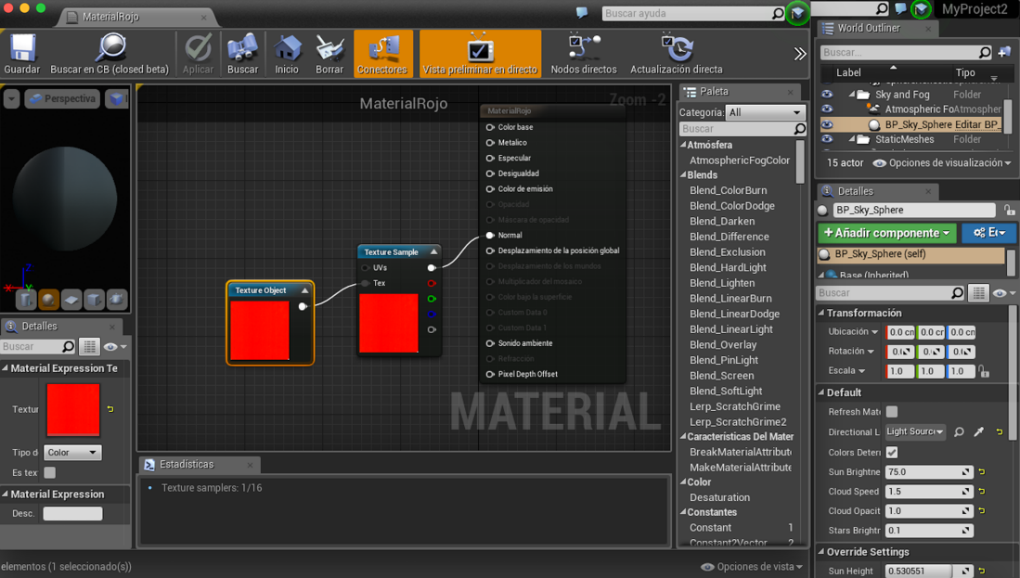
#### /Users/Chihiro/Desktop/Captura de pantalla 2016-04-27 a las 14.39.08.pngUnreal

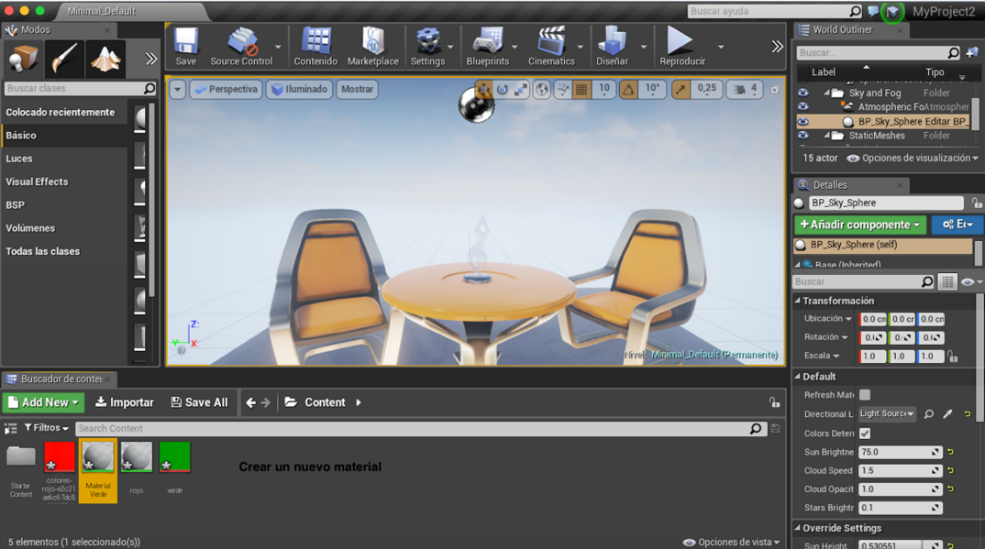
### 3.1.4 Modificación de objetos

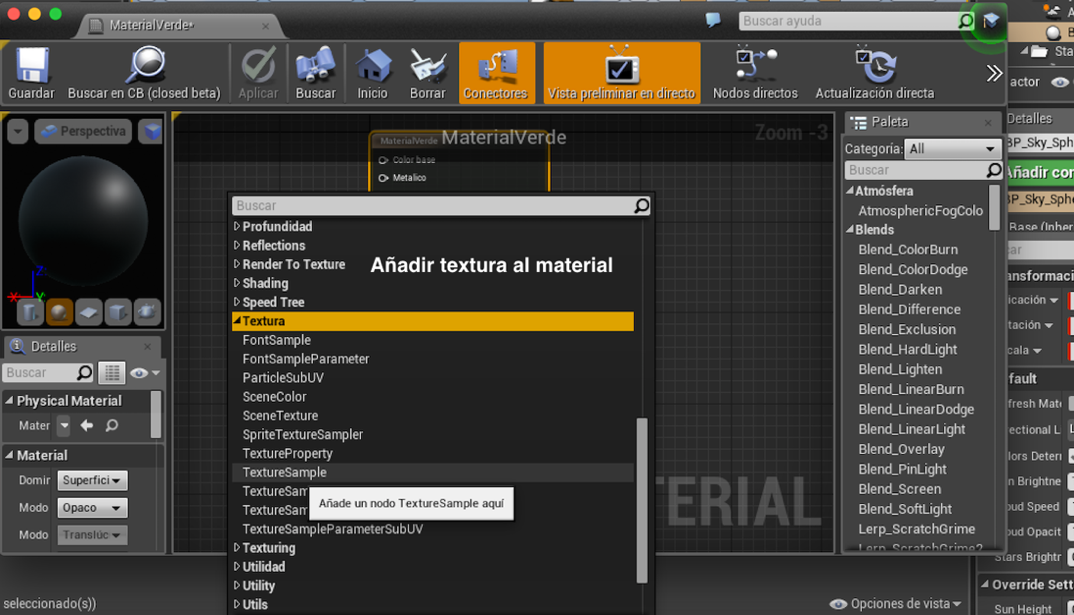
#### Unity

Para añadir una imagen (en este caso una imagen para establecer el color de un bloque) basta con descargar la imagen de internet y arrastrarla a los assets, una vez importado creado se crea un nuevo material y se arrastra el asset al material. Finalmente se arrastra el material al objeto.

Unreal:

Para añadir la misma imagen es necesario descargarla, importarla al proyecto como textura, crear un nuevo material y asignarle esa textura al material. Después desde el objeto cargamos ese material y obtenemos la apariencia deseada.





### 3.1.5 Tiempo de desarrollo del proyecto

Se ha trabajado de manera conjunta en los dos prototipos, por lo que se han dividido los tiempos trabajados en estudio de las tecnologías, implementación propiamente dicha y documentación del desarrollo en la memoria del trabajo.

Dado que no teníamos apenas conocimientos previos, excepto uno de los cuatro integrantes que sabía algo de Unity, para aprender lo básico necesario para el desarrollo que nos ocupa hemos invertido **cinco horas cada uno en aprender Unity y diez horas en aprender Unreal**.

En la **implementación propiamente dicha** hemos invertido **40 horas entre los cuatro** en el desarrollo de **Unity** y **60 horas** en el desarrollo de **Unreal**, lo que supone **diez horas aproximadamente cada integrante para Unity y quince horas cada integrante para Unreal**.

La diferencia de tiempo en el desarrollo en ambos entornos se debe a que Unity, como ya se comentó anteriormente en otros trabajos, está enfocado a un público amateur por lo que no requiere conocimientos avanzados de programación. En cambio, Unreal se enfoca a un público profesional que pretende desarrollar un videojuego con gran capacidad y detalles gráficos. Por estos motivos, el entorno de Unity es más intuitivo que el de Unreal a nivel de programación ya que permite realizar muchas tareas arrastrando y configurando las características a través de la interfaz gráfica.

Destaca en este sentido la capacidad de Unity para configurar los gráficos de las texturas simplemente arrastrándolas al entorno y asignándosela al objeto mientras que Unreal requiere crear una textura examinando los archivos y después asignarla a un material para poder ponérselo al objeto.

Por último, para el **desarrollo de la memoria y de la presentación se han invertido cinco horas** por parte de cada integrante.

## 3.2 Resultado

### 3.2.1 Velocidad de ejecución del juego

En cuanto a la velocidad de ejecución, las pruebas realizadas se llevaron a cabo en el mismo banco de pruebas con las que analizamos el renderizado final del proyecto, como dato adicional añadimos que el banco de pruebas ejecuta ambos proyectos con una GPU dedicada modelo GTX610M 2GB.

Las pruebas de velocidad de ejecución las basamos en el framerate obtenido en la ejecución del proyecto ya empaquetado y exportado a un fichero ejecutable desde Windows de 32bits, siempre claro está con la sincronización vertical desactivada ya que con ella activada no obtendríamos valores válidos para la comparación por encima de 60fps, lo cual es el objetivo a alcanzar.

Los resultados obtenidos fueron:

Una tasa variable entre 80 y 100 fps en el proyecto creado en Unity.

Una tasa estable entre 67 – 72 fps en Unreal.

Analizando los resultados podemos corroborar lo que en anteriores trabajos hemos afirmado, Unity proporciona aparte de un entorno de desarrollo más amigable, una mayor optimización para equipos de gama medio-baja y Unreal destaca por su mejor acabado y una exigencia de computación mayor que se ve reflejada en su menor tasa de fps con el mismo hardware.

### 3.2.2 Renderizado

Para esta comparación hemos realizado ambas pruebas en un entorno cerrado y controlado, nuestro banco de pruebas consta de un procesador i7 a 2,3 GHz, 8GB de RAM a 1133Mhz y disco mecánico SATA 3 con una velocidad de transmisión estándar para discos de 2,5” y 5400 rpm.

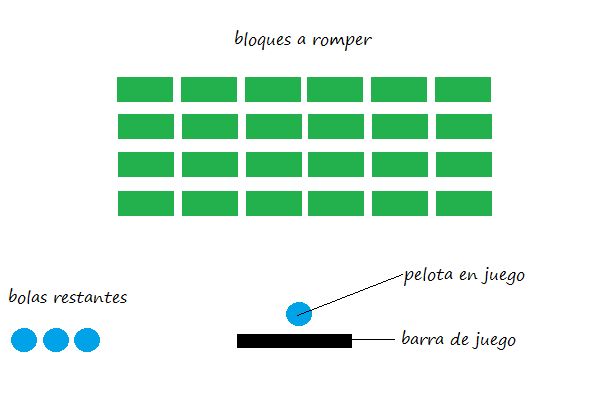
Para realizar las pruebas cronometramos el tiempo de empaquetado de los proyectos en el propio entorno natico de desarrollo obteniendo unos resultados de 1:37 Minutos en Unity y 59 Segundos en Unreal Engine.

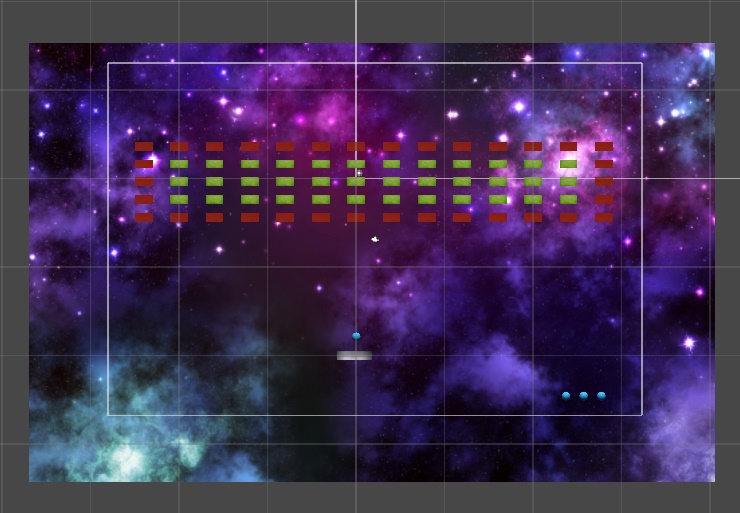
# 4. Proyecto de implementación de un prototipo del videojuego utilizando Unity

## 4.1 Documentación de diseño

Ambos proyectos se van a basar en el mismo diseño, un juego de tipo **arcade** llamado **“arkanoid”.** Este juego se basa en destruir bloques con una pelota que rebota en una plataforma controlada por el usuario que puede moverla de izquierda a derecha con las flechas del teclado. Existen 3 pelotas de máximo para superar el nivel, representando así las vidas del jugador.

El objetivo del juego es destruir todos los bloques mediante los choques de la pelota, cuando esta golpea un bloque este desaparece y suma puntos al jugador, al romper todos los bloques se gana la partida.

****



## 4.2 Documentación de construcción

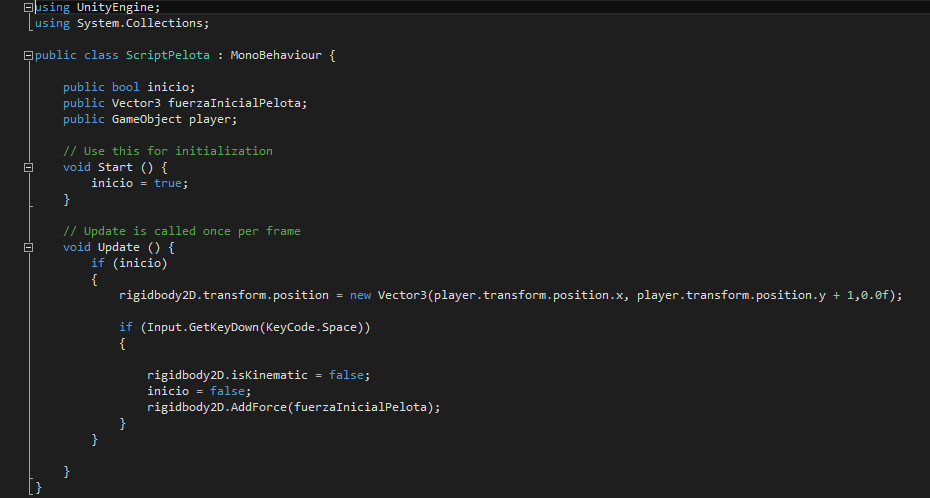
En líneas generales la construcción se puede dividir en creación de objetos y su programación e interfaz del usuario.

### 4.2.1 Creación de objetos y su programación para Unity

El juego consta de cuatro componentes, las paredes, los ladrillos, la pelota y la barra. Los ladrillos tienen una programación sencilla, ya que solo consiste en registrar una colisión con la pelota que implica la desaparición del bloque, para que no vuelva a chocar con él, y la suma de la puntuación.

En cuanto a la pelota, por un lado, se debe programar el lanzamiento al inicio del juego o al perder una vida, esto se realiza con la barra espaciador. Por otro lado, se debe programar la acción de rebote para todos los objetos que componen la escena, es decir, las paredes, los ladrillos y la barra de juego. Dicho rebote es igual para todos los objetos, solo implica conocer el ángulo de entrada para poder calcular el de salida.

Detallando la construcción de este objeto, en Unity a nivel gráfico se trata de una esfera con detalles físicos que se configuran mediante el editor. Además, se debe incluir un material físico configurado para que sea elástico el cual nos permitirá codificar su movimiento de rebote mediante C#. A continuación, se muestra una parte de dicho código:



Sobre la barra, gráficamente es un rectángulo tridimensional. En cuanto a su movimiento, dado que se mueve de izquierda a derecha con las flechas del teclado, la codificación es sencilla consistiendo en registrar cuando se aprieta una de las teclas y codificar el movimiento correspondiente.



### 4.2.2 Interfaz de usuario

La interfaz de usuario consiste en mostrar los puntos obtenidos por el usuario y las vidas que restan durante la ejecución del juego. Además, al iniciar el juego se le indica al jugador que debe presionar la barra espaciadora para lanzar la bola, al perder se muestra el mensaje “GAME OVER” y al ganar “NIVEL SUPERADO”.

## 4.3 Documentación de pruebas

Para la realización de la construcción hemos seguido un modelo iterativo e incremental de desarrollo. Por lo que hemos ido probando el proyecto a medida que íbamos acabando de construir los siguientes elementos.

Como enfoque hemos realizado tanto pruebas de caja negra, como pruebas de caja blanca según el elemento que necesitábamos probar y según los resultados de pruebas anteriores.

Las pruebas realizadas han sido:

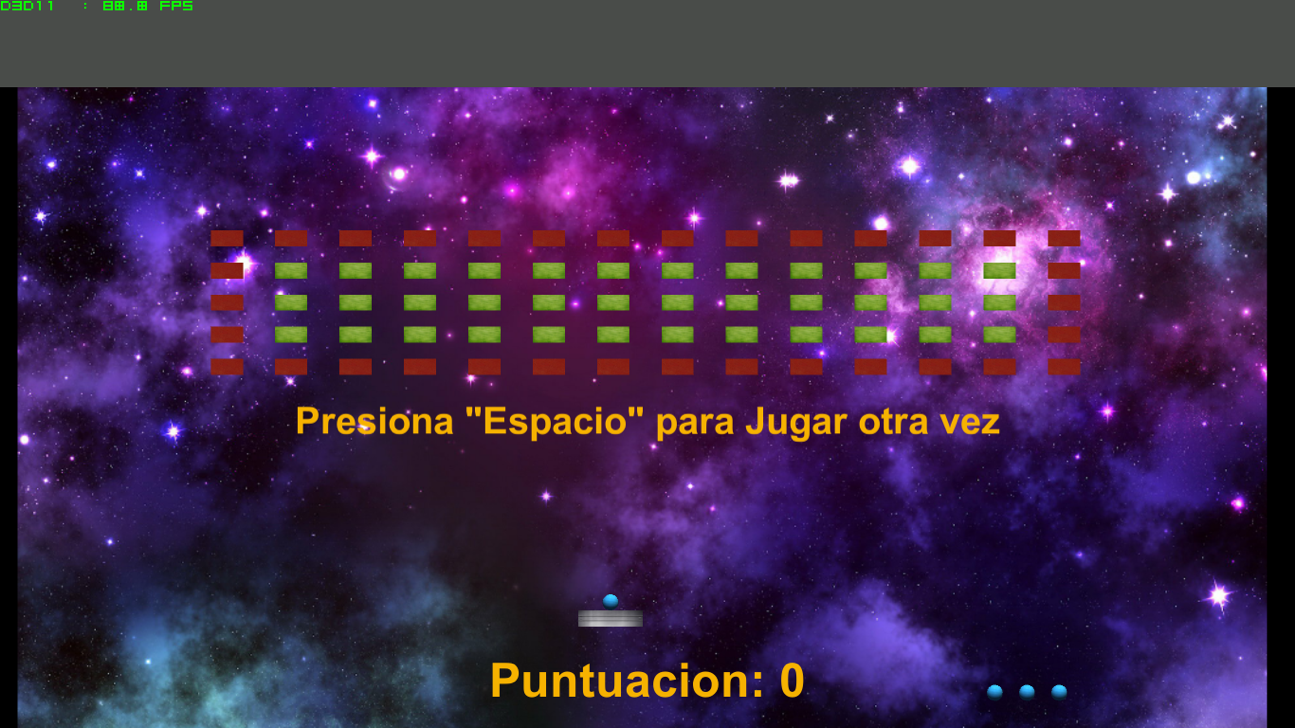
* **Pruebas de compatibilidad**: hemos probado el proyecto tanto en Windows 10 como Windows 7 y en Mac OS 10.11.3, en las cuales hemos comprobado su correcto funcionamiento en todas las plataformas.
* **Pruebas de integración**: dado que cada componente es independiente de los otros hemos probado su funcionamiento en conjunto ya que es necesaria una correcta integración para realizar algunas funciones como puede ser la victoria o derrota en el juego, además de probar la integración con distintos dispositivos de control como pueden ser el teclado y un mando de Xbox.
* **Pruebas de regresión**: como hemos seguid un modelo de desarrollo iterativo e incremental hemos ido probando cada elemento según se acababa de desarrollar, no obstante, también comprobamos que el resto de elementos seguían funcionando bien tras incluir los nuevos objetos.
* **Pruebas unitarias**: dado el modelo de desarrollo de software que hemos usado, es inherente las pruebas modulares ya que tras acabar cada módulo desarrollado hemos de comprobar si está bien desarrollado y funciona correctamente de forma independiente.
* **Pruebas de humo**: una vez terminado el desarrollo del proyecto hemos realizado varias pruebas de humo en las que nosotros el equipo de desarrollo hemos actuado como si fuéramos el usuario final y hemos probado el funcionamiento del proyecto en su totalidad, corrigiendo los pequeños bugs que surgieron y optimizándolo para un correcto desempeño en el usuario final.
* **Pruebas de usabilidad**: dado que en el desarrollo de un videojuego es muy importante la interacción con el usuario, hemos realizado pruebas de usabilidad para que la jugabilidad sea óptima y el usuario no tenga problemas a la hora de transmitir el movimiento al mundo a través del control.

Todas las pruebas inicialmente se han hecho siguiendo un modelo de caja negra ya que es el que mejor nos permite ver la funcionalidad desde el punto de vista del usuario final y nos permite observar de una forma más objetiva cómo funciona el conjunto. No obstante, hay ciertos elementos como la pelota o el jugador que debido a su componente de movimiento hemos tenido que realizar pruebas con un enfoque de caja blanca ya que necesitábamos que fueran los más precisos posibles ya que son las partes más delicadas y un funcionamiento incorrecto puede dar lugar a grandes anomalías altamente visibles de cara a la ejecución del juego.

## 4.4 Documentación de instalación

El juego es un ejecutable, de forma que no necesita instalación. Al hacer doble clic se ejecuta directamente.

## 4.5 Manual de usuario

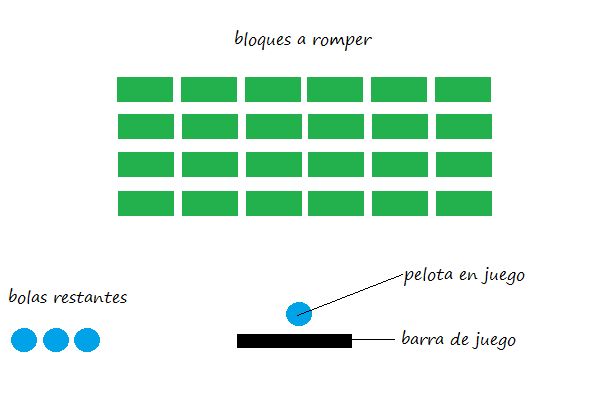
1. Hacer doble clic sobre la aplicación Arkanoid.exe
2. Pulsar la barra espaciadora para iniciar el juego
3. La pelota rebotará en los bloques superiores eliminándolos y aumentando la puntuación. Para conseguir que vuelva a subir hay que atraparla con la barra inferior (gris). Desplazar la barra con las flechas del teclado hacia la izquierda o la derecha, colocándola siempre debajo de donde se cree que va a caer la pelota.
4. Si la pelota cae fuera de la barra se perderá una vida. Pulsar la barra espaciadora para volver a lanzar.
5. Las vidas restantes se muestran abajo a la derecha (3 pelotas) y la puntuación se muestra abajo en el centro.
6. Para cerrar pulsar Alt+f4.

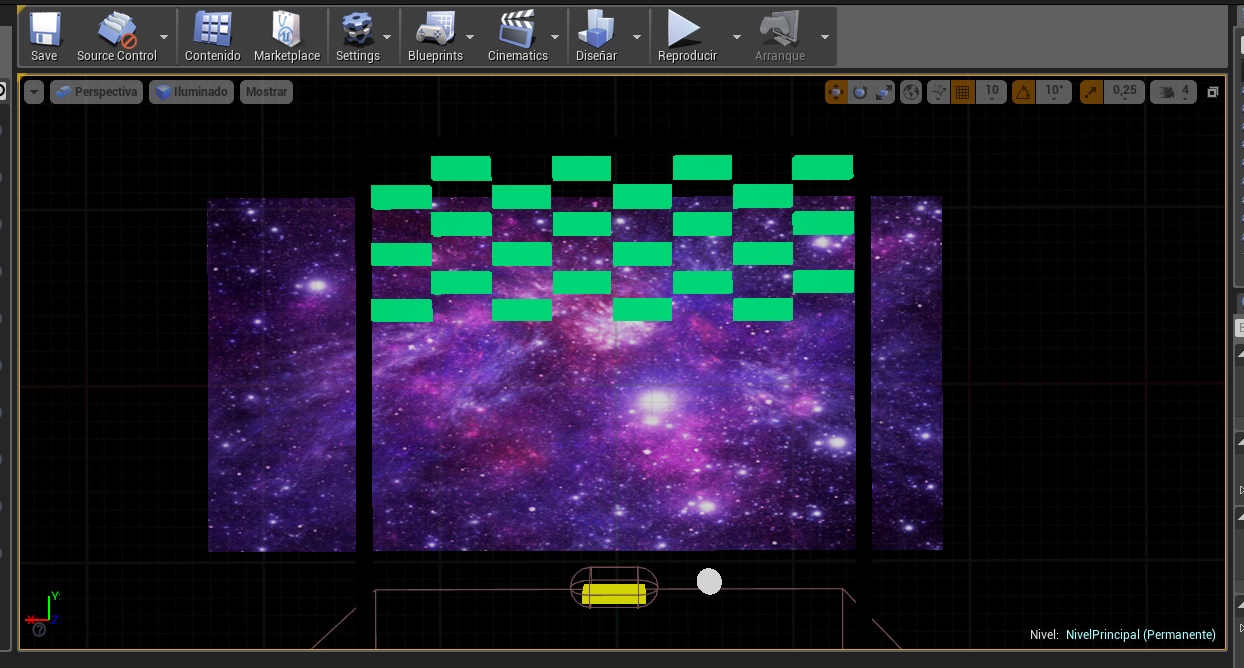
# 5. Proyecto de implementación de un prototipo del sistema utilizando Unreal

## 5.1 Documentación de diseño

Ambos proyectos se van a basar en el mismo diseño, un juego de tipo **arcade** llamado **“arkanoid”.** Este juego se basa en destruir bloques con una pelota que rebota en una plataforma controlada por el usuario que puede moverla de izquierda a derecha con las flechas del teclado. Existen 3 pelotas de máximo para superar el nivel, representando así las vidas del jugador.

El objetivo del juego es destruir todos los bloques mediante los choques de la pelota, cuando esta golpea un bloque este desaparece y suma puntos al jugador, al romper todos los bloques se gana la partida.

****



## 5.2 Documentación de construcción

En líneas generales la construcción se puede dividir en creación de objetos y su programación e interfaz del usuario.

### 5.2.1 Creación de objetos y su programación para Unreal

Los componentes creados en Unreal son los mismos que los de Unity siendo su comportamiento exactamente el mismo, solo que cambiando la forma de implementación.

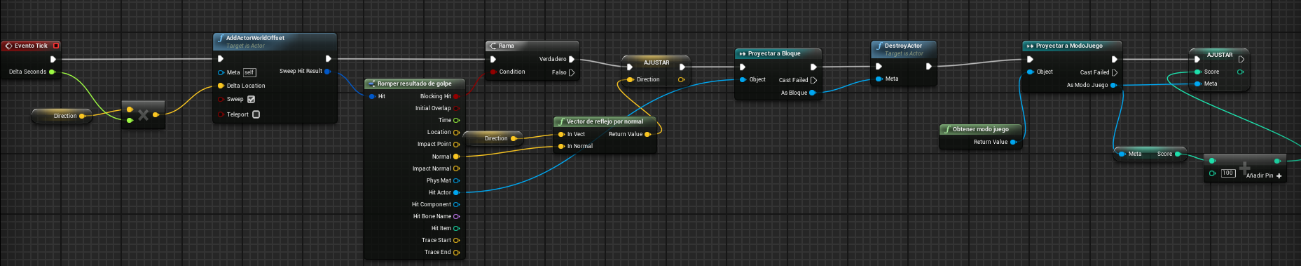
Dado que las funcionalidades son las mismas la única diferencia es la implementación, seguimos teniendo los ladrillos que se destruyen al colisionar la bola, el límite inferior que hace que perdamos una vida, las paredes de los lados y el techo, etc.

No obstante, en Unity nos veíamos obligados a programarlo todo en C# ya que es la única forma de establecer el comportamiento de los objetos dentro del juego; pero en Unreal tenemos dos opciones a la hora de implementar estas funciones.

La primera opción es la de utilizar código C++ para programar de forma detallada el comportamiento de los objetos del juego, la segunda es la de utilizar la interfaz de programación gráfica de Blueprints, la cual nos permite establecer el comportamiento de los objetos a través de funciones predefinidas del motor.

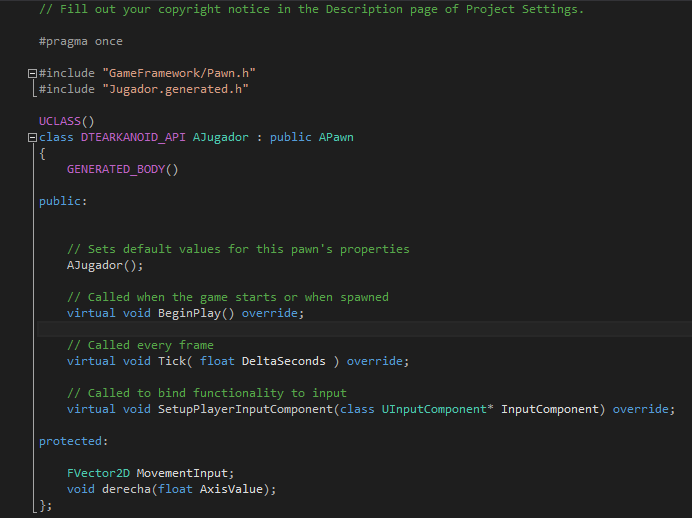
Nosotros hemos optado para programar los objetos con blueprints, ya que con las funciones que vienen predefinidas nos vale para conseguir el comportamiento que queríamos ahorrándonos tiempo de esta forma y reservando C++ para comportamientos más complejos. No obstante, hemos decidido programar la barra del jugador en lenguaje C++ para hacer una comparación más rigurosa como establecemos más adelante.

Si comparamos la construcción de los componentes, como hemos dicho anteriormente las funciones actúan de la misma forma con la diferencia de que en la pelota, en Unity, el comportamiento de rebote estaba cargado al material físico que nos daba Unity y en Unreal hemos tenido que programarlo nosotros mismos.

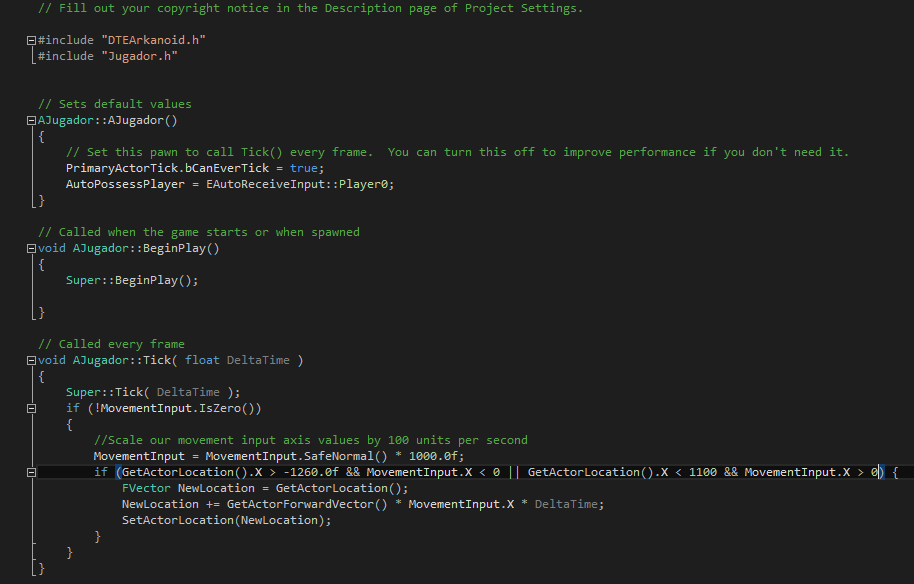


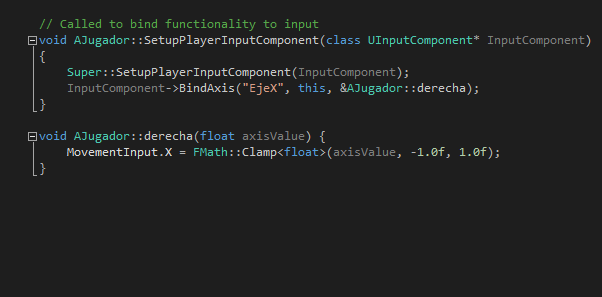
Sobre la barra, al igual que en Unity, gráficamente es un rectángulo tridimensional. El movimiento se programa igualmente de forma sencilla. En el caso de Unreal no es necesario programar este movimiento con C++, ya que se puede programar con blueprints o funciones predeterminadas. Sin embargo, para poder establecer una comparación más rigurosa se ha programado de la forma tradicional, como podemos ver en el siguiente código.

Jugador.h



Jugador.cpp





### 5.2.2 Interfaz de usuario

La interfaz de usuario consiste en mostrar los puntos obtenidos por el usuario y las vidas que restan durante la ejecución del juego. Además, al iniciar el juego se le indica al jugador que debe presionar la barra espaciadora para lanzar la bola, al perder se muestra el mensaje “GAME OVER” que se da cuando al jugador no le quedan vidas y al ganar “NIVEL SUPERADO” que se da cuando ha obtenido el score completo, estos eventos están programados a través de blueprints.

## 5.3 Documentación de pruebas

Para la realización de la construcción hemos seguido un modelo iterativo e incremental de desarrollo. Por lo que hemos ido probando el proyecto a medida que íbamos acabando de construir los siguientes elementos.

Como enfoque hemos realizado tanto pruebas de caja negra, como pruebas de caja blanca según el elemento que necesitábamos probar y según los resultados de pruebas anteriores.

Las pruebas realizadas han sido:

* **Pruebas de compatibilidad**: hemos probado el proyecto tanto en Windows 10 como Windows 7 y en Mac OS 10.11.3, en las cuales hemos comprobado su correcto funcionamiento en todas las plataformas.
* **Pruebas de integración**: dado que cada componente es independiente de los otros hemos probado su funcionamiento en conjunto ya que es necesaria una correcta integración para realizar algunas funciones como puede ser la victoria o derrota en el juego, además de probar la integración con distintos dispositivos de control como pueden ser el teclado y un mando de Xbox.
* **Pruebas de regresión**: como hemos seguid un modelo de desarrollo iterativo e incremental hemos ido probando cada elemento según se acababa de desarrollar, no obstante también comprobamos que el resto de elementos seguían funcionando bien tras incluir los nuevos objetos.
* **Pruebas unitarias**: dado el modelo de desarrollo de software que hemos usado, es inherente las pruebas modulares ya que tras acabar cada módulo desarrollado hemos de comprobar si está bien desarrollado y funciona correctamente de forma independiente.
* **Pruebas de humo**: una vez terminado el desarrollo del proyecto hemos realizado varias pruebas de humo en las que nosotros el equipo de desarrollo hemos actuado como si fuéramos el usuario final y hemos probado el funcionamiento del proyecto en su totalidad, corrigiendo los pequeños bugs que surgieron y optimizándolo para un correcto desempeño en el usuario final.
* **Pruebas de usabilidad**: dado que en el desarrollo de un videojuego es muy importante la interacción con el usuario, hemos realizado pruebas de usabilidad para que la jugabilidad sea óptima y el usuario no tenga problemas a la hora de transmitir el movimiento al mundo a través del control.

Todas las pruebas inicialmente se han hecho siguiendo un modelo de caja negra ya que es el que mejor nos permite ver la funcionalidad desde el punto de vista del usuario final y nos permite observar de una forma más objetiva cómo funciona el conjunto. No obstante, hay ciertos elementos como la pelota o el jugador que debido a su componente de movimiento hemos tenido que realizar pruebas con un enfoque de caja blanca ya que necesitábamos que fueran los más precisos posibles ya que son las partes más delicadas y un funcionamiento incorrecto puede dar lugar a grandes anomalías altamente visibles de cara a la ejecución del juego.

## 5.4 Documentación de instalación

Igual que con Unity no es necesario instalar el juego, ya que directamente tendremos un ejecutable.

## 5.5 Manual de usuario

1. Hacer doble clic sobre la aplicación ArkanoidUnreal.exe
2. El juego se iniciará automáticamente.



1. La pelota rebotará en los bloques superiores eliminándolos y aumentando la puntuación. Para conseguir que vuelva a subir hay que atraparla con la barra inferior (amarilla). Desplazar la barra con las flechas del teclado hacia la izquierda o la derecha, colocándola siempre debajo de donde se cree que va a caer la pelota.
2. Si la pelota cae fuera de la barra se perderá una vida. Pulsar la barra espaciadora para volver a lanzar.
3. Las vidas y la puntuación se muestran abajo a la izquierda.
4. Para cerrar pulsar Alt+f4.

# 6. Comparación de las dos implementaciones

## 6.1 Evaluación de los criterios en la implementación usando Unity

| **CRITERIO** | **EVALUACIÓN** |
| --- | --- |
| **3.1 Implementación** | |
| 3.1.1 Necesidades de ejecución del entorno | 8 |
| 3.1.2 Recursos ofrecidos por el entorno | 9 |
| 3.1.3 Creación de objetos | 9 |
| 3.1.4 Modificación de objetos | 8 |
| 3.1.5 Tiempo de desarrollo del proyecto | 8 |
| **3.2 Resultado** | |
| 3.2.1 Velocidad de ejecución del juego | 9 |
| 3.2.2 Renderizado | 8 |

## 6.2 Evaluación de los criterios en la implementación usando Unreal

| **CRITERIO** | **EVALUACIÓN** |
| --- | --- |
| **3.1 Implementación** | |
| 3.1.1 Necesidades de ejecución del entorno | 7 |
| 3.1.2 Recursos ofrecidos por el entorno | 9 |
| 3.1.3 Creación de objetos | 9 |
| 3.1.4 Modificación de objetos | 6 |
| 3.1.5 Tiempo de desarrollo del proyecto | 6 |
| **3.2 Resultado** | |
| 3.2.1 Velocidad de ejecución del juego | 9 |
| 3.2.2 Renderizado | 7 |

| **CRITERIOS** | **UNITY** | **UNREAL** | **COMENTARIOS** |
| --- | --- | --- | --- |
| **3.1 Implementación** | | | |
| 3.1.1 Necesidades de ejecución del entorno | 8 | 7 | Unity tiene menos requisitos |
| 3.1.2 Recursos ofrecidos por el entorno | 9 | 9 | Ambos ofrecen gran cantidad de recursos |
| 3.1.3 Creación de objetos | 9 | 9 | Se crean objetos predefinidos fácilmente en ambos |
| 3.1.4 Modificación de objetos | 8 | 6 | En Unreal es mucho más complicado |
| 3.1.5 Tiempo de desarrollo del proyecto | 8 | 6 | Unity al ser más fácil se desarrolla más rápido |
| **3.2 Resultado** | | | |
| 3.2.1 Velocidad de ejecución del juego | 9 | 9 | Los resultados son similares |
| 3.2.2 Renderizado | 8 | 7 | En este caso en concreto en Unity es más rápido |

# 7. Comparación de la implementación de las tecnologías

# 8. Conclusiones

En conclusión, hemos podido corroborar las deducciones teóricas extraídas de los anteriores trabajos, Unreal resulta mucho más complejo de trabajar que Unity y requiere de conocimientos avanzados de programación y diseño.

Durante el desarrollo de ambos prototipos hemos podido comprobar que Unity tiene una interfaz de usuario fácilmente comprensible con una curva de aprendizaje mucho más asequible para nuevos usuarios que Unreal.

Adentrándonos en el entorno de creación de objetos, Unity provee de herramientas gráficas que facilitan las tareas del programador a la hora de crear objetos y dotarlos de grafismos y mecánicas, mientras que Unreal resulta complejo en estos términos.

En cuanto a la programación, a pesar de que Unity requiere programar en C++, resulta más fácil para un programador que comprender Unreal, ya que el sistema de blueprints aun siendo gráfico es complejo de manejar.