Universidad de Valladolid

Escuela de Ingeniería Informática

Master en Ingeniería Informática

"Desarrollo de un videojuego FPS con Unity 3D"

Alumno: Borja Cerezo Redondo

Tutor: Benjamín Sahelices Fernández

Resumen:

En esta memoria, se presenta el proceso de desarrollo, diseño e implementación de un videojuego online multijugador mediante el manejo de las herramientas Unity 3D como motor de desarrollo, Cinema 4D como herramienta de modelado 3D y Adobe Photoshop para la edición y creación de texturas.

El objetivo principal de este proyecto consiste en, por un lado aprender a desarrollar este tipo de aplicaciones y por otro, experimentar todo el proceso y trabajo que conlleva realizar un videojuego desde cero.

Abstract:

In this report, it's presented the development process, design and implementation of an online multiplayer game. This development process will be conducted using Unity 3D such as engine development, Cinema 4D such as 3D modeling tool and Adobe Photoshop for editing and creating textures.

The main objective of this project is, first learn to develop these applications and second, to experience the whole process and work involved in making a game from scratch.

Agradecimientos

A mis padres y hermanos por todo el apoyo y paciencia que han tenido conmigo durante todo este tiempo.

A Andrés Escobar por su ayuda prestada con la revisión de la documentación y las pruebas de funcionamiento del juego.

Gracias por todo vuestro apoyo y ánimos que han hecho posible este proyecto fin de Master.

Índice:

Ilustraciones:

Tablas:

1 Introducción

A día de hoy podemos considerar a los videojuegos como una parte esencial de nuestra sociedad, no solo desde el punto de vista de ocio, sino también desde los avances tecnológicos.

Los programas de carácter lúdico nacieron tras el fin de la segunda guerra mundial [1] con la construcción de los primeros supercomputadores y a partir de los años 60 comenzaron a circular las primeras consolas y videojuegos comerciales. Desde entonces el mundo de los videojuegos no ha dejado de crecer.

Imagen 1. 1962 primer juego comercial: SpaceWar para DEC PDP-1

Desde sus comienzos hasta la actualidad, el avance hardware y software de los computadores ha evolucionado condicionado en gran parte con los avances de videojuegos dando lugar a nuevas ramas de investigación y desarrollo. Algoritmos de simulación de fluidos, inteligencia artificial, sistemas de procesamiento gráfico… todos estos avances nacieron de diferentes desarrolladores y gracias a los videojuegos lograron un rápido desarrollo y optimización, permitiendo extender sus aplicaciones a otros campos como las industrias del cine y de la música.

Imagen 2. Tecnología Phys-X de Nvidia para simulación de fluidos

Imagen 3. Evolución gráfica de los videojuegos

Por otro lado, los videojuegos se han convertido en una forma de ocio preferida por gran parte de la población y además, no solo es una herramienta de entretenimiento, también se usa como recurso educativo, formativo y gracias a la aparición de la realidad aumentada como método de rehabilitación.

1.1 Motivación y Objetivos

1.1.1 Motivación:

Como ya he mencionado, los videojuegos son una parte esencial en nuestra sociedad, llegando a ocupar uno de los primeros puestos en los grandes negocios mediáticos. Un juego de éxito puede generar fácilmente en sus primeros días de venta ingresos iguales o superiores a un gran éxito de cartelera.

Sin embargo, el desarrollo de un juego no es tarea fácil, se requieren conocimientos de diferentes disciplinas y herramientas. No existe manuales ni tutoriales de cómo abordar el desarrollo de este con una única metodología concreta. Normalmente son necesarias decenas de personas trabajando en equipo durante largos periodos de tiempo que superan fácilmente el año.

A través de este proyecto, se pretende abordar el desarrollo de un juego tipo FPS (First Person Shooter) online. Se ha elegido este género en concreto ya que a diferencia de otros tipos de videojuegos, éste nos ofrece por una parte una gran flexibilidad en cuanto a su contenido, permitiéndonos desarrollar versiones iniciales jugables sin necesidad de una trama o historia y por otro lado posee un nivel de complejidad en el que no es fácil su desarrollo pero que gracias a los motores gráficos actuales, puede realizarse bajo el desarrollo de una única persona.

1.1.2 Objetivos:

Por tanto, como ya se ha mencionado anteriormente, el objetivo de este proyecto fin de Master es la elaboración de un juego tipo FPS mediante el uso del motor gráfico "Unity 3D".

Para el desarrollo de dicho juego, es primordial el aprendizaje y realización de las siguientes tareas:

- Recogida de información y aprendizaje sobre el uso de las herramientas que se van a usar (Unity 3D, Cinema 4D y Adobe Photoshop).
- Modelado de diferentes modelos 3D (como los personajes jugables y elementos decorativos del mapa) a través de la herramienta "Cinema 4D".

- Modelado y creación de mapas a través de "Unity 3D", con la complementación de "Cinema 4D".
- Desarrollo de los diferentes Scripts en C# para establecer la mecánica de funcionamiento del juego, junto con las diferentes interacciones, eventos, colisiones y resto de reglas que permitan el funcionamiento correcto del juego.
- Sincronización de los movimientos y acciones de los diferentes jugadores online entre cliente y servidor.
- Creación de la interfaz y menús de la aplicación.

1.2 Estructura de este documento

Se ha organizado los diferentes capítulos que componen esta memoria de la siguiente manera:

- **Introducción:** Contexto de la evolución de los videojuegos desde su primera aparición hasta la actualidad, motivación y objetivos de la realización de este proyecto.
- **Estado del arte:** Definición de un motor de videojuegos, evolución, tipos y comparativa de los motores más representativos.
- **Gestión del proyecto:** Descripción de la metodología de Ingeniería del software aplicada para el desarrollo del proyecto y plan de riesgos a aplicar.
- **Plan del proyecto:** Descripción del alcance del proyecto, su coste, la planificación a seguir, análisis de requisitos, casos de uso y modelo de dominio.
- **Plan de iteraciones:** Explicación del proceso de desarrollo del proyecto en cada iteración aplicada. Cuáles son las funcionalidades que se han ido aplicando, corrección de fallos detectados y explicación de cómo se ha implementado.
- **Conclusiones:** Se establece una serie de conclusiones y líneas de trabajo futuras junto con la mención de las dificultades encontradas en el desarrollo de este proyecto
- **Contenido del CD-ROM:** Archivos de proyecto que encontraremos en el disco.
- **Bibliografía:** Referencias bibliográficas que han sido utilizadas.
- **Anexo:** Manuales básicos de las herramientas utilizadas donde se explica su uso y los controles de los que dispone.

2 Estado del arte

Este capítulo explica que es un "Game engine", (también denominado "Motor gráfico" o "Motor de Juegos") junto con una breve comparación de los diferentes tipos de motores gráficos más conocidos. Además se explica también el manejo básico de "Unity 3D", junto del resto de herramientas utilizadas en el desarrollo del videojuego.

2.1 ¿Qué es un Game Engine?

Es un "framework" [2], es decir, un conjunto estandarizado de conceptos, prácticas y criterios que nos sirven de referencia y ayudan a resolver un problema en particular (en este caso el desarrollo de un videojuego).

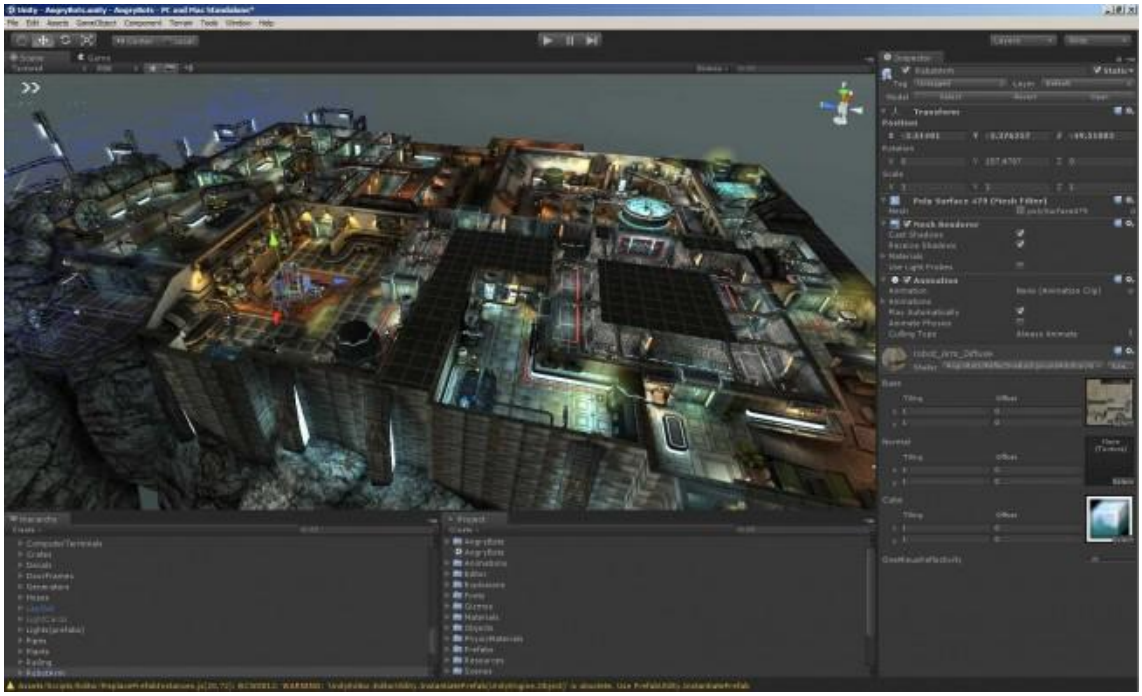


Imagen 4. Captura del "game engine" de Unity 3D

Su principal propósito es proveer al usuario o desarrollador una interfaz visual de fácil manejo, junto con un conjunto de herramientas integradas (como por ejemplo, herramientas de renderizado 2D y 3D, "physics engine", detección de colisiones, editor de sonidos, animación, "networking", administración de memoria, inteligencia artificial…). Estas suelen presentarse con una funcionalidad limitada donde, por medio de un entorno visual o a través de un lenguaje específico del "Game Engine", podemos utilizarlas sin necesidad de un conocimiento profundo en el campo o área a la que pertenecen y limitándose únicamente al uso que se le pudiera dar en un videojuego.

2.2 Historia del Game Engine

Antes de la aparición de estos "frameworks", con la comercialización de las primeras video-consolas (en 1970) los juegos se diseñaban [3] con programación a bajo nivel haciendo un uso optimizado de todos los recursos y sacando el máximo partido al hardware de estas consolas.

```
GameCalc

   lda GameStatus
   and #%10100010
   beq Move00        ;If not game over and not player dying
   jmp DontMove      ;and not button waiting, move legal

Move00
   dec FramesLeft    ;is it time to move invaders?
   beq Move01
   jmp DontMove

Move01
   ldx InvLeft       ;reset frames-left count
   lda advancetable,X
   sta FramesLeft

   lda Direction
   bne MovingLeft

   lda PixelOffset
   clc
   adc #1
   sta PixelOffset   ;move invaders right one pixel.
   cmp #3
   beq MI1
   cmp #6
   beq MI0
   jmp DontMove

MI0 lda #0           ;move invaders right one cycle.
   sta PixelOffset
MI1
   lda CycleOffset
   clc
```

Imagen 5. Fragmento del código fuente del Space Invaders de Atari

Debido a las limitaciones de memoria y procesamiento, cada juego se diseñaba considerando sus propias optimizaciones haciendo que su código fuente fuese único y por tanto imposible de reutilizar en otros juegos. Por eso era impensable en aquel entonces el desarrollo de un motor.

En consolas como la Atari o la Odyssey (conocida en España bajo el nombre de Videopac) podemos observar como el mero hecho de mostrar por pantalla el juego consumía casi todos los recursos de estas, dejando muy poco margen para la realización del resto de operaciones. Estas limitaciones se veían claramente en juegos como el ajedrez, donde para poder suplir las restricciones de las consolas de esta época, era necesaria la implementación de un módulo adicional con un procesador y memoria propios (aparte del de la consola) para poder ejecutarlo.

Imagen 6. Consola videopac junto con el módulo de ajedrez

No fue hasta 1990 cuando comienza el boom de los "game engines" con la aparición de los primeros juegos 3D, hasta entonces (en torno a principios de 1980) algunas compañías lanzaron "kits" de desarrollo para que desarrolladores pudieran diseñar sus propios juegos, facilitando así los "third party games" (ya que por regla general era la propia compañía que sacaba la consola la que desarrollaba los juegos, no existían terceros), como "Pinball Construction Set" (1983), "Adventure Construction Set" (1984), "Garry KitCHen's GameMaker Set" (1985), "Wargame Construction Set" (1986), "Shoot'Em-Up Construction Kit" (1987), "Arcade Game Construccion Kit" (1998) y uno de los más populares aún muy conocido a día de hoy el "RPG Maker" (1988).

Durante 1990 el término "game engine" estaba en estrecha conexión con los juegos tipo FPS (First person shooter). Con la aparición del 3D juegos como Doom y Quake, los desarrolladores, en vez de trabajar de nuevo desde cero, comenzaron a registrar los derechos sobre fragmentos de código y se empezó a separar el motor del juego de su contenido danto lugar de esta manera a "Game Engines" tal y como los conocemos a día de hoy. Algunos por aquel entonces se hicieron muy populares como los de los juegos de "Quake III" y "Half Life", los cuales fueron liberados de cara al público, permitiendo a los usuarios crear sus propios niveles, mapas, y versiones de estos juegos. A día de hoy, casi 20 años después siguen teniendo sus "desarrolladores" y jugadores fieles.

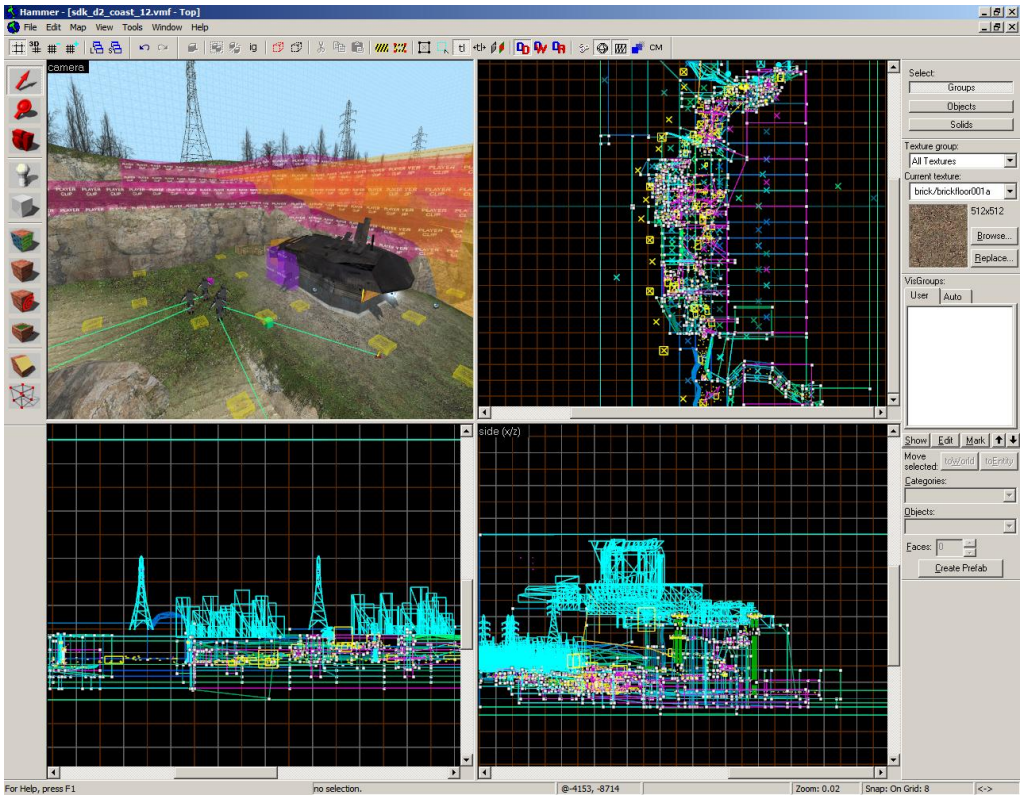


Imagen 7. Game engine: Valve Hammer liberado para su uso gratuito

2.3 Tipos de motores

En la actualidad, existen infinidad de motores de videojuegos. Estos se suelen clasificar de dos maneras diferentes según las herramientas que incluyan y el grado de programación requerido.

Según el tipo de herramienta que tengan, se suelen categorizar en [4] cuatro tipos, "2D", "3D", "Mobile" y "Game Mods":

- **2D Engine:**
  Tanto en matemáticas como en el mundo de los videojuegos, 2D es un término que hace referencia a objetos que solo poseen 2 dimensiones, es decir, alto por ancho, y por tanto comprende aquellos motores que solo pueden renderizar gráficos en dos dimensiones. Mucha gente lo confunde con la perspectiva de visión del juego. Existen juegos en 3D que usan una vista "Side Scroller" que dan un efecto 2D, y lo mismo pasa al revés, existen juegos 2D que dan la impresión de profundidad según la posición de las imágenes y mediante el uso de la perspectiva isométrica.

- **3D Engine:**
  Renderizan en tres dimensiones, permitiendo representaciones de objetos geométricos en el espacio. Por lo general todos los motores 3D incluyen también la posibilidad de crear también juegos en dos dimensiones.

- **Mobile:**
  Son motores diseñados específicamente para móviles. Con el auge del sistema operativo Android, este tipo de motores ha atraído la atención de "desarrolladores casuales" que gracias al avance de las nuevas tecnologías se les ha facilitado en gran medida la creación de videojuegos propios sin necesidad de grades conocimientos de programación.

- **Game Mods:**
  Normalmente son diseñados para modificar o manipular juegos que tienen una alta popularidad en el mercado actual. Estos motores permiten desde alterar las reglas del juego como cambiar la vida de los personajes, dinero, atravesar paredes... hasta mejorar su rendimiento, funcionalidad o apariencia, como son por ejemplo el mod de hiperrealismo del "Skyrim" o el mod de modo online multi-jugador para el "GTA 3".

Según el grado de programación y desarrollo previo requerido se pueden categorizar [5] en tres que son "Roll your own", "Mostly ready" y "Point and click", aunque también existen motores que engloban varias como es el caso de "Unity 3D":

- **Roll your own:**
  A pesar del coste y tiempo que requiera la creación de un motor de juego, muchas compañías (y también desarrolladores de juegos independientes) siguen desarrollando sus propios "game engine". Para ello, recurren a aplicaciones abiertas al público o de código abierto, tales como las API XNA, DirectX, OpenGL, las APIs de Windows y Linux o las SDL, junto con otras librerías ya sean de código abierto o comercial.
  La ventaja que posee este tipo de motores, es su completa especialización hacia el juego a desarrollar. Permite al desarrollador la selección de los componentes necesarios para el videojuego sin necesidad de sobrecargar el procesador.

- **Mostly ready:**

La mayoría de los motores de videojuegos corresponden a esta categoría. Estos motores están listos para su manejo e incluyen un gran repertorio de herramientas de renderizado, interfaz gráfica, cálculo de físicas y colisiones…
Permiten crear un juego a través de "scripts" y a través del uso de programación de bajo nivel y están optimizados para casos generales.
Ofrecen un gran rendimiento con mucho menos esfuerzo de los de "roll your own" incluso si no se ajustan a lo que realmente necesitas.

- **Point and click:**
  Son motores que te permiten de manera intuitiva y amigable crear un videojuego sin necesidad de saber programar. Mediante una interfaz visual puedes ir desarrollando un videojuego utilizando casi únicamente el ratón.

  Son los más demandados a día de hoy ya que permiten a personas con escasos conocimientos de programación desarrollar sus propios juegos.

Aunque son motores con bastantes limitaciones ya que están orientados a modelos concretos de juegos para facilitar a los usuarios su manejo, y así poder desarrollar rápidamente y sin apenas esfuerzo.

2.4 Game Engine actuales

Como se puede ver, para el desarrollo de un videojuego, es fundamental saber de antemano que se quiere realizar, sobre que plataforma, el personal implicado, el tiempo a dedicar y coste que nos supondrá. Y en base a ello elegir el motor gráfico que mejor se adapte a nuestras necesidades.

A día de hoy existen una gran variedad de motores de videojuegos. A continuación se describen aquellos con mayor relevancia en la actualidad:

2.4.1 Unity 3D

Unity [6] es un motor de videojuego multiplataforma creado por Unity Technologies. Dispone de varias versiones de pago y una gratuita, siendo su versión actual la 5.4.0 y su última versión estable la 5.3.5.

Su plataforma de desarrollo está disponible para Microsoft Windows, OS X y Linux, y permite exportar juegos a las siguientes plataformas:

- Microsoft (Windows, Xbox 360, Phone)
- OS X
- Linux
- Play Station (PS3, PS4, PS Vita)
- Nintendo (Wii, Wii U)
- Iphone
- Android

Unity Technologies fue fundada por David Helgason, Nicholas Francis y Joachim Ante en 2004 con la idea de "democratizar el desarrollo de los videojuegos", es decir, se enfoca en los desarrolladores independientes que no disponen de los conocimientos y/ o las herramientas necesarias para poder crear su propio motor de videojuegos facilitándoles una plataforma de fácil manejo con posibilidad de uso a través de licencias gratuitas.

La primera versión de Unity se lanzó en el año 2005 en la conferencia mundial de desarrolladores de Apple. Esta primera versión funcionaba únicamente bajo la plataforma mac y a diferencia de sus últimas versiones no era multiplataforma.

Con el lanzamiento de la tercera versión de Unity, con el fin de captar el interés de los desarrolladores, se centró en la introducción formatos de ficheros de herramientas de edición

gráfica que la mayoría de estudios de desarrollo tienen a su disposición e introdujo además el soporte multiplataforma.

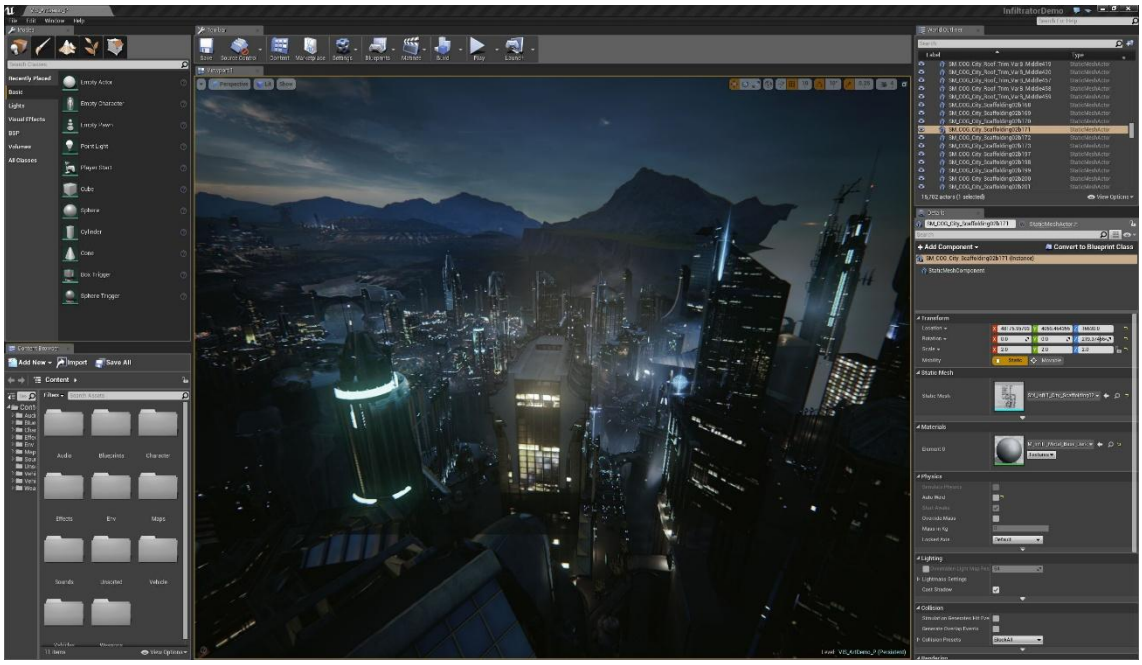
Unity puede usarse junto con Cinema 4D, 3ds Max, Blender, Modo, ZBrush, Maya, Adobe Photoshop y Adobe Fireworks. Los cambios realizados a los objetos creados con estos productos se actualizan automáticamente dentro del proyecto, sin necesidad de volver a importarlos.

Usa el motor PhysX de Nvidia y el scripting se puede realizar a través de Monodevelop o del Visual Studio. Los desarrolladores además pueden utilizar C# o JavaScript como lenguaje de programación.

Otras de sus características es Umbra. Unity maneja una de las mejores herramientas de oclusión en la industria actual. Es capaz de evitar renderizados innecesarios, haciendo hincapié en los objetos de la escena que en un momento determinado se ven ocluidos visualmente por otros.

A día de hoy es sin duda el motor gráfico más utilizado a nivel global, sobre todo con desarrolladores independientes.

2.4.2 Unreal Development Kit



Es [7] un motor multiplataforma orientado principalmente para PC y videoconsolas. Fue creado por la empresa Epic Games e implementado por primera vez en el videojuego Unreal en 1998.

A diferencia de Unity, este motor solo disponía de licencias de pago y solo estaba disponible para empresas de desarrollo. El 2 de marzo del 2015, a través de un comunicado oficial, la versión 4 de Unreal Engine se puso disponible para todo aquel que lo deseara de manera gratuita, con la condición de que cuando el juego se comercializara, Epic Games obtendría el 5% de los beneficios de la obra por trimestre cuando dicho producto superara sus primeros 3000 dólares de ganancias.

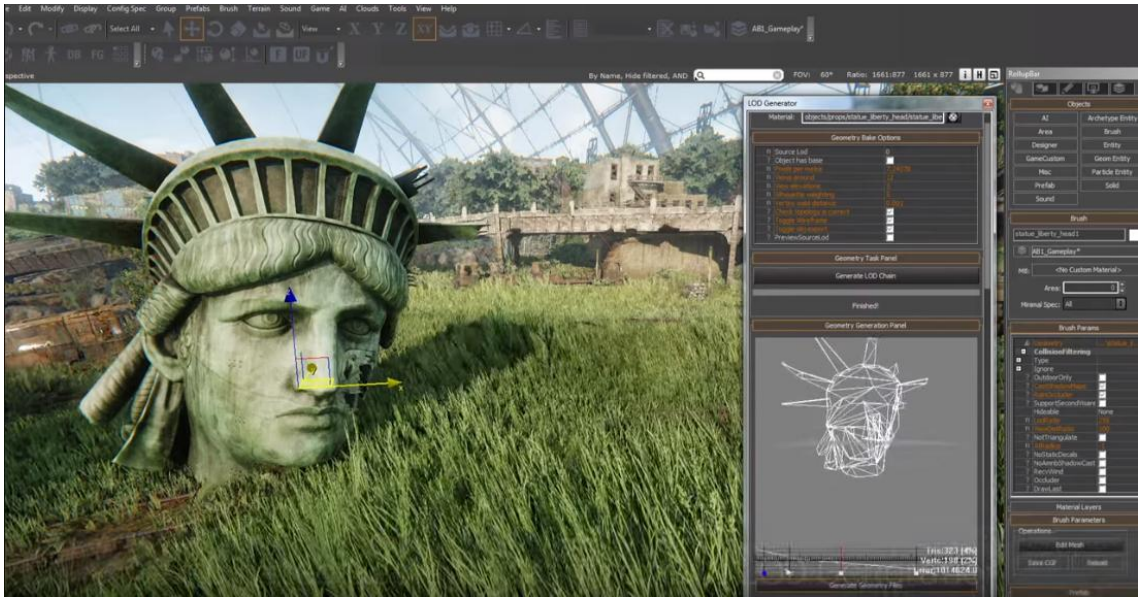
Una de las principales cualidades con las que destaca este motor es por su sistema de iluminación global en tiempo real usando "voxel cone tracing", que elimina la luz pre-computerizada y usa en su lugar usa algoritmos de física para calcular los rayos de luz según el punto de visión.

La programación se realiza en C++ y permite la edición de código en tiempo de ejecución y a diferencia de sus predecesores, esta última versión ya no dispone de soporte de desarrollo de código en UnrealScript.

Soporta renderizado avanzado con DirectX 11 y posee un sistema optimizado de materiales basados en simulación de propiedades físicas.

Gracias a la decisión de permitir su uso gratuito, este motor ha generado gran expectación entre los desarrolladores y ha comenzado a arañarle terreno a Unity 3D.

2.4.3 Cry Engine



Con la aparición de CryEngine 5, la compañía desarrolladora de este motor, Crytek, ha destacado más por su nuevo modelo de negocio que por sus características.

Este motor, se pasa al modelo de negocio de "paga lo que quieras", por lo que cualquier persona puede descargarlo aportando la cantidad de dinero que desee y usarlo sin ninguna restricción.

Se centra [8] principalmente en plataformas de videoconsolas como la Xbox 360 y la PlayStation 3 y de PC como el sistema Windows.

Usa su famoso editor SandBox, que facilita a los desarrolladores un control total sobre sus creaciones multiplataforma en tiempo real. Este editor permite la separación de capas de los diferentes niveles de juego y permite que varios desarrolladores trabajen en una misma capa sin causar impacto alguno en el trabajo del compañero.

Sus últimas novedades incluyen el desarrollo de scripts en C# y un entorno 3D optimizado para VR.

2.5 Comparativa de estos game engine

MOTOR:	UNITY 3D	UDK	CRYENGINE
PLATAFORMAS SOPORTADAS:	Mac OS, Windows, Linux, IOS, TvOS, Windows Phone, Android, Xbox 360, Xbox One, PlayStation 3, PlayStation 4, PS Vita, Wii, Wii U, WebLG, Tizen, Samsung Tv, WebPlayer.	Mac OS, Windows, IOS, Android, PlayStation 3, PS Vita, Xbox 360, Wii, WebPlayer.	Windows, PlayStation 4, Xbox One, Oculus Rift.
2D/3D	Si	Si	Si
LEGUAJE DE PROGRAMACIÓN:	C#, JS script	C++	VisualScript
LICENCIA:	Gratuita/Pago	Gratuita con restricciones a su uso comercial	Gratuita. "Paga lo que consideres"
COMUNIDAD:	Muy Grande	Normal	Normal
FACILIDAD DE MANEJO:	Normal	Compleja	Compleja

Tabla 1. Comparativa de los diferentes motores actuales

3 Gestión del proyecto

Este capítulo expone, aplicando la metodología de Ingeniería del software, el alcance, coste, planificación y gestión de este proyecto.

3.1 Perspectiva y objetivo del proyecto

Como ya se ha expuesto en el capítulo 1 (apartado motivación y objetivos), la finalidad de este proyecto la podemos resumir o englobar en dos objetivos. Por un lado lo que podemos considerar como el objetivo principal que es el desarrollo de un videojuego tipo FPS, y por otro lado el objetivo secundario que es experimentar todo el proceso de desarrollo que conlleva la realización de un videojuego desde cero junto con el aprendizaje y acercamiento a las herramientas necesarias para su desarrollo.

3.2 Planificación del proyecto

Como cualquier otro proyecto software, es necesaria una metodología en la que se definen las tareas a realizar y las etapas de las que consta el proceso de desarrollo, para de esta manera, no solo tener una mejor organización durante su realización sino también una mejora en cuanto la productividad y calidad final del producto.

En este caso se ha optado por una metodología muy común en el desarrollo de los videojuegos: la iterativa o incremental. Empezar con una versión simple o reducida de la aplicación e ir añadiendo funcionalidades nuevas y/o mejorar funcionalidades ya existentes en cada una de las iteraciones posteriores, consiguiendo así diferentes versiones del juego hasta llegar a su versión final.

Para ello, se ha optado dirigir el desarrollo de este proyecto a través del modelo de proceso OpenUP [9]. Este modelo de desarrollo pertenece a la compañía Eclipse y es un proceso de desarrollo unificado que aplica enfoques iterativos e incrementales dentro de un ciclo de vida estructurado.

Las principales características de OpenUP las podemos resumir en lo siguiente:

- Desarrollo incremental
- Uso de casos de uso y escenarios
- Manejo de riesgos
- Diseño basado en la arquitectura

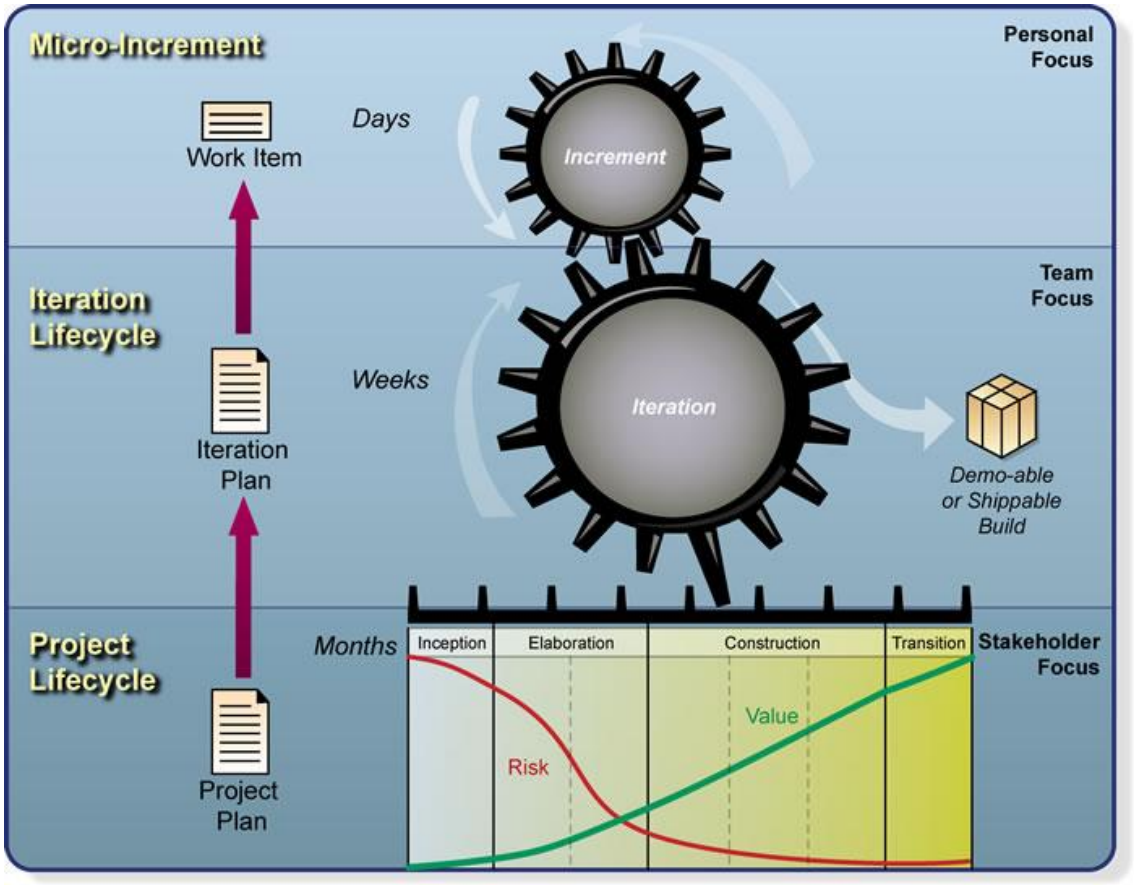


Imagen 8. Ciclo de vida de un proyecto OpenUP

El proceso de desarrollo de un proyecto mediante OpenUP [10] se realiza en base a tres fases:

- **Plan del proyecto:** Constituido a su vez por cuatro fases: Inicio, Elaboración, Construcción y Transición, permitiendo una toma de decisión acorde a cada fase mitigando e interceptando los posibles riesgos lo antes posible. Estas fases consisten en lo siguiente:
  - *Inicio:* En esta primera fase, las necesidades de cada participante son tomadas en cuenta y plasmadas en objetivos para el proyecto. Además se definen:
    - El ámbito del proyecto
    - Objetivos del proyecto
    - Personal involucrado
    - Estimación inicial del coste
    - Planificación estimada
  - *Elaboración:* Se realiza el Plan de proyecto y se especifican:
    - Los casos de uso
    - Diagramas de secuencia
    - Modelo de dominio
    - Diagrama de clases
    - Arquitectura del sistema.
  - *Construcción:* Montaje del proyecto. Esta fase consiste en la elaboración y desarrollo de las diferentes tareas asignadas al personal encargado del proyecto.

  - *Transición*: Se pasa el producto a los usuarios, es decir, se presenta el producto de prueba en sus diferentes fases de desarrollo a un número limitado de usuarios para corregir y localizar fallos, errores y defectos en cada iteración.
- **Plan de Iteración:** Define lo que debe ser entregado en cada iteración de la aplicación y el resultado es una acumulación con las iteraciones previas.
  El trabajo del personal se organiza en micro-incrementos. Estos representan unidades cortas de trabajo que se pueden realizar desde horas a varios días. Estos micro-incrementos proporcionan un bucle de retroalimentación extremadamente corto que impulsa las decisiones de adaptación en cada iteración.

3.2.1 Riesgos

El objetivo de todo proyecto es, en general, la obtención de un producto con la mayor calidad y mínimo coste a la vez que se minimiza el impacto de los riesgos que se puedan presentar.

Aspectos normalmente desconocidos o difíciles de establecer pueden incidir directamente en la consecución de las estimaciones del tiempo previsto y con el presupuesto establecido.

La gestión de riesgos [11] influye aspectos desconocidos y siempre implica dos características:

- Probabilidad
- Pérdida

Según el diccionario de la Real Academia, riesgo se define como contingencia o proximidad de un daño. Por tanto, se define la gestión de riesgos como el análisis continuo de una situación actual para adaptar la situación presente o futura que garantice una situación más favorable que la inicial.

Un método efectivo para identificar riesgos es la creación de una lista de comprobación de elementos de riesgo, sus consecuencias y alternativas para combatirlos. Esta lista debe centrarse en los riesgos relacionados con la disponibilidad del personal, la organización administrativa del proyecto, las herramientas usadas, los requerimientos estimados en el diseño y la estimación de tiempo que conlleva realizar las diferentes tareas que componen dicho proyecto.

Las estrategias para combatir los diferentes riesgos que se puedan presentar son los siguientes:

- Impedir el riesgo
- Transferir el riesgo
- Aceptarlo
- Mitigarlo
- Crear un plan de contingencia

3.2.1.1 Impacto y probabilidad de los posibles riesgos del proyecto

En la siguiente tabla, se muestran los riesgos que se pueden dar en el desarrollo del proyecto junto con la probabilidad de que se sucedan (que puede ser baja, media o alta) y el impacto que causaría de darse (tolerable, serio y catastrófico).

Riesgo	Probabilidad	Impacto
Los requisitos capturados no son correctos	Alta	Serio
Mal diseño del proyecto	Moderada	Catastrófico
Error en la implementación	Alta	Tolerable
Cambios de requisitos que precisan de modificaciones de código	Moderada	Serio
Entorno de trabajo inadecuado	Moderada	Serio
Fallo del equipo hardware usado para el desarrollo del proyecto	Baja	Tolerable
Planificación demasiado optimista	Alta	Serio
Las herramientas usadas no son aptas para el desarrollo del proyecto	Moderada	Catastrófico
El desarrollador no es capad de adquirir los conocimientos requeridos	Moderada	Catastrófico

Tabla 2. Probabilidad e impacto de los posibles riesgos del proyecto

3.2.1.2 Estrategia de gestión de riesgos

En base a los posibles riesgos localizados que se pueden dar en el proyecto, a continuación procedemos a explicar las estrategias a aplicar en cada caso:

[RG-001]: Los requisitos capturados no son correctos	
Tipo:	Riesgo de requerimientos
Probabilidad:	Alta
Impacto:	Serio
Descripción:	Las listas de requisitos no son fáciles de comprender ni hacer correctamente lo que aumenta la probabilidad de que las capturadas sean incorrectas.
Consecuencias:	Aumento del tiempo de desarrollo del proyecto.
Plan de Contingencia:	Reunirse con los jefes/clientes del proyecto, rehacer la captura de requisitos y planificar nuevamente los tiempos del proyecto.

Tabla 3. Riesgo-001

[RG-002]: Mal diseño del proyecto	
Tipo:	Riesgo de estimación
Probabilidad:	Moderada
Impacto:	Catastrófico
Descripción:	Elaboración de la propuesta de trabajo, es decir, establecer los objetivos, actividades y recursos necesarios para su correcto desarrollo.
Consecuencias:	Aumento del tiempo de desarrollo e imposibilidad de su correcto y/o completa implementación
Plan de Contingencia:	Identificar el problema de diseño. Estimar en tiempo su corrección, aumentar si es necesarios las horas dedicadas al proyecto. Corregir el defecto.

Tabla 4. Riesgo-002

[RG-003]: Error en la implementación	
Tipo:	Riesgo tecnológico
Probabilidad:	Alta
Impacto:	Tolerable
Descripción:	Realización incorrecta de parte del código de la aplicación.
Consecuencias:	Demora en el desarrollo y aparición de fallos y errores durante la ejecución de la aplicación. De no solucionarlo al momento, puede acarrear otros problemas en el uso de la aplicación.
Plan de Contingencia:	Identificar el problema en la implementación. Estimar en tiempo su corrección, aumentar si es necesario las horas dedicadas al proyecto. Corregir el defecto.

Tabla 5. Riesgo-003

[RG-004]: Cambios de requisitos que precisan modificaciones de código	
Tipo:	Riesgo de requisitos
Probabilidad:	Media
Impacto:	Bajo
Descripción:	La modificación o actualización de los requisitos influye sobre el código desarrollado de la aplicación.
Consecuencias:	Aumento del tiempo de desarrollo y corrección de parte del código ya realizado.
Plan de Contingencia:	Asumirlo Procurar que los cambios de requisitos no requieran grandes cambios de código.

Tabla 6. Riesgo-004

[RG-005]: Entorno de trabajo inadecuado	
Tipo:	
Probabilidad:	Moderada
Impacto:	Serio
Descripción:	El entorno actual donde se desarrolla diariamente el proyecto no es adecuado.
Consecuencias:	Aumento del tiempo de desarrollo y mayor probabilidad de cometer fallos y errores de código.
Plan de Contingencia:	Si las condiciones lo permiten, buscar un entorno más adecuado donde asentarse.

Tabla 7. Riesgo-005

[RG-006]: Fallo del equipo hardware usado para el desarrollo del proyecto	
Tipo:	Riesgo tecnológico
Probabilidad:	Baja
Impacto:	Tolerable
Descripción:	El equipo informático con el que se desarrolla diariamente el proyecto deja de funcionar.
Consecuencias:	Pérdida de información. Necesidad de instalar de nuevo las herramientas de desarrollo en otro equipo.
Plan de Contingencia:	Tener a disposición otro equipo con las herramientas ya instaladas y toda la información sincronizada a través de internet.

Tabla 8. Riesgo-006

[RG-007]: Planificación demasiado optimista	
Tipo:	Riesgo de estimación
Probabilidad:	Alta
Impacto:	Serio
Descripción:	La planificación planteada para el desarrollo de este proyecto es imposible de cumplir.
Consecuencias:	Imposibilidad de presentar el proyecto a tiempo.
Plan de Contingencia:	Acortar o limitar el alcance del proyecto.

Tabla 9. Riesgo-007

[RG-008]: Las herramientas usadas no son aptas para el desarrollo del proyecto	
Tipo:	Riesgo de herramientas
Probabilidad:	Moderada
Impacto:	Catastrófico
Descripción:	Las herramientas utilizadas para el desarrollo de este proyecto no sirven para crear la aplicación deseada
Consecuencias:	Aumento del tiempo de desarrollo. Posibilidad de tener que replantear de nuevo las bases de este proyecto.
Plan de Contingencia:	Estudio previo de las herramientas a utilizar.

Tabla 10. Riesgo-008

[RG-009]: El desarrollador no es capad de adquirir los conocimientos requeridos	
Tipo:	Riesgo de personal
Probabilidad:	Moderada
Impacto:	Catastrófico
Descripción:	El desarrollador no es capaz de adquirir los conocimientos necesarios para poder realizar este proyecto.
Consecuencias:	Imposibilidad de realizar el proyecto.
Plan de Contingencia:	Abandono del proyecto y realización de una propuesta que se adecue a las capacidades del desarrollador.

Tabla 11. Riesgo-009

4 Plan del proyecto

Siguiendo la metodología OpenUP, en este capítulo se establecen los diferentes apartados que forman el plan de proyecto. Dado que el ámbito y objetivos del proyecto ya han sido clarificados previamente en la introducción de esta memoria, procederemos con a partir del siguiente punto.

4.1 Identificación del personal involucrado en el proyecto y sus responsabilidades

A continuación se especifican las personas involucradas en este proyecto y sus roles:

- **Impulsores del proyecto:** tutor: Benjamín Sahelices Fernández y el estudiante: Borja Cerezo Redondo
- **Beneficiarios:** Universidad de Valladolid y la sociedad en general
- **Usuarios:** abierto a todo el público en general
- **Desarrollador:** estudiante Borja Cerezo Redondo.

Para el correcto desarrollo del proyecto se requiere por un lado usuarios con las siguientes cualidades:

- Tener experiencia en el manejo de juegos online FPS
- Ser capaces de localizar y distinguir fallos, defectos y errores en un videojuego
- Aportar sugerencias de mejora de apariencia y funcionalidad

En cuanto al desarrollador encargado de la realización del proyecto, debe de ser capaz de:

- Controlar las diferentes herramientas necesarias para el desarrollo de la aplicación
- Realización de las tareas establecidas en cada iteración del proyecto
- Cumplimiento con la planificación estimada
- Adecuación ante situaciones imprevistas

4.2 Costes del proyecto

En este apartado especificamos los costes estimados del desarrollo de este proyecto, los cuales dividimos en tres tipos de gastos; de hardware, software y personal.

4.2.1 Costes hardware

En el caso del coste hardware, usaremos como referencia el ordenador sobre el que se ha trabajado para el desarrollo de este proyecto que aparte de no solo cumplir con los requisitos necesarios para el uso de las herramientas software, gracias a que dispone de una tarjeta gráfica Nvidia avanzada, es óptimo para el uso de algunas de estas mismas.

El equipo hardware es un portátil **ASUS R510JK** que consta por tanto de lo siguiente:

▪ **Intel Core i5-4200H 3.0Ghz**	
▪ **Nvidia GeForce GTX850M**	
▪ **Memoria RAM 6GB DDR3 1600Mhz**	
▪ **Disco duro 500GB 5400rpm SATA WD**	
▪ **Display 15.6" Retroiluminación LED**	
▪ **Conectividad Wifi 802.11g/n y Bluetooth 4.0**	
▪ **Batería 4 celdas Ion de litio**	
	Total: 569,00€

4.2.2 Costes software

En el caso de las herramientas software se han utilizado las siguientes:

Unity 3D "Licencia Gratuita"	**0,00€**
Cinema 4D R12 "Licencia Estudiante"	0,00€
LibreOffice	0,00€
Adobe photoshop CS2 "Licencia Gratuita"	0,00€
Astah Profesional "Licencia Estudiante"	0,00€
	Total: 0,00€

4.2.3 Costes de personal

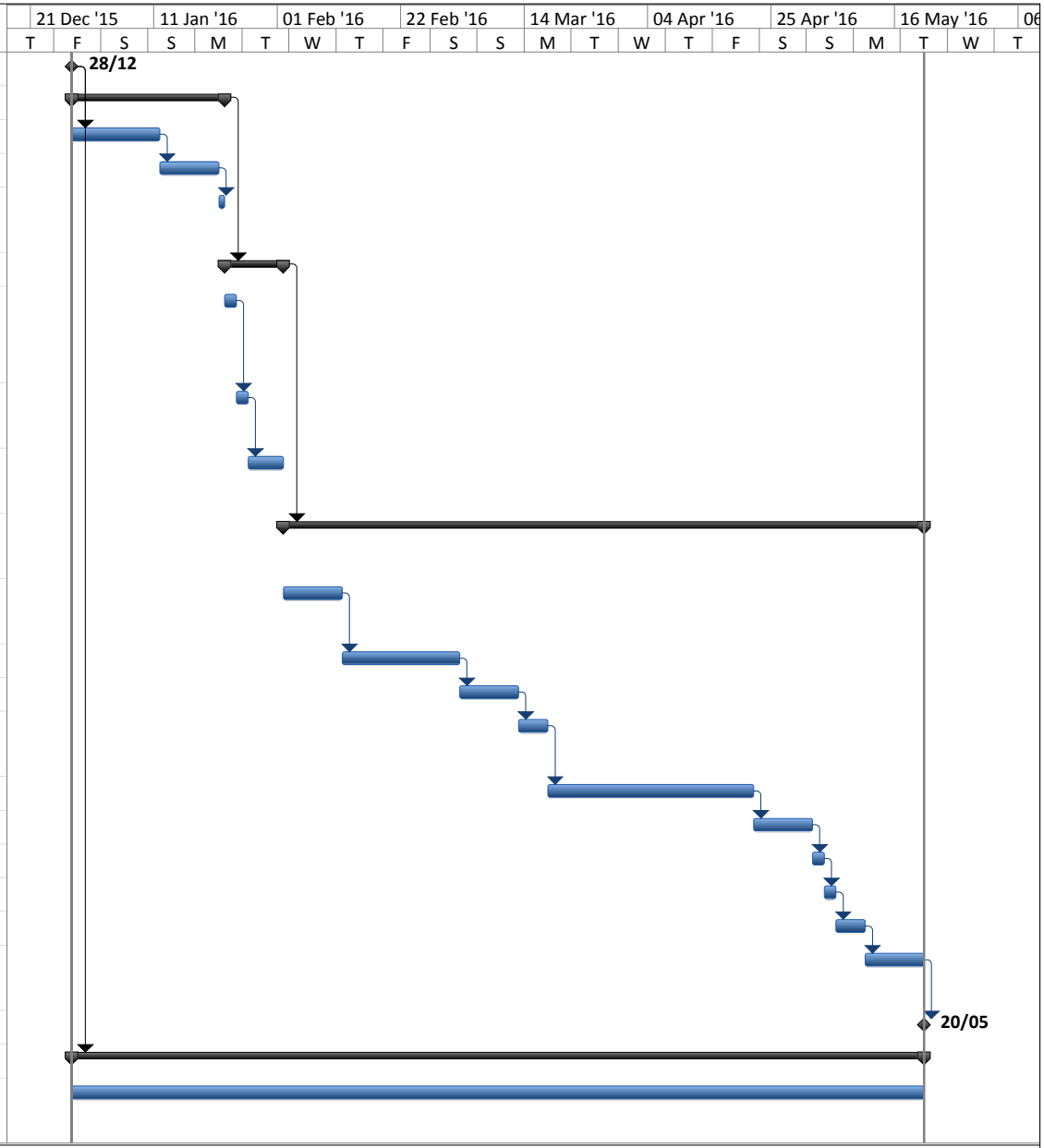
Según el convenio colectivo en vigor para el sector de Tecnologías de la Información y la comunicación, un Ingeniero Informático recién incorporado a una empresa cobra un salario base de 1569.25€/mes (unos 52€ diarios). En base al tiempo que ha llevado la realización de este proyecto (el cual se puede observar en la tabla de planificación real que se muestran en las siguientes páginas de esta memoria) el coste sin contar el tiempo de desarrollo de la documentación sería el siguiente:

Coste del proyecto:	
Tiempo de formación (31días)	1612,00€
Gestión del proyecto (12 días)	624,00€
Desarrollo Iteración 1 (20 días)	1040,00€
Desarrollo Iteración 2 (49 días)	2584,00€
Desarrollo Iteración 3 (52 días)	2704,00€
Pruebas de funcionamiento (2 días)	104,00€
Corrección de errores (25 días)	1300,00€
	Total: 9968,00

4.3 Planificación del proyecto

A continuación se muestras la planificación inicial estimada para la realización del proyecto y la finalmente realizada:

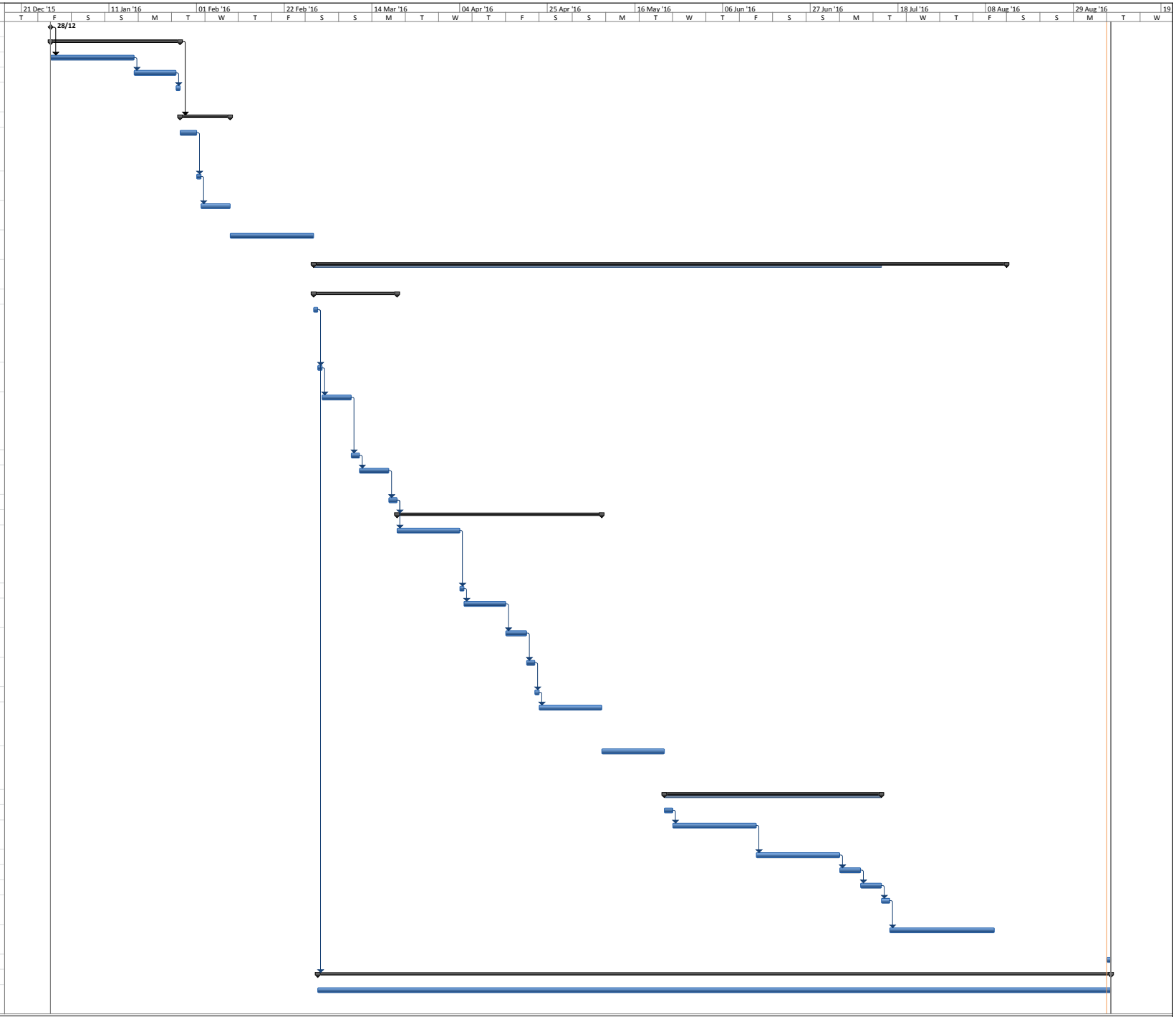
ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		Inicio	0 days	Mon 28/12/15	Mon 28/12/15	
2		**Formación**	**26 days**	**Mon 28/12/15**	**Fri 22/01/16**	
3		Estudio Unity	15 days	Mon 28/12/15	Mon 11/01/16	1
4		Estudio Cinema 4D	10 days	Tue 12/01/16	Thu 21/01/16	3
5		Estudio Adobe Photoshop	1 day	Fri 22/01/16	Fri 22/01/16	4
6		**Gestión del proyecto**	**10 days**	**Sat 23/01/16**	**Mon 01/02/16**	**2**
7		Selección y estudio de la metodología a seguir	2 days	Sat 23/01/16	Sun 24/01/16	
8		Planificación del proyecto	2 days	Mon 25/01/16	Tue 26/01/16	7
9		Realización de un plan de proyecto	6 days	Wed 27/01/16	Mon 01/02/16	8
10		**Desarrollo de la aplicación**	**109 days**	**Tue 02/02/16**	**Fri 20/05/16**	**6**
11		Desarrollo motor de juego	10 days	Tue 02/02/16	Thu 11/02/16	
12		Diseño de personajes	20 days	Fri 12/02/16	Wed 02/03/16	11
13		UNET	10 days	Thu 03/03/16	Sat 12/03/16	12
14		Animaciones y efectos especiales	5 days	Sun 13/03/16	Thu 17/03/16	13
15		Diseño de mapas	35 days	Fri 18/03/16	Thu 21/04/16	14
16		Menu de juego	10 days	Fri 22/04/16	Sun 01/05/16	15
17		Audio	2 days	Mon 02/05/16	Tue 03/05/16	16
18		Correccion de fallos	2 days	Wed 04/05/16	Thu 05/05/16	17
19		Pruebas de uso	5 days	Fri 06/05/16	Tue 10/05/16	18
20		Mejoras en base a las pruebas	10 days	Wed 11/05/16	Fri 20/05/16	19
21		Fin	0 days	Fri 20/05/16	Fri 20/05/16	20
22		**Documentación**	**145 days**	**Mon 28/12/15**	**Fri 20/05/16**	**1**
23		Desarrollo de la memoria	145 days	Mon 28/12/15	Fri 20/05/16	



Project: Planificación Estimada
Date: Fri 20/12/15

Task	External Tasks	Manual Task	Finish-only		
Split	External Milestone	Duration-only	Deadline		
Milestone	Inactive Task	Manual Summary Rollup	Progress		
Summary	Inactive Milestone	Manual Summary			
Project Summary	Inactive Summary	Start-only			

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		Inicio	0 days	Mon 28/12/15	Mon 28/12/15	
2		**Formación**	**31 days**	**Mon 28/12/15**	**Wed 27/01/16**	
3		Estudio Unity	20 days	Mon 28/12/15	Sat 16/01/16	1
4		Estudio Cinema 4D	10 days	Sun 17/01/16	Tue 26/01/16	3
5		Estudio Adobe Photoshop	1 day	Wed 27/01/16	Wed 27/01/16	4
6		**Gestión del proyecto**	**12 days**	**Thu 28/01/16**	**Mon 08/02/16**	2
7		Selección y estudio de la metodología a seguir	4 days	Thu 28/01/16	Sun 31/01/16	
8		Planificación del proyecto	1 day	Mon 01/02/16	Mon 01/02/16	7
9		Realización de un plan de proyecto	7 days	Tue 02/02/16	Mon 08/02/16	8
10		Interrupción temporal del proyecto	20 days	Tue 09/02/16	Sun 28/02/16	
11		**Desarrollo de la aplicación**	**166 days**	**Mon 29/02/16**	**Fri 12/08/16**	
12		**Iteración 1**	**20 days**	**Mon 29/02/16**	**Sat 19/03/16**	
13		Creación de un mapa básico y personajes de juego	1 day	Mon 29/02/16	Mon 29/02/16	
14		Movimiento del personaje	1 day	Tue 01/03/16	Tue 01/03/16	13
15		UNET: Sincronización del movimiento de personajes en red	7 days	Wed 02/03/16	Tue 08/03/16	14
16		Volar	2 days	Wed 09/03/16	Thu 10/03/16	15
17		Disparar y recibir daño	7 days	Fri 11/03/16	Thu 17/03/16	16
18		Morir y reaparecer	2 days	Fri 18/03/16	Sat 19/03/16	17
19		**Iteración 2**	**49 days**	**Sun 20/03/16**	**Sat 07/05/16**	18
20		Modelo 3D del jugador y animaciones de movimiento	15 days	Sun 20/03/16	Sun 03/04/16	18
21		Mirilla	1 day	Mon 04/04/16	Mon 04/04/16	20
22		Capas de vista para la cámara	10 days	Tue 05/04/16	Thu 14/04/16	21
23		Efectos de disparo y explosiones	5 days	Fri 15/04/16	Tue 19/04/16	22
24		Limitación tiempo de vuelo	2 days	Wed 20/04/16	Thu 21/04/16	23
25		Bloqueo del ratón	1 day	Fri 22/04/16	Fri 22/04/16	24
26		Flotar sobre cualquier superficie	15 days	Sat 23/04/16	Sat 07/05/16	25
27		Interrupción temporal del proyecto	15 days	Sun 08/05/16	Sun 22/05/16	
28		**Iteración 3**	**52 days**	**Mon 23/05/16**	**Wed 13/07/16**	
29		Matchmakin	2 days	Mon 23/05/16	Tue 24/05/16	
30		ServerList, hosting, joining	20 days	Wed 25/05/16	Mon 13/06/16	29
31		Mapas	20 days	Tue 14/06/16	Sun 03/07/16	30
32		Menú de pausa	5 days	Mon 04/07/16	Fri 08/07/16	31
33		Menú de juego	5 days	Sat 09/07/16	Wed 13/07/16	32
34		Pruebas de funcionamiento	2 days	Thu 14/07/16	Fri 15/07/16	33
35		Corrección de errores y optimización de código	25 days	Sat 16/07/16	Tue 09/08/16	34
36		Fin	1 day	Tue 06/09/16	Tue 06/09/16	
37		**Documentación**	**190 days**	**Tue 01/03/16**	**Tue 06/09/16**	13
38		Desarrollo de la memoria	190 days	Tue 01/03/16	Tue 06/09/16	



Project: Planificación Real
Date: Tue 06/09/16

Task | Milestone | Project Summary | External Milestone | Inactive Milestone | Manual Task | Manual Summary Rollup | Start-only | Deadline
Split | Summary | External Tasks | Inactive Task | Inactive Summary | Duration-only | Manual Summary | Finish-only | Progress

4.4 Análisis de la aplicación

Esta es una de las fases más importantes del desarrollo de software. Nos permite identificar y clasificar las estructuras y funcionalidades de la aplicación, proporcionándonos una guía de referencia para su desarrollo.

4.4.1 Actores

Los actores definen los diferentes roles con los que se puede interactuar con el sistema. En el caso de la aplicación a desarrollar, el único actor por el que se describen los casos de uso será obviamente el jugador o usuario encargado de manejarla.

Act-001	**Usuario**
Versión	1.0
Descripción	Este actor representa a un usuario del sistema

Tabla 12. Act-001

Act-002	**Jugador**
Versión	1.0
Descripción	Este actor representa a un usuario del sistema

Tabla 13. Act-002

4.4.2 Análisis de requisitos

A continuación se describen los requisitos funcionales y no funcionales de la aplicación. Antes de proceder a describir cada requisito se va a proceder a explicar la estructura y campos que forma las tablas de requisitos.

Cada requisito está se específica en una tabla de la siguiente forma:

<ID>	**<nombre>**
Versión	
Descripción	
Prioridad	
Necesidad	

Tabla 14. Tabla ejemplo de requisitos

Donde cada campo significa lo siguiente:

- **Identificador:** Código que identifica de forma unívoca el requisito
- **Versión:** número de versión o revisión del requisito.
- **Nombre:** Título descriptivo del requisito.
- **Descripción:** Explicación clara y directa del requisito
- **Prioridad:** Orden de prioridad del requisito. Puede ser alta, media o baja. A mayor prioridad, mayor necesidad de su realización.
- **Necesidad:** Interés en la realización de ese requisito. Puede ser esencial, deseable o opcional.

4.4.2.1 Requisitos funcionales

FRQ-001	**Iniciar partida**
Versión	1.0
Descripción	El sistema debe de permitir al usuario entre crear una nueva partida o unirse a una ya creada.
Prioridad	Alta
Necesidad	Esencial

Tabla 15. FRQ-001

FRQ-002	**Salir de la aplicación**
Versión	1.0
Descripción	La aplicación debe de ofrecer la capacidad de salir y cerrar esta misma al usuario. El sistema deberá de liberar los recursos que se estuviese utilizando por la aplicación para que el sistema operático se los pueda asignar a otros procesos.
Prioridad	Alta
Necesidad	Esencial

Tabla 16. FRQ-002

FRQ-003	**Abandonar partida**
Versión	1.0
Descripción	El usuario debe de poder abandonar la partida en cualquier momento sin afectar al resto de jugadores, a excepción de ser este el creador de la partida, en cuyo caso todos los jugadores presentes en dicha partida serán expulsados.
Prioridad	Alta
Necesidad	Esencial

Tabla 17. FRQ-003

FRQ-004	**Configuración**
Versión	1.0
Descripción	El usuario deberá de poder personalizar las siguientes características desde un menú de configuración: • Controles de movimiento • Opciones de sonido del juego • Opciones de video del juego • Opciones de pantalla del juego
Prioridad	Baja
Necesidad	Deseable

Tabla 18. FRQ-004

FRQ-005	**Selección de mapa**
Versión	1.0
Descripción	El usuario, al crear una nueva partida, podrá elegir entre uno de los diferentes mapas del juego.
Prioridad	Media
Necesidad	Deseable

Tabla 19. FRQ-005

FRQ-006	**Desplazar personaje**
Versión	1.0
Descripción	El usuario podrá desplazar el personaje que controlará a través de las teclas de movimiento asignadas.
Prioridad	Alta
Necesidad	Esencial

Tabla 20. FRQ-006

FRQ-007	**Mover cámara**
Versión	1.0
Descripción	El usuario podrá mover la cámara en primera persona del personaje a través del ratón del ordenador.
Prioridad	Alta
Necesidad	Esencial

Tabla 21. FRQ-007

FRQ-008	**Personalizar personaje**
Versión	1.0
Descripción	El usuario deberá de ser capaz de personalizar y cambiar el nombre de jugador y el color de personaje antes de comenzar una partida.
Prioridad	Baja
Necesidad	Opcional

Tabla 22. FRQ-008

FRQ-009	**Atacar**
Versión	1.0
Descripción	El usuario deberá poder atacar y eliminar a sus adversarios dentro del juego a través de las armas que se facilitan en el juego.
Prioridad	Alta
Necesidad	Esencial

Tabla 23. FRQ-009

FRQ-010	**Visualizar ranking/estadísticas**
Versión	1.0
Descripción	La aplicación deberá de disponer de un menú que se pueda visualizar en todo momento dentro de la partida para observar el ranking de los jugadores en base al número de "muertes".
Prioridad	Baja
Necesidad	Opcional

Tabla 24. FRQ-010

FRQ-011	**Menú principal**
Versión	1.0
Descripción	La aplicación dispondrá de una menú principal desde donde el usuario podrá elegir entre las siguientes opciones: • Iniciar partida • Configuración • Salir del juego
Prioridad	Media
Necesidad	Esencial

Tabla 25. FRQ-011

FRQ-012	**Respawn**
Versión	1.0
Descripción	Cuando la vida del personaje de un jugador llegue a cero, este realizará un efecto de explosión y reaparecerá de manera aleatoria al cabo de unos segundos en uno de los puntos de comienzo del mapa.
Prioridad	Alta
Necesidad	Esencial

Tabla 26. FRQ-012

FRQ-013	**Reproducción de música y efectos sonoros**
Versión	1.0
Descripción	El sistema deberá de poder reproducir música y efectos de sonido relacionados con las diferentes acciones y animaciones de los componentes del juego.
Prioridad	Media
Necesidad	Esencial

Tabla 27. FRQ-013

FRQ-014	Guardar datos y estadísticas
Versión	1.0
Descripción	El sistema guardará los datos de configuración y personalización del usuario, junto con sus estadísticas de juego.
Prioridad	Baja
Necesidad	Opcional

Tabla 28. FRQ-014

FRQ-015	Selección de armas
Versión	1.0
Descripción	El usuario podrá elegir entre las diferentes armas facilitadas por el juego.
Prioridad	Baja
Necesidad	Opcional

Tabla 29. FRQ-015

FRQ-016	Barra de vida
Versión	1.0
Descripción	El usuario podrá activar y desactivar la visualización de las barras de vida de los enemigos.
Prioridad	Baja
Necesidad	Opcional

Tabla 30. FRQ-016

FRQ-017	Volar
Versión	1.0
Descripción	El personaje del usuario deberá de ser capaz de volar al pulsar el control correspondiente durante un tiempo limitado
Prioridad	Baja
Necesidad	Opcional

Tabla 31. FRQ-017

FRQ-018	Visualizar animaciones
Versión	1.0
Descripción	El sistema deberá de ser capaz de mostrar por pantalla los movimientos y animaciones de todos los personajes presentes en la partida.
Prioridad	Alta
Necesidad	Esencial

Tabla 32. FRQ-018

4.4.2.2 Requisitos no funcionales:

NFR-001	Lenguaje de programación
Versión	1.0
Descripción	El sistema deberá desarrollarse utilizando C#
Prioridad	Alta
Necesidad	Deseable

Tabla 33. NFR-001

NFR-002	Compatibilidad con diferentes plataformas
Versión	1.0
Descripción	El sistema deberá de funcionar sobre diferentes sistemas operativos y plataformas de video-consolas.
Prioridad	Baja
Necesidad	Opcional

Tabla 34. NFR-002

NFR-003	Resolución de pantalla
Versión	1.0
Descripción	La aplicación será compatible con diferentes modos y resoluciones de pantalla.
Prioridad	Media
Necesidad	Deseable

Tabla 35. NFR-003

NFR-004	Textos legibles
Versión	1.0
Descripción	El sistema adaptará el tamaño de fuente de los textos en función de la resolución de pantalla, siendo estos en todo momento legibles para el usuario.
Prioridad	Media
Necesidad	Deseable

Tabla 36. NFR-004

NFR-005	Interfaz intuitiva
Versión	1.0
Descripción	La aplicación deberá de ser intuitiva y sencilla para que cualquier usuario pueda utilizarla sin esfuerzo alguno.
Prioridad	Alta
Necesidad	Esencial

Tabla 37. NFR-005

NFR-006	**Control de errores**
Versión	1.0
Descripción	La versión de desarrollo dispondrá de mensajes de notificación para controlar y erradicar los posibles errores que se presenten durante su desarrollo y pruebas de uso.
Prioridad	Alta
Necesidad	Esencial

Tabla 38. NFR-006

NFR-007	**Fluidez**
Versión	1.0
Descripción	El sistema garantizará la óptima fluidez del juego.
Prioridad	Alta
Necesidad	Esencial

Tabla 39. NFR-007

NFR-008	**Lock Mouse**
Versión	1.0
Descripción	La aplicación deberá de impedir que el puntero del ratón no se salga de la pantalla del juego.
Prioridad	Media
Necesidad	Deseable

Tabla 40. NFR-008

NFR-009	**Ranking**
Versión	1.0
Descripción	El sistema deberá garantizar que el ranking está actualizado en todo momento.
Prioridad	Baja
Necesidad	Opcional

Tabla 41. NFR-009

NFR-010	**Visualización de los gráficos**
Versión	1.0
Descripción	La aplicación garantizará la correcta visualización de los gráficos desde cualquier dispositivo.
Prioridad	Baja
Necesidad	Opcional

Tabla 42. NFR-010

4.4.3 Modelo de dominio

A continuación se muestra el diagrama correspondiente al modelo de dominio:

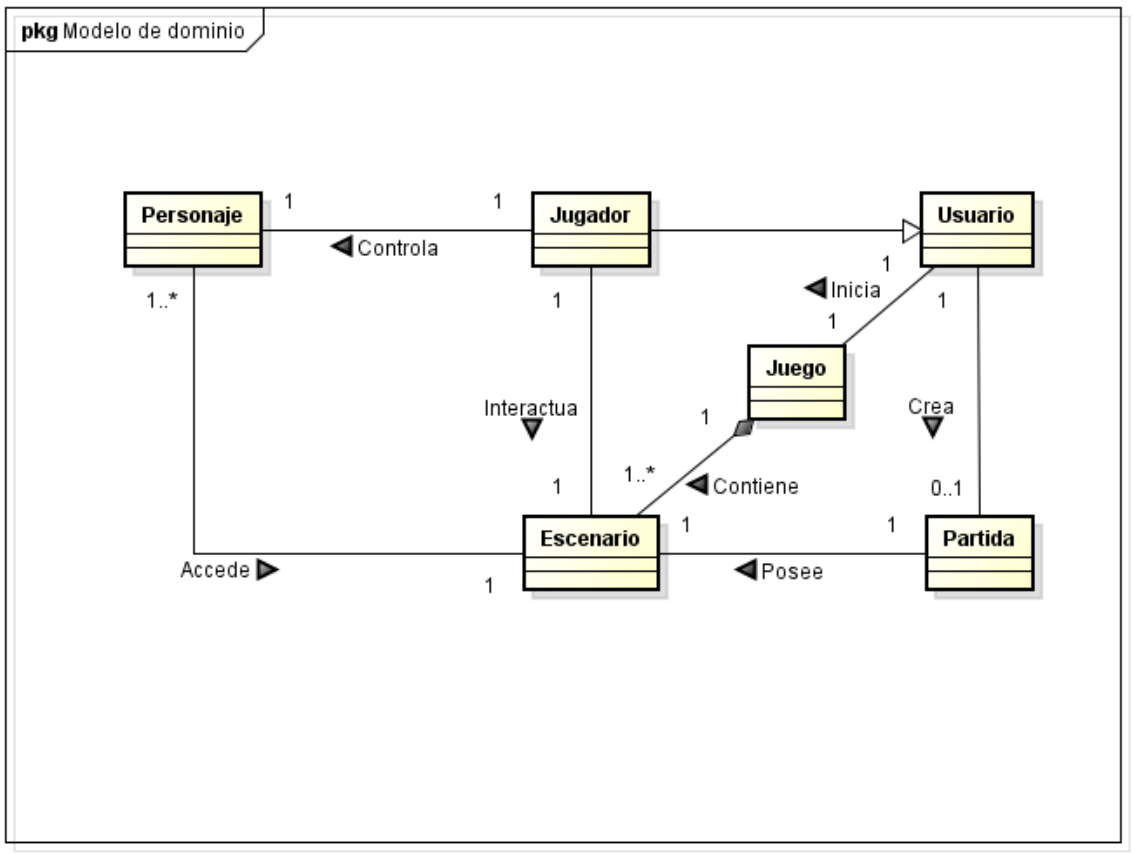


Imagen 9. Diagrama de modelo de dominio

4.4.4 Diagrama de clases de análisis

Dado que Unity separa completamente los "niveles" de un juego a través de los "scenes" o escenarios, para esta aplicación se usarán dos tipos de escenarios, una para el menú de del juego, y otra para el motor del juego dentro de un nivel o mapa:

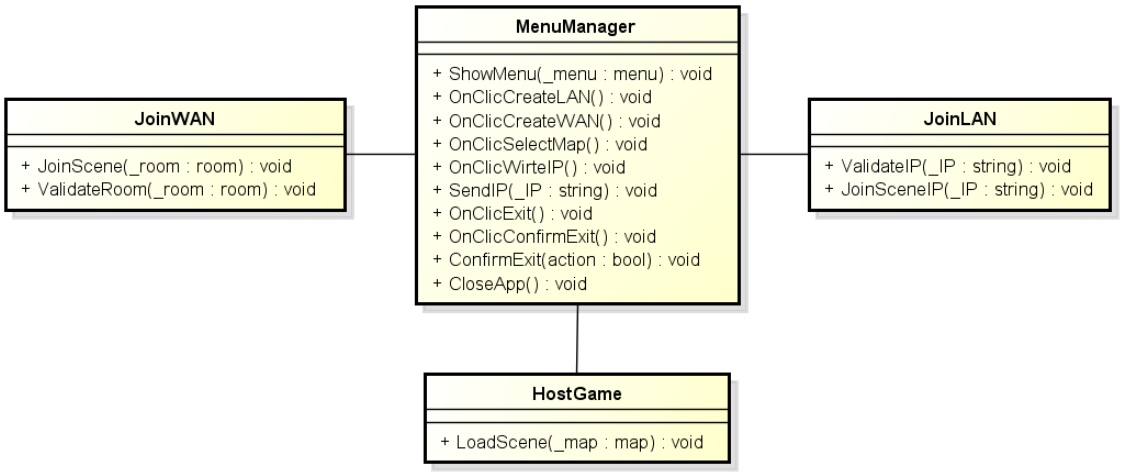


Imagen 10. Diagrama de clases de análisis del menú de juego

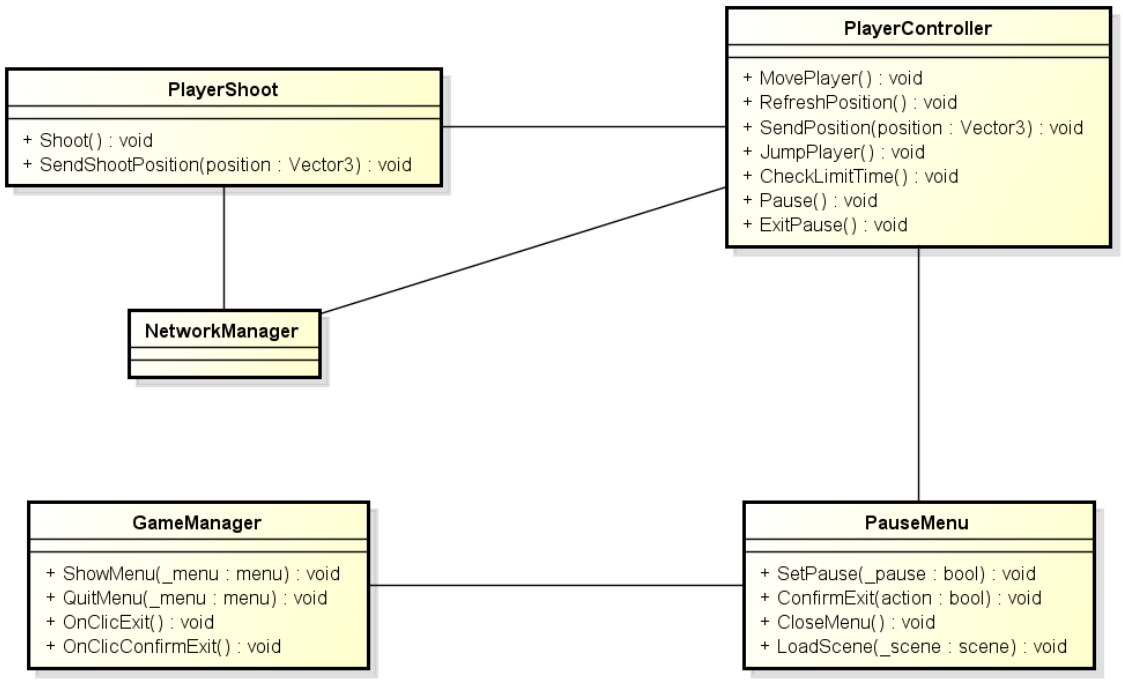


Imagen 11. Diagrama de clases de análisis del motor de juego

4.4.5 Casos de uso

En este apartado se van a detallar los diferentes casos de uso que se han definido durante la realización del análisis del proyecto. Estos casos de uso cubren las diferentes funcionalidades que pueden realizar los usuarios que interactúen con la aplicación.

4.4.5.1 Especificación de los casos de uso

Antes de proceder con lo casos de uso, se va a detallar el formato de las tablas utilizadas para definir los diferentes casos de uso:

<ID>	**<Nombre>**
Versión	
Actor	
Descripción	
Precondición	
Secuencia	
Postcondición	
Excepciones	

Tabla 43. Tabla de ejemplo de Casos de uso

Campos de la tabla:

- **Identificador:** Código que identifica de forma unívoca el caso de uso.
- **Versión:** número de versión o revisión del caso de uso.
- **Actor:** tipo de usuario de la aplicación.
- **Nombre:** Título descriptivo del caso de uso.

- **Descripción:** Explicación clara y directa del caso de uso.
- **Precondición:** Condiciones que se deben de cumplir previamente para poder realizar la operación descrita del caso de uso actual.
- **Secuencia:** Explicación de los pasos necesarios para realizar el caso de uso actual.
- **Postcondición:** Estado que presenta el sistema tras la ejecución de una determinada operación.
- **Excepciones:** Casos en los que no se cumple el caso de uso especificado.

4.4.5.2 Casos de uso "Scene" Menu

En la ilustración mostrada a continuación se puede observar los casos de uso que puede realizar un usuario a través del menú de la aplicación:

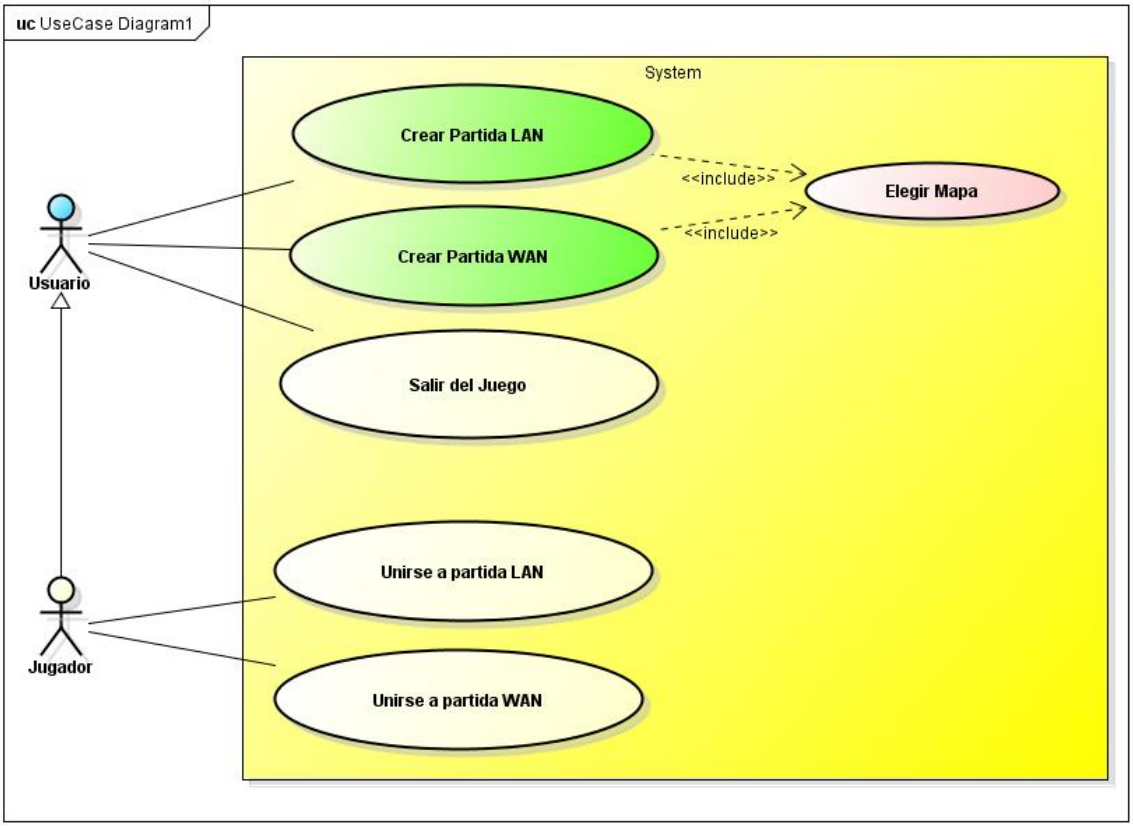


Imagen 12. Casos de uso Diagrama 1

UC-001	**Crear partida en red local**
Versión	1.0
Actor	Usuario
Descripción	El actor podrá crear una partida en red local en cualquier momento.
Precondición	Ninguna
Secuencia	1- El actor selecciona el botón "Crear partida en red privada" 2- El sistema facilitará al usuario de la elección de un mapa para crear la partida. 3- El actor elige uno de los diferentes mapas disponibles. 4- El actor presiona el botón crear partida.
Postcondición	El sistema carga la "scene" de juego del mapa elegido y crea además las funcionalidades LAN correspondientes.
Excepciones	Si el actor ya está en una "scene" de juego, no puede crear una nueva partida hasta que abandone la actual.

Tabla 44. UC-001

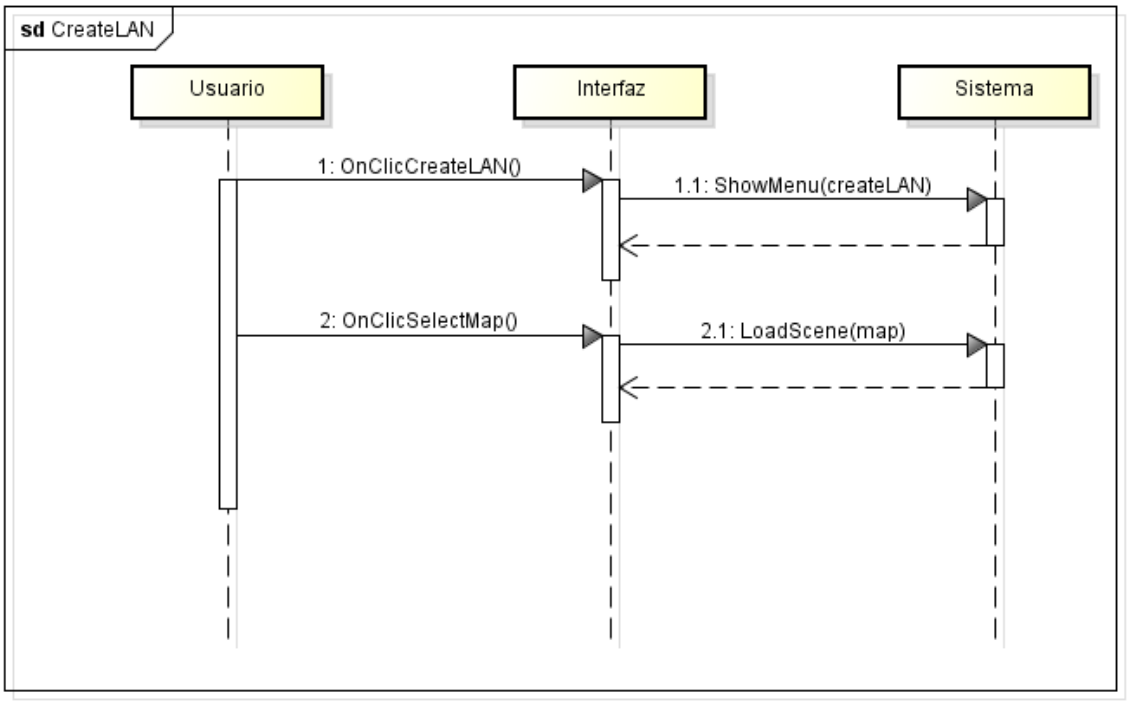


Imagen 13. Diagrama de secuencia de UC-001

UC-002	**Unirse a partida en red local**
Versión	1.0
Actor	Usuario
Descripción	El actor podrá unirse a una partida en red local en cualquier momento.
Precondición	Conocer la dirección IP del usuario que ha creado previamente la partida.
Secuencia	1- El actor selecciona el botón "Unirse a partida en red privada" 2- El sistema solicitará la introducción de la dirección IP de la partida a unirse. 3- El actor introduce los datos y acepta.
Postcondición	El sistema se une a la partida en red correspondiente a dicha dirección IP.
Excepciones	Si la dirección IP es errónea, el sistema solicita de nuevo la introducción de datos. Si el actor ya está en una "scene" de juego, no puede unirse a una nueva partida hasta que abandone la actual.

Tabla 45. UC-002

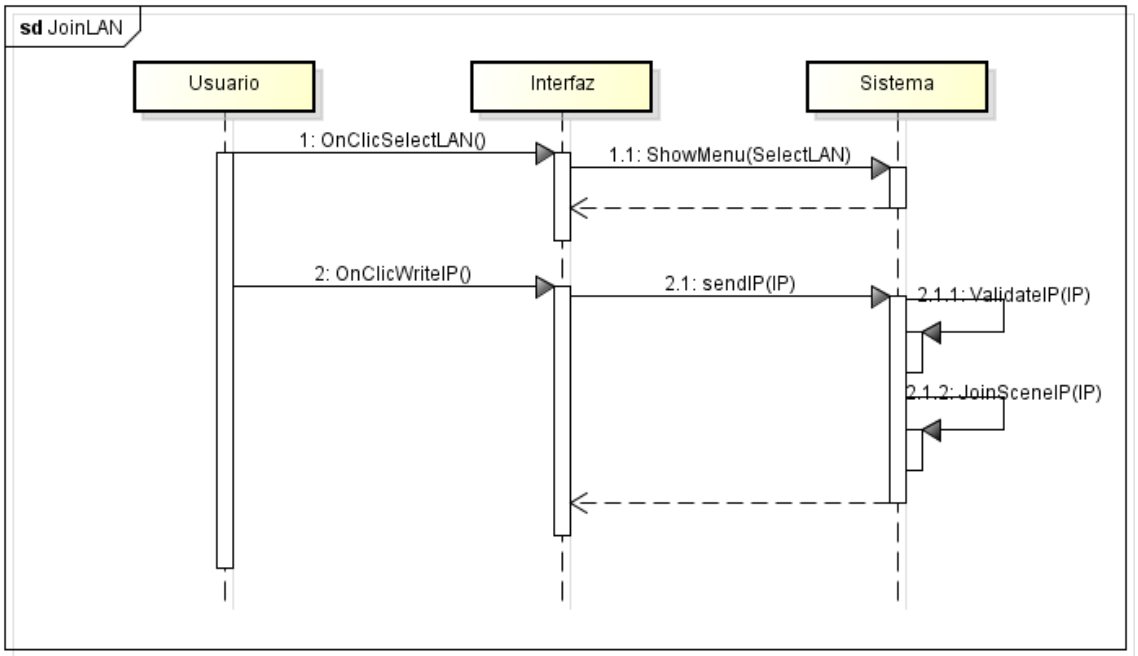


Imagen 14. Diagrama de secuencia de UC-002

UC-003	**Crear sala online**
Versión	1.0
Actor	Usuario
Descripción	El actor podrá crear una sala Online en cualquier momento.
Precondición	Ninguna
Secuencia	1- El actor selecciona el botón "Crear sala Online" 2- El sistema facilitará al usuario de la elección de un mapa para crear la sala. 3- El usuario elige uno de los diferentes mapas disponibles. 4- El usuario presiona el botón crear partida.
Postcondición	El sistema carga la "scene" de juego del mapa elegido y crea además las funcionalidades WAN correspondientes.
Excepciones	Si el jugador ya está en una "scene" de juego, no puede crear una nueva partida hasta que abandone la actual.

Tabla 46. UC-003

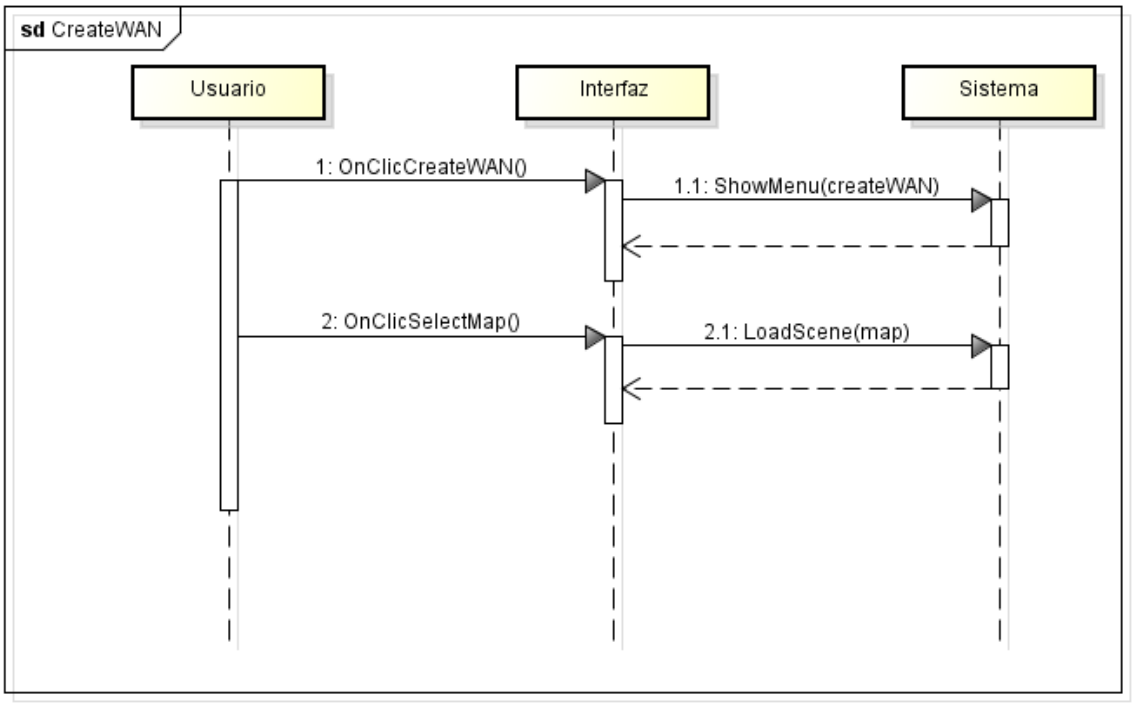


Imagen 15. Diagrama de secuencia de UC-003

UC-004	**Unirse a sala online**
Versión	1.0
Actor	Usuario
Descripción	El actor podrá unirse a una partida Online en cualquier momento.
Precondición	Debe de existir al menos una sala a la que poder unirse.
Secuencia	1- El actor selecciona el botón "Unirse a sala" 2- El sistema accede a la sala seleccionada.
Postcondición	El sistema se une a la partida Online correspondiente a la sala seleccionada.
Excepciones	Si la sala está llena, el sistema solicita la selección de otra sala diferente. Si el actor ya está en una "scene" de juego, no puede unirse a una nueva partida hasta que abandone la actual.

Tabla 47. UC-004

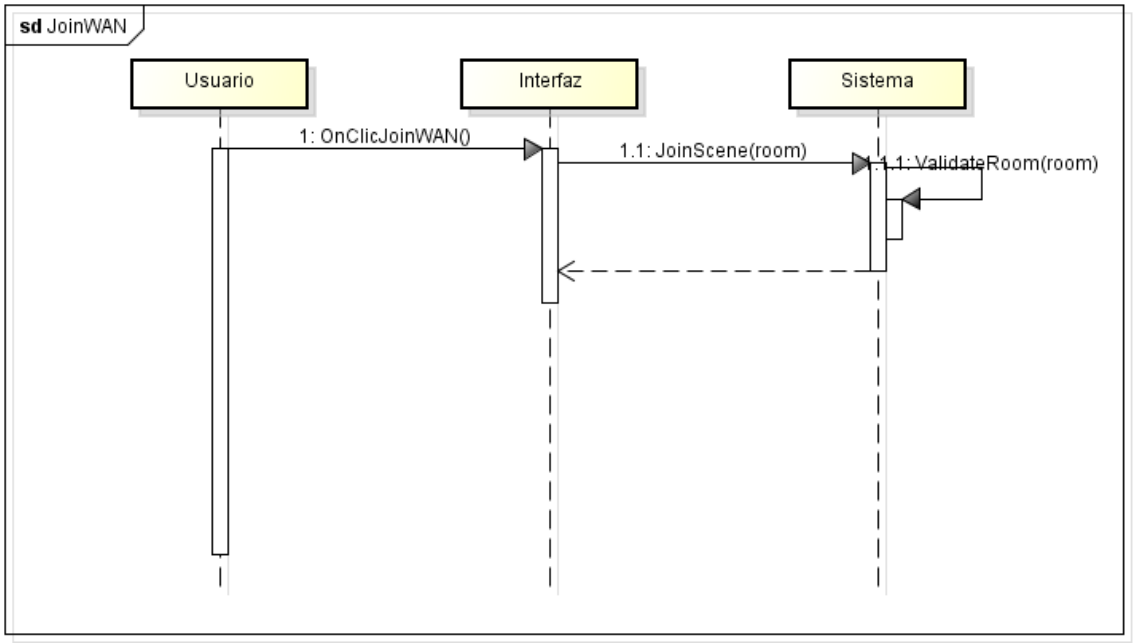


Imagen 16. Diagrama de secuencia de UC-004

UC-005	**Salir del juego**
Versión	1.0
Actor	Usuario
Descripción	El actor podrá cerrar la aplicación en cualquier momento.
Precondición	Ninguna
Secuencia	1- El actor selecciona el botón "Salir del Juego". 2- El sistema muestra una advertencia para asegurar que el actor realmente desea cerrar la aplicación. 3- El actor acepta.
Postcondición	La aplicación se cierra.
Excepciones	Si el actor no acepta en la advertencia, el sistema mantiene la aplicación abierta. Para cerrar la aplicación es necesario haber abandonado previamente cualquier partida.

Tabla 48. UC-005

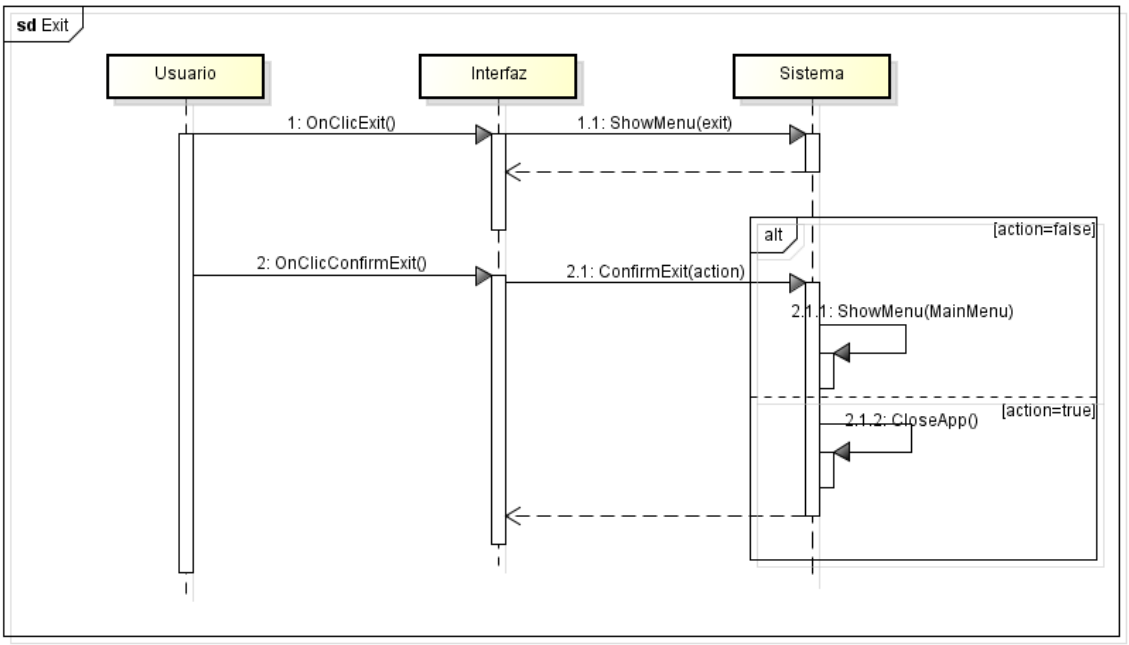


Imagen 17. Diagrama de secuencia de UC-005

4.4.5.3 Casos de uso "scene" Juego

Una vez elegida la modalidad de juego (LAN o WAN) y el mapa, Unity carga una nueva scene en el que el usuario dispone de los siguientes casos de uso:

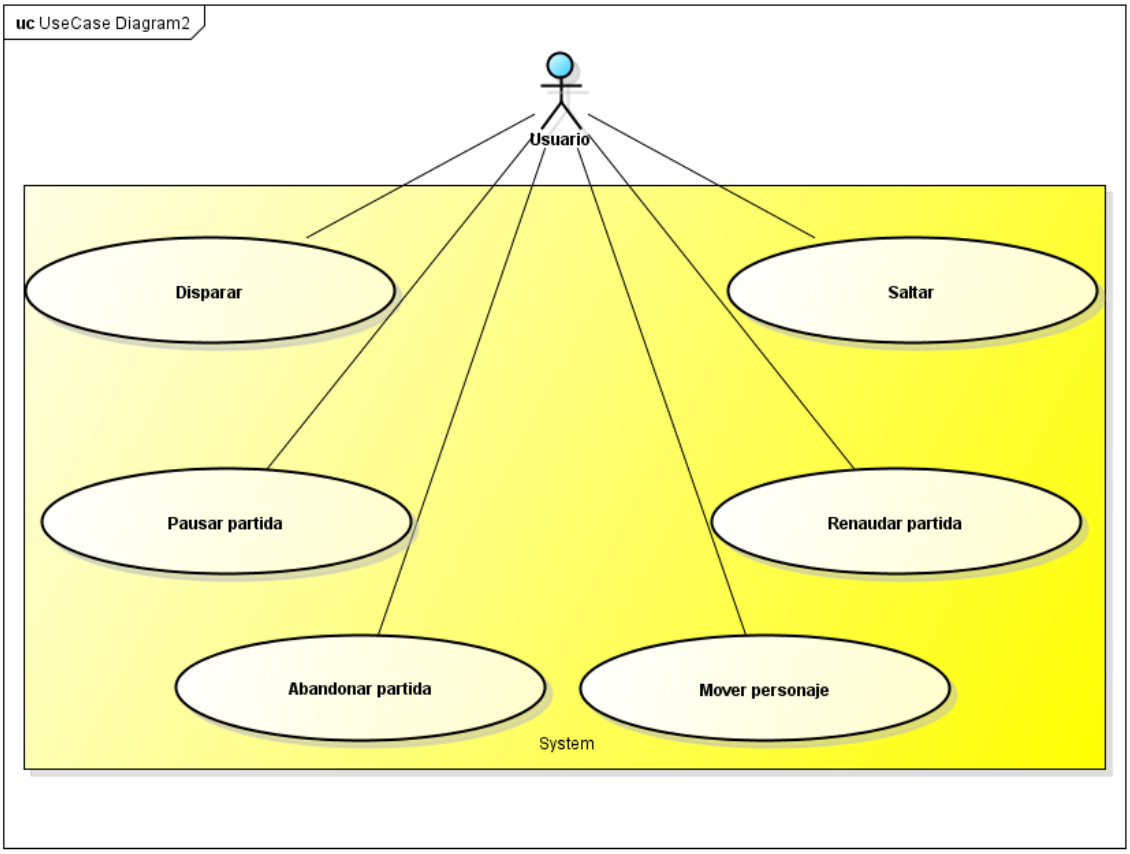


Imagen 18. Casos de uso Diagrama 2

UC-006	**Mover personaje**
Versión	1.0
Actor	Jugador
Descripción	El actor podrá utilizar el teclado del ordenador para mover el personaje de la aplicación por pantalla.
Precondición	El sistema debe encontrarse en la scene de juego.
Secuencia	1- El actor pulsa una de las teclas de movimiento. 2- El sistema reconoce su pulsación.
Postcondición	El personaje se mueve en la dirección correspondiente a la tecla pulsada.
Excepciones	Si el juego está en pausa, no se detectan las teclas de movimiento.

Tabla 49. UC-006

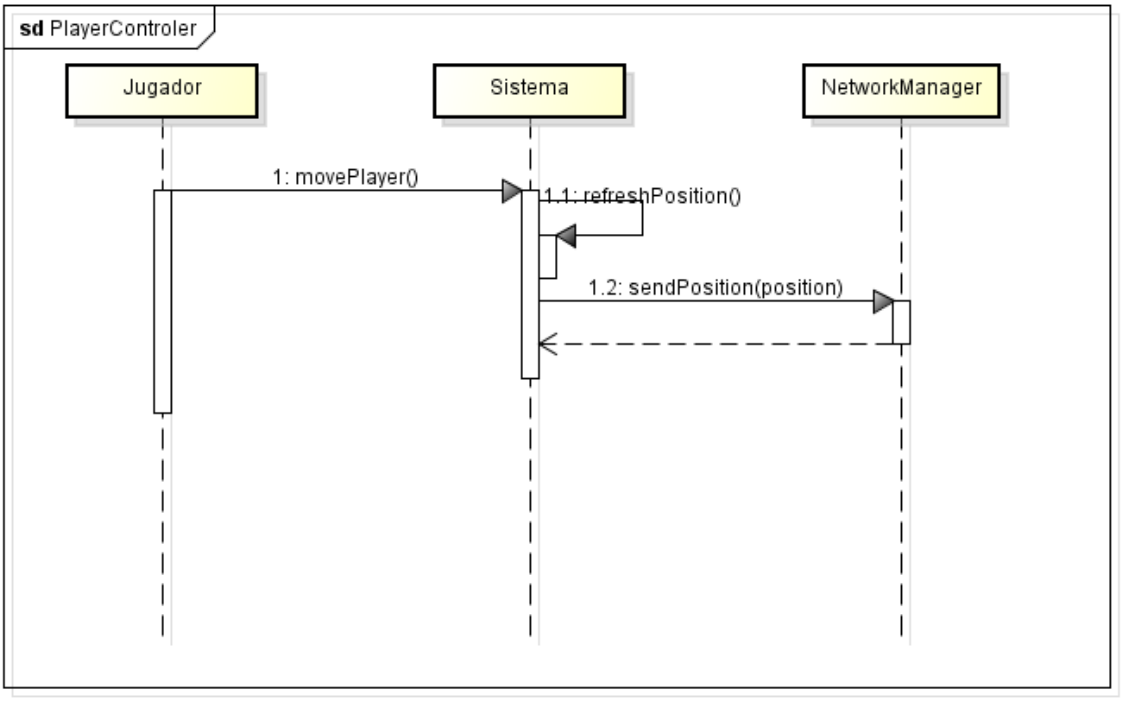


Imagen 19. Diagrama de secuencia de UC-006

UC-007	**Saltar**
Versión	1.0
Actor	Jugador
Descripción	El actor podrá utilizar el teclado del ordenador para realizar saltos con el personaje por pantalla.
Precondición	El sistema debe encontrarse en la scene de juego.
Secuencia	1- El actor pulsa la tecla de salto. 2- El sistema reconoce su pulsación.
Postcondición	El personaje asciende mientras se tenga la tecla de salto pulsada.
Excepciones	El sistema ha detectado que se ha superado el tiempo máximo de vuelo. Si el juego está en pausa, no se detecta la tecla de salto.

Tabla 50. UC-007

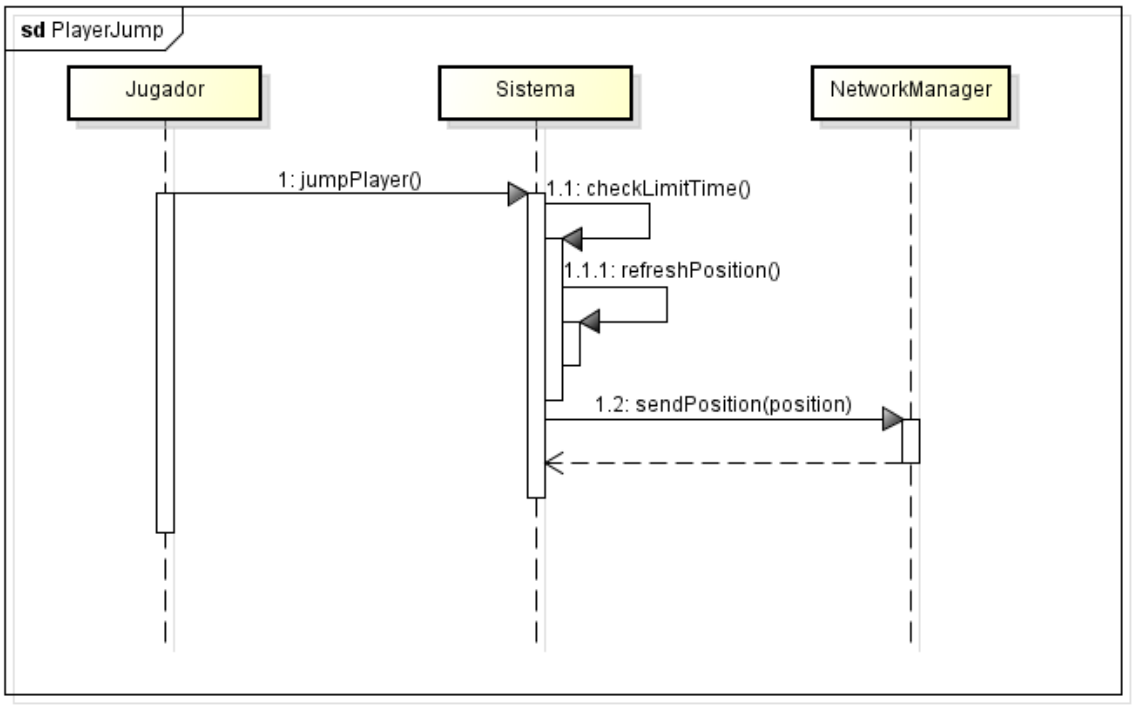


Imagen 20. Diagrama de secuencia de UC-007

UC-008	**Disparar**
Versión	1.0
Actor	Jugador
Descripción	El actor podrá utilizar el teclado del ordenador para realizar disparos con el personaje por pantalla.
Precondición	El sistema debe de encontrarse en la scene de juego.
Secuencia	1- El actor pulsa la tecla de disparo. 2- El sistema reconoce su pulsación.
Postcondición	El personaje dispara en el punto que marque la mirilla del personaje por pantalla.
Excepciones	Si el juego está en pausa, no se detecta la tecla de disparo.

Tabla 51. UC-008

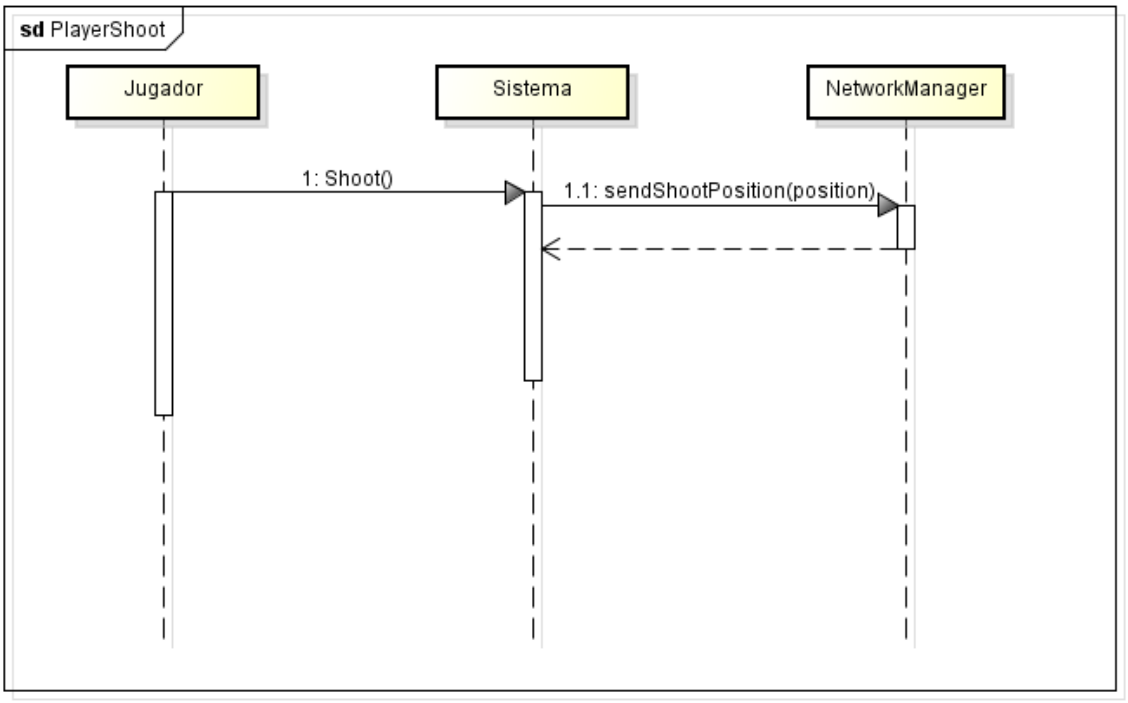


Imagen 21. Diagrama de secuencia de UC-008

UC-009	**Pausar partida**
Versión	1.0
Actor	Jugador
Descripción	El actor podrá pausar el juego en cualquier momento.
Precondición	El sistema debe de encontrarse en la scene de juego.
Secuencia	1- El actor pulsa la tecla de escape. 2- El sistema reconoce su pulsación.
Postcondición	El sistema muestra el menú de pausa
Excepciones	Si el juego ya está en pausa, al pulsar escape, este vuelve a reanudarse.

Tabla 52. UC-009

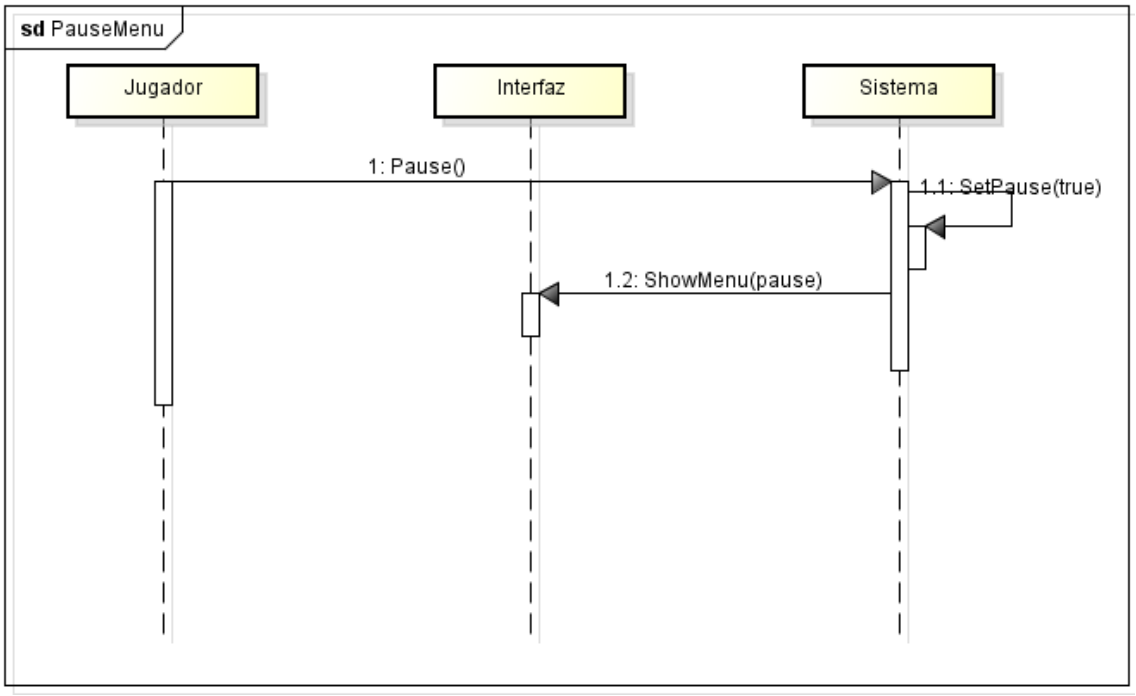


Imagen 22. Diagrama de secuencia de UC-009

UC-010	**Reanudar partida**
Versión	1.0
Actor	Jugador
Descripción	El actor podrá reanudar la partida.
Precondición	El sistema debe de encontrarse en la scene de juego. El sistema debe de encontrarse en el menú de pausa.
Secuencia	1- El actor pulsa la tecla de escape. 2- El sistema reconoce su pulsación.
Postcondición	El sistema reanuda el juego.
Excepciones	Ninguna.

Tabla 53. Tabla UC-010

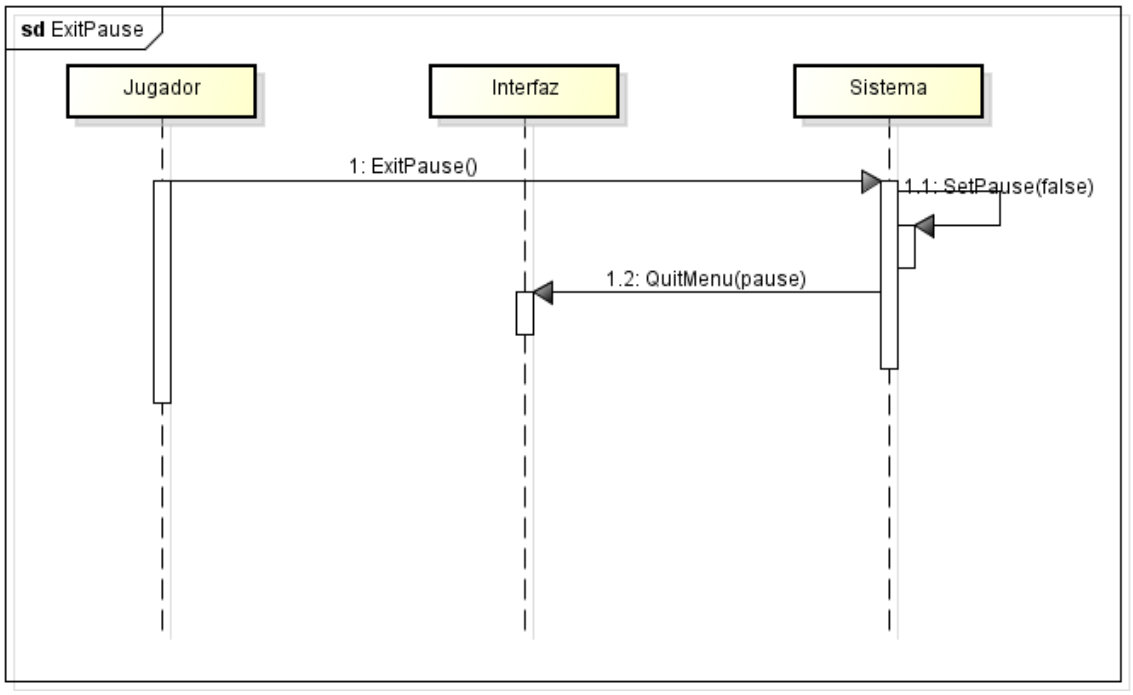


Imagen 23. Diagrama de secuencia de UC-010

UC-011	Abandonar partida
Versión	1.0
Actor	Jugador
Descripción	El actor podrá abandonar la partida.
Precondición	El sistema debe de encontrarse en la scene de juego. El sistema debe de encontrarse en el menú de pausa.
Secuencia	1- El actor pulsa el botón abandonar partida. 2- El sistema reconoce su pulsación.
Postcondición	El sistema abandona la partida volviendo al menú de juego.
Excepciones	Ninguna.

Tabla 54. UC-011

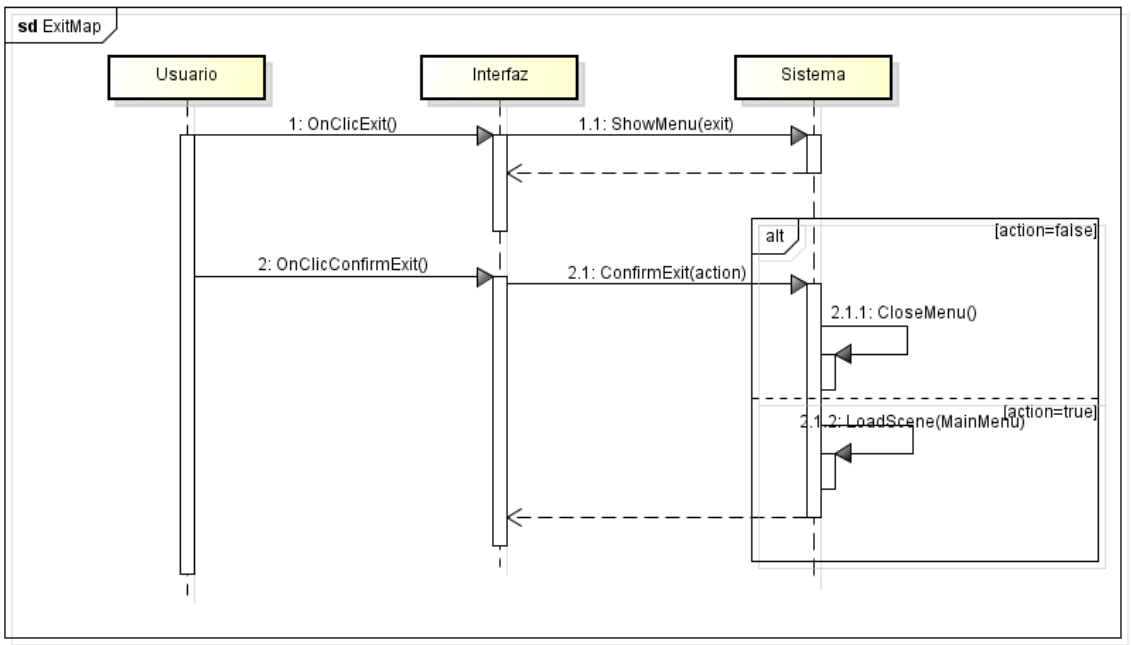


Imagen 24. Diagrama de secuencia de UC-011

4.4.6 Arquitectura lógica

Dado que el desarrollo de esta aplicación se realiza a partir del motor de desarrollo de Unity 3D, la arquitectura lógica de desarrollo viene marcada por este framework que hemos elegido como herramienta de desarrollo, siendo en este caso una arquitectura basada en el patrón MVC (modelo vista controlador).

El diagrama [12] correspondiente a la arquitectura lógica de la aplicación es por tanto el siguiente:

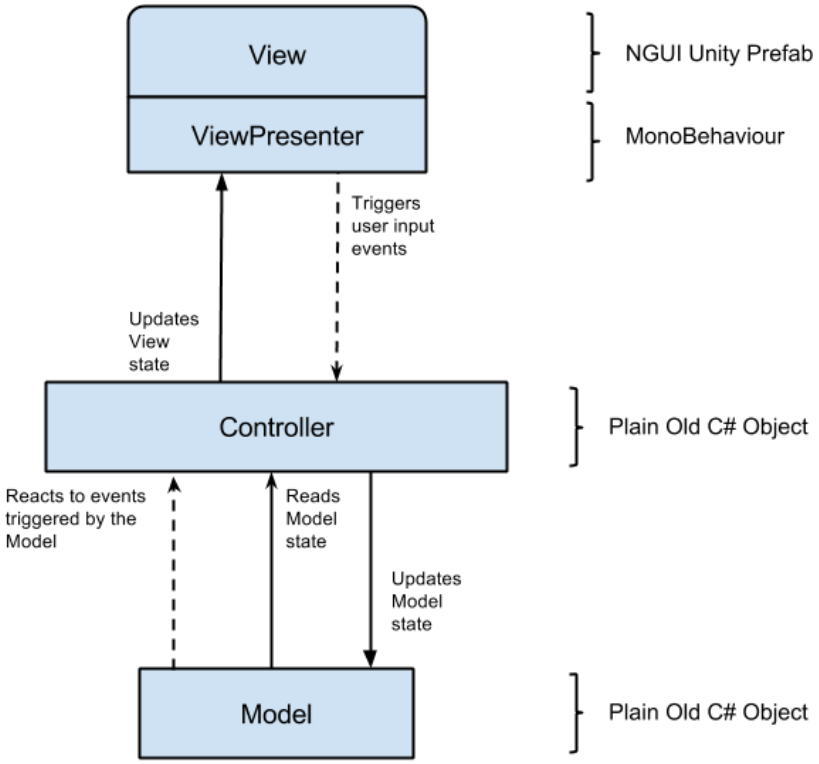


Imagen 25. Diagrama de la arquitectura lógica de Unity 3D

4.4.7 Arquitectura física del sistema

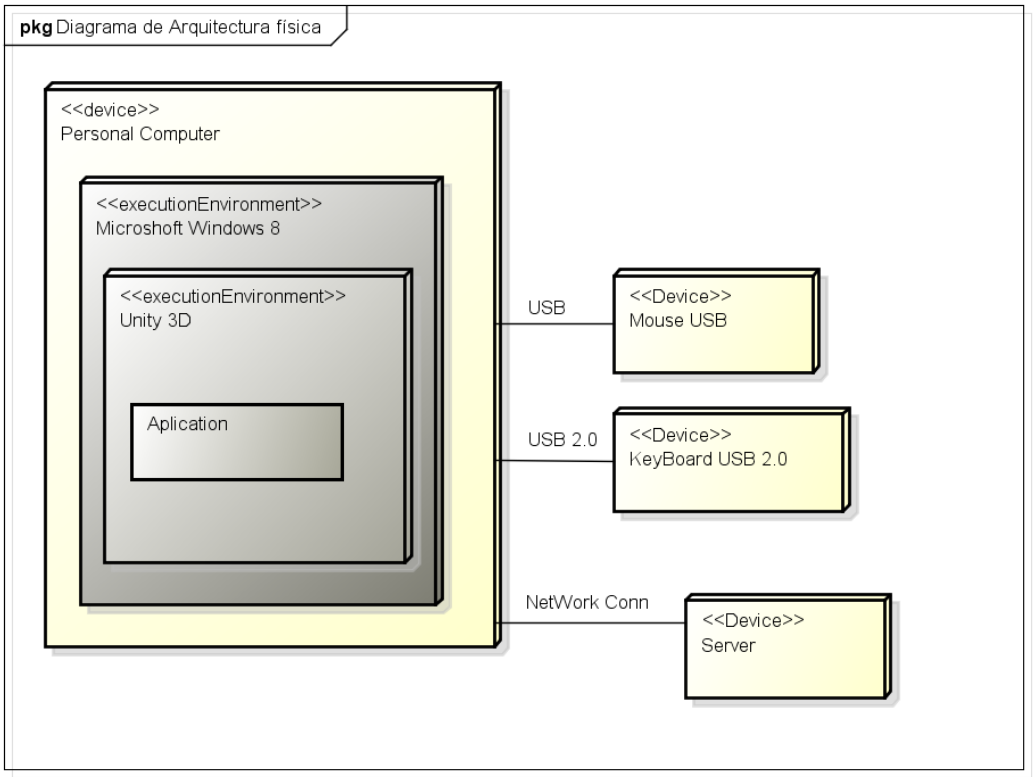


Imagen 26. Diagrama de la arquitectura física

5 Plan de Iteraciones

En el desarrollo de este proyecto se han llevado a cabo se han realizado un total de tres iteraciones las cuales se explican a continuación.

5.1 Primera Iteración

En esta primera iteración se establecen el motor básico del que dispondrá la aplicación, estableciéndose lo siguientes puntos clave, los cuales se explicará su desarrollo a continuación:

- Creación de un mapa básico y personajes de juego.
- Movimiento del personaje
- UNET: Sincronización del movimiento de personajes en red
- Volar
- Disparar y recibir daño
- Morir y reaparecer

5.1.1 Diagrama de clases de diseño

El diagrama de clases de diseño correspondiente a esta primera iteración es el siguiente:

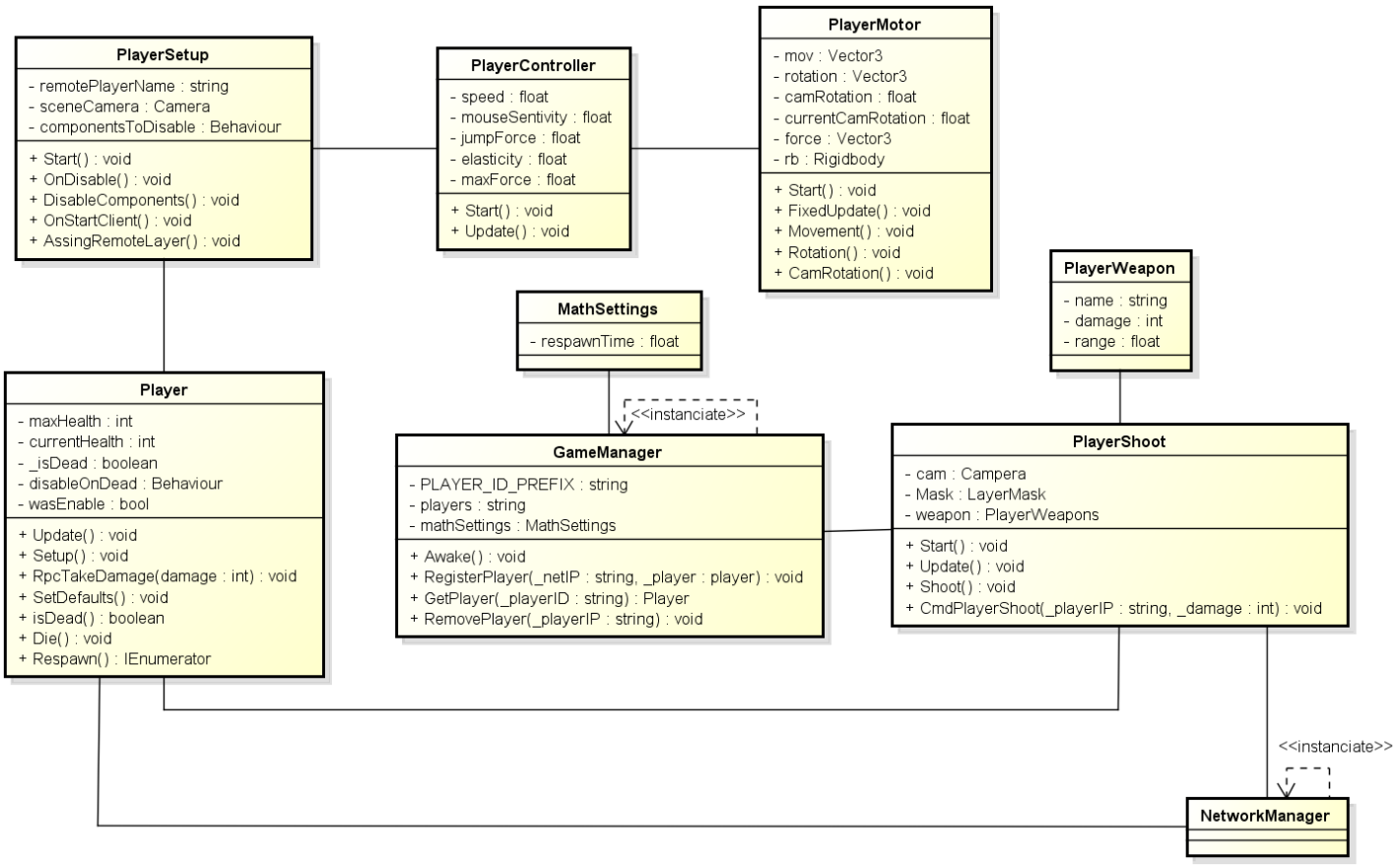


Imagen 27. Diagrama de clases de diseño Iteración 1

5.1.2 Creación de un mapa básico y personajes de juego

El primer punto para poder comenzar a desarrollar es disponer de un mapa por el que se pueda desplazar loar personajes del juego y como no, el propio personaje. Por ahora comenzaremos

con prototipos básicos que no requieran tiempo en su construcción y se puedan realizar rápidamente a través de las herramientas básicas de Unity.

Para ello, importamos los Assets de prototipado [14] para desarrollar el mapa:

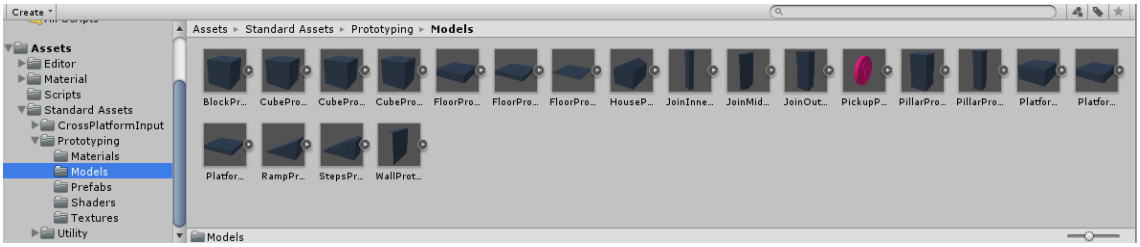


Imagen 28. Standard Assets: Prototyping

Y construimos un mapa como el siguiente:

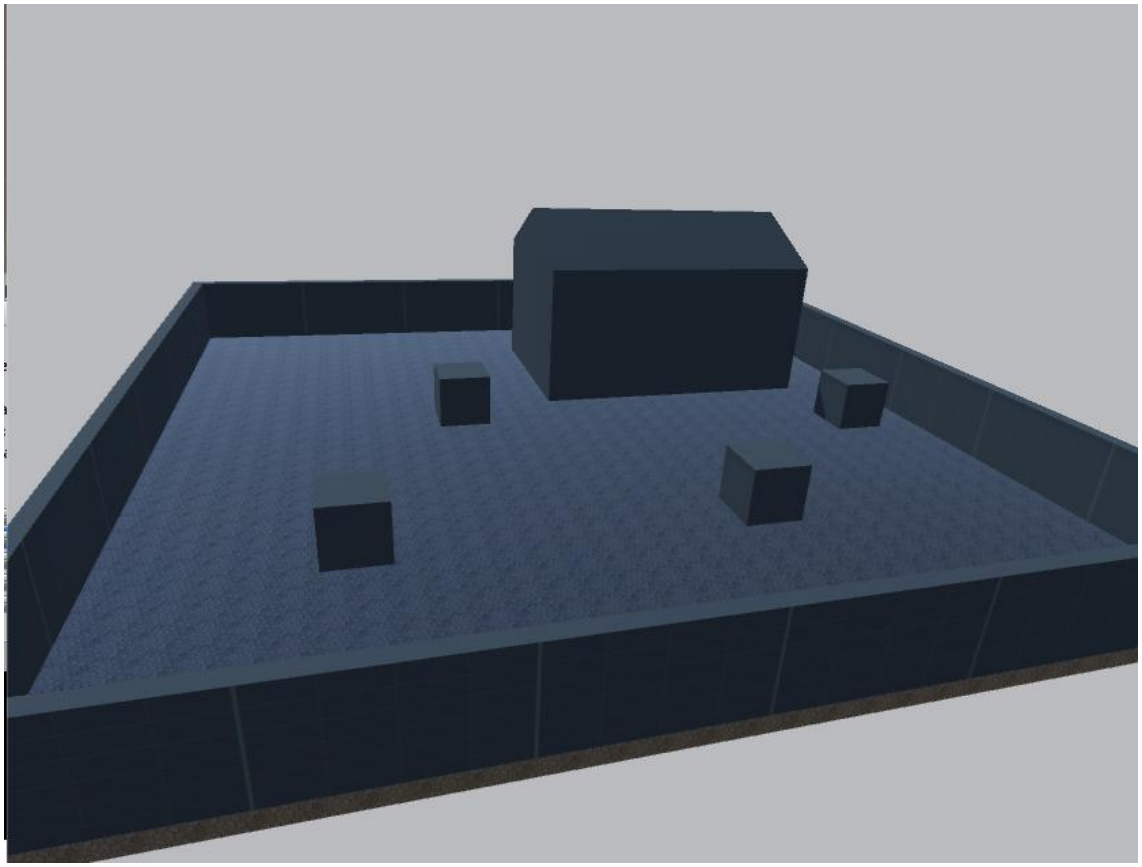


Imagen 29. Mapa básico prototipo

Se agrupan todos los componentes que lo forman dentro de un "Empy game object" al que denominaremos "Environment".

En cuanto al personaje, por el momento, construimos un modelo simple a partir de objetos simples como una esfera, un cubo y dos cilindros, formando así el cuerpo principal del personaje, junto con su arma.

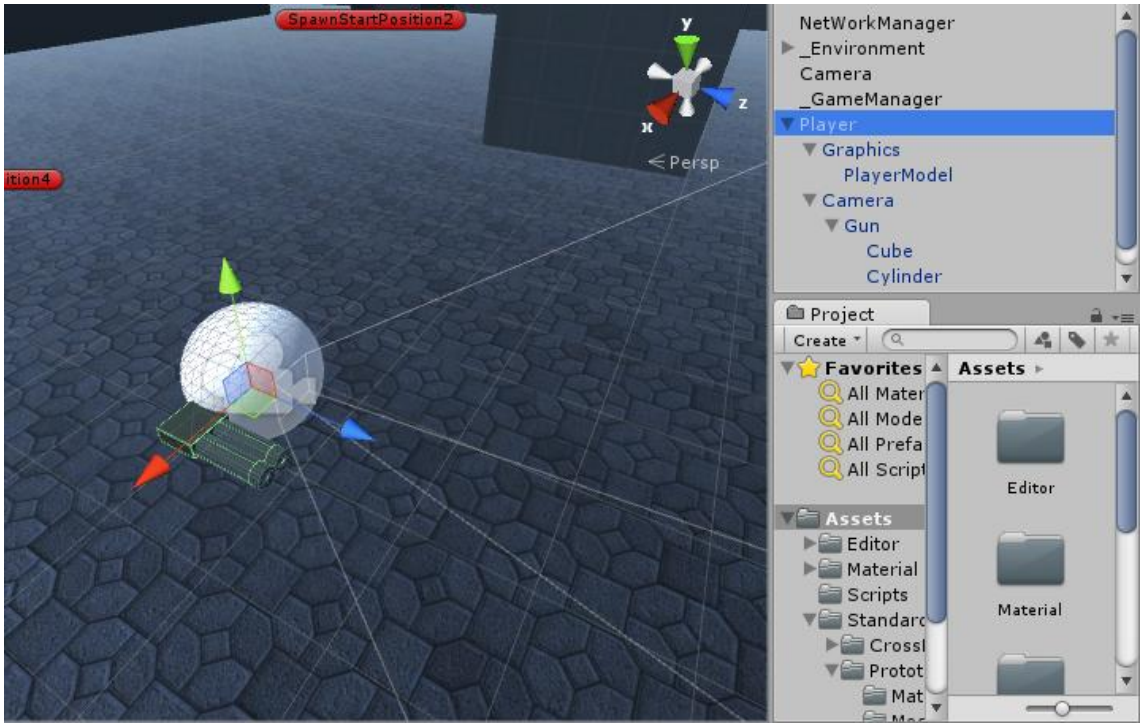


Imagen 30. Personaje prototipo

Para finalizar la construcción del modelo de personaje, le insertamos una cámara de visión y le bloqueamos los movimientos de rotación en el motor de físicas, ya que estos se realizarán a través del ratón del ordenador y deshabilitamos también la gravedad para de esta manera permitir que se mantenga flotando en el espacio.

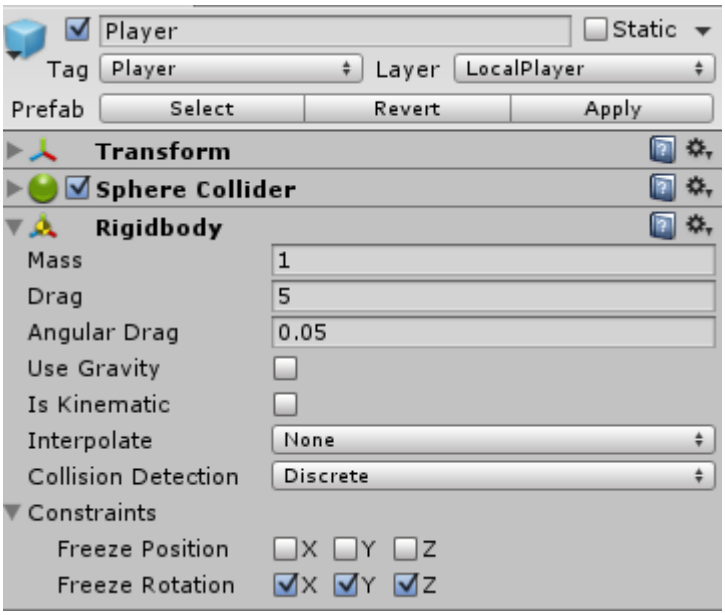


Imagen 31. Player Rigid body: Constraints Rotation

5.1.3 Movimiento del personaje

Llegados a este punto, si compilamos el juego lo único que obtendremos es una vista estática en primera persona sobre la cual no podemos mover ni alterar de ninguna forma posible.

Para solucionar esto, el siguiente paso, es proveer al personaje de movimiento a través de scripts. Estos scripts, los aplicaremos sobre el objeto al que hemos denominado "Player" y se encargarán en un principio de dos funciones básicas, por un salo, comprobar las pulsaciones del teclado y movimientos del ratón, y por otro transformar estas pulsaciones en el movimiento del objeto "Player" y de la rotación de su cámara.

```
void Update () {
Vector3 movX = transform.right*Input.GetAxisRaw("Horizontal");
Vector3 movZ = transform.forward*Input.GetAxisRaw("Vertical");
Vector3 mov = (movX + movZ).normalized * speed;

motor.Move(mov);

Vector3 rotation = new Vector3(0f, Input.GetAxisRaw("Mouse X"), 0f) *
mouseSensitivity;
float camRotation = Input.GetAxisRaw("Mouse Y")*(-1) * mouseSensitivity;

 motor.Rotate(rotation);
 motor.CamRotate(camRotation);
}
```

Tabla 55. Captura de pulsaciones de teclado

Como es evidente existen infinidad de maneras de capturar las pulsaciones de teclado. El método utilizado en este proyecto se realiza a través de la función "Inpu.GetAxisRaw" que recoge las pulsaciones de las teclas a las cuales Unity tiene asignado un nombre por defecto (el cual se puede personalizar y editar en cualquier momento).

En este caso "Horizontal" corresponde a las teclas "A" y "D" para las cuales en caso de ser la "D", se le asigna el valor positivo "1" y "-1" para la letra "A". Lo mismo ocurre con "Vertical" que utiliza las letras "W" y "S" (que como se puede observar corresponden a los controles de movimiento típicos de todo juego FPS) y los ejes del ratón que se encuentran bajo el nombre de "Mouse X" y "Mouse Y".

```
if (mov != Vector3.zero)
        {
            rb.MovePosition(rb.position + mov * Time.fixedDeltaTime);
        }
rb.MoveRotation(rb.rotation * Quaternion.Euler(rotation));

currentCamRotation += camRotation;
        currentCamRotation = Mathf.Clamp(currentCamRotation, -camRotationLimit,
camRotationLimit);
        cam.transform.localEulerAngles = new Vector3(currentCamRotation, 0f,
0f);
```

Tabla 56. Movimiento del personaje

En el caso del movimiento de los ejes "Z" y "X" ("horizontal" y "vertical"), estos, incrementamos o decrementamos su valor cada vez que su valor sea distinto de cero en función del tiempo y en el caso del movimiento de la cámara a través del ratón, es decir, los ejes "Y" y "X", estos a diferencia del anterior, en vez de realizarlo a través de posiciones relativas, lo realizamos mediante posición absoluta ya que la cámara dentro del personaje se mantiene fija en la misma posición en ese espacio relativo y para asegurarnos de que el eje "Y" no termine boca abajo, limitamos el número de grados en el espacio que es capaz de rotar.

5.1.4 UNET: Sincronización del movimiento de personajes en red

Establecer una función multijugador online no es tarea fácil, afortunadamente Unity nos provee un módulo de administración de red (UNET: Unity Networking) con el que podemos establecer las opciones online de juego sin tener que crear protocolos de conexión ni bases de datos.

Para ello, creamos un "empy object" en la scene donde tenemos el mapa y le añadimos el componente "networt manager" y dado que no tenemos aún ninguna interfaz para poder configurar las opciones online, añadimos también el componente "network manager HUD". De esta manera, al ejecutar el juego, nos aparecerán las opciones de conexión en LAN de crear y unirse a una partida.

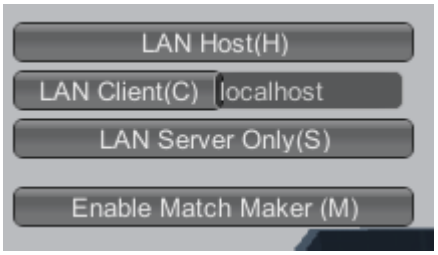


Imagen 32. Network Manager HUD

Realizado únicamente esto, solo podemos conectarnos online a la sala pero no podemos observar los movimientos de los jugadores "enemigos". Para ello es necesario, por un lado, indicar en el mapa cuales son los puntos de inicio donde aparecerán los jugadores y cual el "objeto" jugador. Y por otro indicar que objetos necesitan que sus posiciones en el espacio sean actualizadas y compartidas en red.

Por tanto, en primer lugar creamos los puntos de inicio a partir de "empy objects" y los situamos en diferentes ubicaciones del mapa y les incluimos además el componente "network start position"

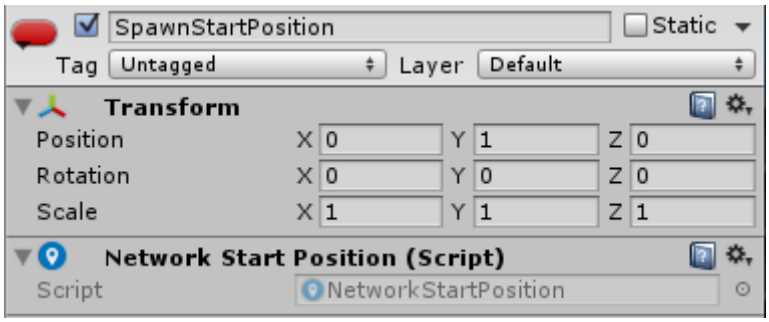


Imagen 33. Network Start Position

El siguiente punto es incluir al objeto "Player" los componentes "Network Identity" para asignarle un identificador único a cada jugador que se conecte y "Network transform" para indicar la posición de este objeto al resto de jugadores.

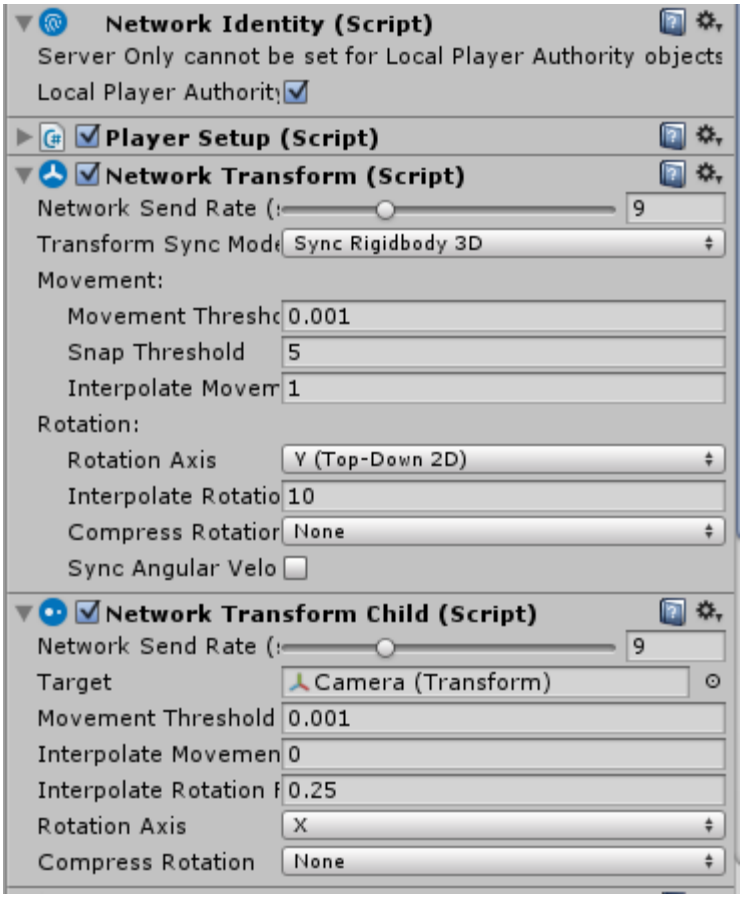


Imagen 34. Network Transform y Network Identity

(Es necesario configurar los valores de interpolación y tiempo de refresco según las necesidades del juego, ya que estas marcan la diferencia entre que un personaje se mueva a "trompicones" o con lag, a saturar la comunicación entre cliente y servidor de un gran volumen de paquetes.)

Finalmente convertimos el objeto "Player" en un prefab y lo eliminamos del mapa, Desde el network manager establecemos el objeto que será el jugador junto con el método de rotación de los puntos de inicio.

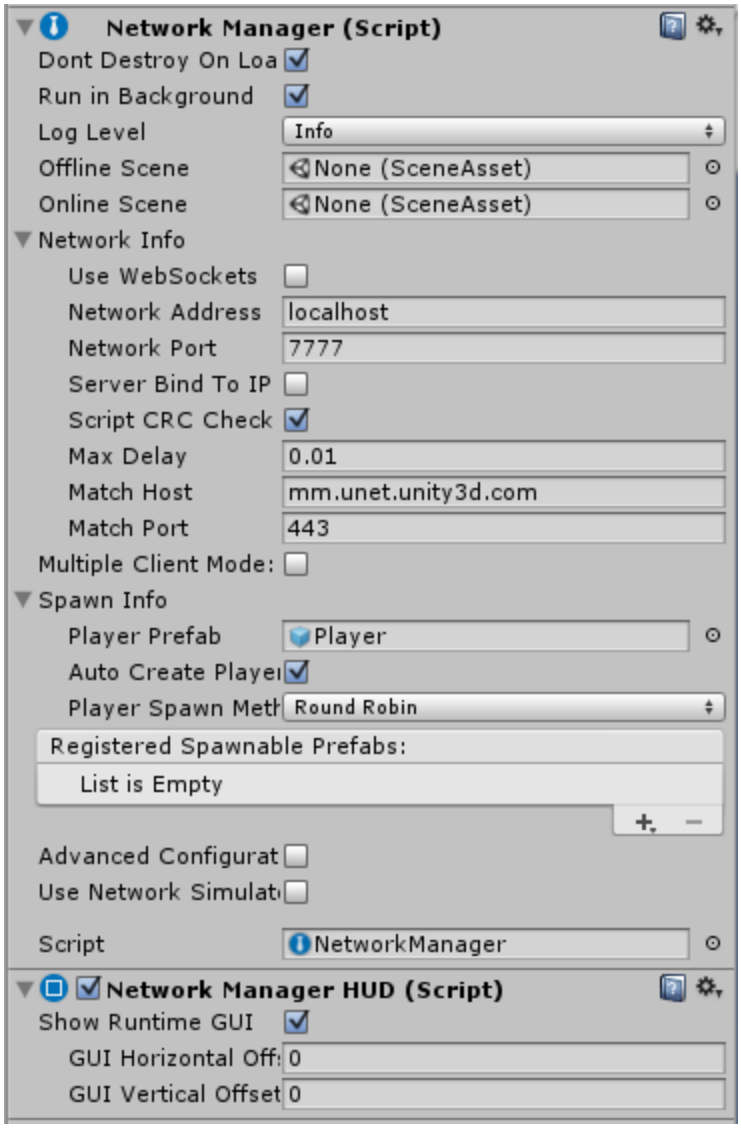


Imagen 35. Configuración Network Manager

Realizado esto, al compilar el juego podemos crear una partida desde el menú de Network Manager HUD y que otros jugadores se unan. Pero sigue existiendo un inconveniente en el momento que realizamos un movimiento, el resto de jugadores se mueven igual, es decir, la aplicación es incapaz de distinguir que controles pertenecen al jugador local y cuales a los de red por lo que cualquier movimiento efectuado por uno de los jugadores se realiza en todos. Para solucionarlo, basta con añadir la comprobación condicional de "islocal" para descartar aquellas instrucciones que no sor realizadas en el cliente local.

5.1.5 Volar

Dado que se ha establecido que uno de los personajes del juego sea un Dron Volador, es imprescindible que vuele y se mantenga suspendido en el aire. Para ello, es necesario la realización de dos puntos, el primero, que al acercarse el Dron al suelo o aterrice este tenga un efecto tipo "colchón" o "muelle" para dar una impresión de mayor realismo y suavizado en sus movimientos aéreos y el segundo punto, el desarrollo de un script que cuando se pulse la tecla correspondiente, el Dron se eleve en el aire.

Para el primer punto, es necesario añadir al objeto "Player" el componente "configurable join" y establecer una fuerza de repulsión y elasticidad en el eje "Y" para que de esta manera el objeto rebote como si se tratase de una colchoneta o muelle.

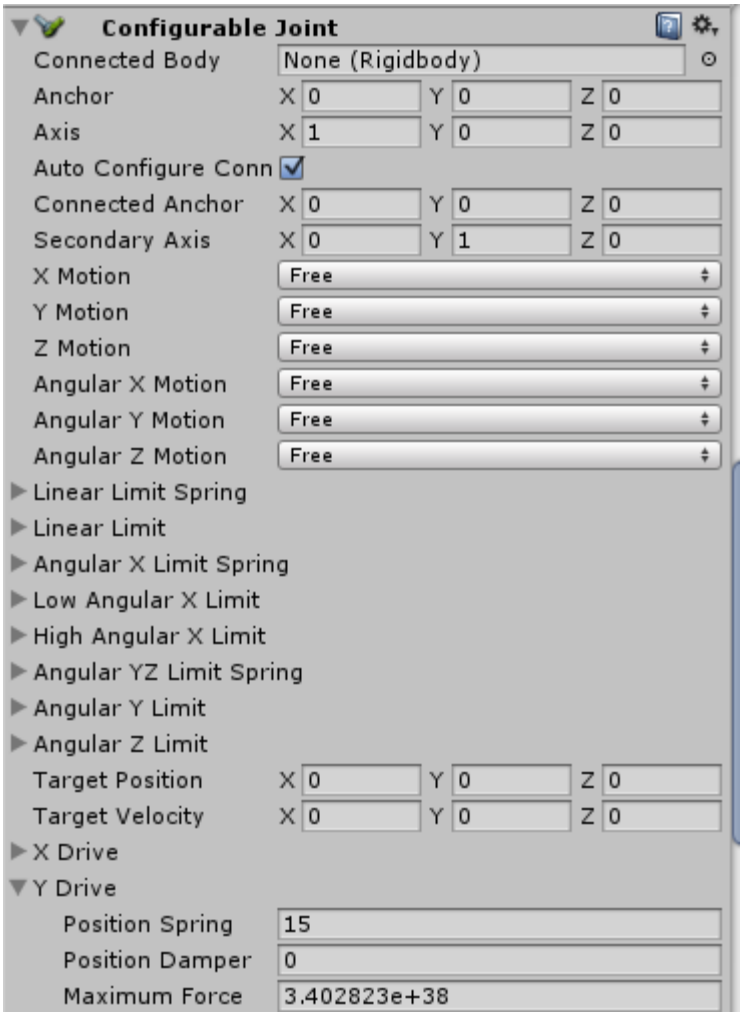


Imagen 36. Configurable Join

Y para el segundo punto, creamos un script que nos permita ascender mediante una fuerza de aceleración en el eje "Y" y ascendente basada en el motor de físicas.

```
if (Input.GetButton("Jump"))
        {
            force = Vector3.up * jumpForce;
            SetJointSettings(0f);
        }
        else
        {
            SetJointSettings(elasticity);
        }
        motor.jumper(force);
```

Tabla 57. Jump script

Para ello creamos por una parte el control que detecte la tecla de salto y además cambie temporalmente los valores del "Configurable Join".

```
if (force != Vector3.zero)
        {
            rb.AddForce(force* Time.fixedDeltaTime,ForceMode.Acceleration);
        }
```

Tabla 58. Vertical acceleration

Y por otro lado, que cada vez que se detecte la aplicación de una "fuerza" en el eje "Y", aplicar una aceleración de desplazamiento vertical ignorando la masa del objeto.

5.1.6 Disparar y recibir daño

Para "disparar", se puede realizar de múltiples formas como por ejemplo creando un objeto bala y que al impactar sobre un jugador este reciba daño, o como en nuestro caso, a través del uso de un "raycast" que consiste en lanzar un rayo invisible en línea recta desde un punto concreto de origen y por medio de la función "raycastHit", comprobar si este ha colisionado sobre un objeto.

De esta manera podemos realizar los mismos efectos que con el uso de una "bala" real sin tener que consumir grandes recursos de procesamiento en el cálculo de las físicas de disparo.

```
void Update()
    {
        if (Input.GetButtonDown("Fire1"))
        {
            Shoot();
        }
    }
    [Client]
    void Shoot()
    {
        RaycastHit _hit;
        if (Physics.Raycast(cam.transform.position, cam.transform.forward, out
_hit, weapon.range, mask))
        {
            if (_hit.collider.tag == PLAYER_TAG)
            {
                CmdPlayerShoot(_hit.collider.name, weapon.damage);
            }
        }
    }
    [Command]
    void CmdPlayerShoot(string _playerID, int _damage)
    {
        Debug.Log("Ha sido disparado:" + _playerID);
        Player _player = GameManager.GetPlayer(_playerID);
        _player.RpcTakeDamage(_damage);
    }
```

Tabla 59. Player Shoot

Creamos una función que solo se ejecute en los clientes (ya que no deseamos que realice los cálculos de físicas el servidor) que consista en que al pulsar el botón de disparo se proyecte un "raycast" en función de la posición a la que esté orientada la cámara y con un cierto rango válido que más adelante variará en función del arma seleccionada. Además es necesario indicar el objeto con el que ha colisionado para así, en caso de ser un jugador, descontarle la vida correspondiente.

```
[ClientRpc]
    public void RpcTakeDamage(int damage)
    {
        if (isDead)
            return;
        currentHealth -= damage;
        Debug.Log(transform.name + "- Vida restante: " + currentHealth);
        if (currentHealth <= 0)
            Die();
    }
```

Tabla 60. TakeDamage

Añadimos también una función que se ejecute únicamente en el cliente del jugador que ha recibido el daño y le reste cierta cantidad de vida sobre la vida total que posee.

5.1.7 Morir y reaparecer

Llegados a este punto, se ha conseguido un mapa en que poseemos una "esfera" suspendida en el aire que puede volar con un arma capaz de descontar vida a objetos de cierto tipo y además se pueden conectar otros jugadores para pelear entre ellos. ¿Pero qué pasa cuando la vida de un jugador llega a cero? Nada

Por eso el siguiente punto esencial en un FPS online es que lo jugadores puedan morir y reaparecer al cabo de un momento en algún punto del mapa de nuevo.

```
    public void SetDefaults()
    {
        isDead = false;
        currentHealth = maxHealth;
        for(int i = 0; i <disableOnDeath.Length; i++)
        {
            disableOnDeath[i].enabled = wasEnable[i];
        }

        Collider _col = GetComponent<Collider>();
        if (_col != null)
            _col.enabled = true;
    }

    public bool isDead
    {
        get { return _isdead; }
        protected set { _isdead = value; }
    }

    private void Die()
    {
        isDead = true;
        for (int i = 0; i < disableOnDeath.Length; i++)
        {
            disableOnDeath[i].enabled = false;
        }
        Collider _col = GetComponent<Collider>();
        if (_col != null)
            _col.enabled = false;
        Debug.Log(transform.name + " ha muerto");
        StartCoroutine(Respawn());
    }
```

Tabla 61. IsDead

Creamos un script que se encargue de que en el momento en el que la vida de un jugador llegue a cero y sea declarado como "muerto", se deshabiliten las funcionalidades de disparar y controlar al personaje.

```
private IEnumerator Respawn()
    {
        yield return new
WaitForSeconds(GameManager.instance.matchSettings.respawnTime);

        SetDefaults();
        Transform _spawnPoint = NetworkManager.singleton.GetStartPosition();
        transform.position = _spawnPoint.position;
        transform.rotation = _spawnPoint.rotation;
        Debug.Log(transform.name + ": Respawn");
    }
```

Tabla 62. Respawn

Una vez muerto, hacemos que vuelva a reaparecer al cabo de algunos segundos en alguna otra posición determinada por el "networkManager" y habilitamos los controles del personaje, junto con la puesta de los valores de vida iniciales.

5.2 Segunda Iteración

Desarrolladas las funciones básicas y más esenciales de un motor FPS, en esta segunda iteración se ha procedido a mejorar la apariencia del juego modelando y aplicando texturas al Dron de los jugadores, junto con la aplicación de animaciones de movimiento, imágenes y efectos visuales. Por tanto los puntos a desarrollar en esta iteración son los siguientes:

- Modelo 3D del jugador y animaciones de movimiento
- Mirilla
- Capas de vista para la cámara
- Efectos de disparo y explosiones
- Limitación tiempo de vuelo
- Bloqueo del Ratón
- Flotar sobre cualquier superficie

5.2.1 Diagrama de clases de diseño

El diagrama de clases de diseño correspondiente a esta segunda iteración es el siguiente:

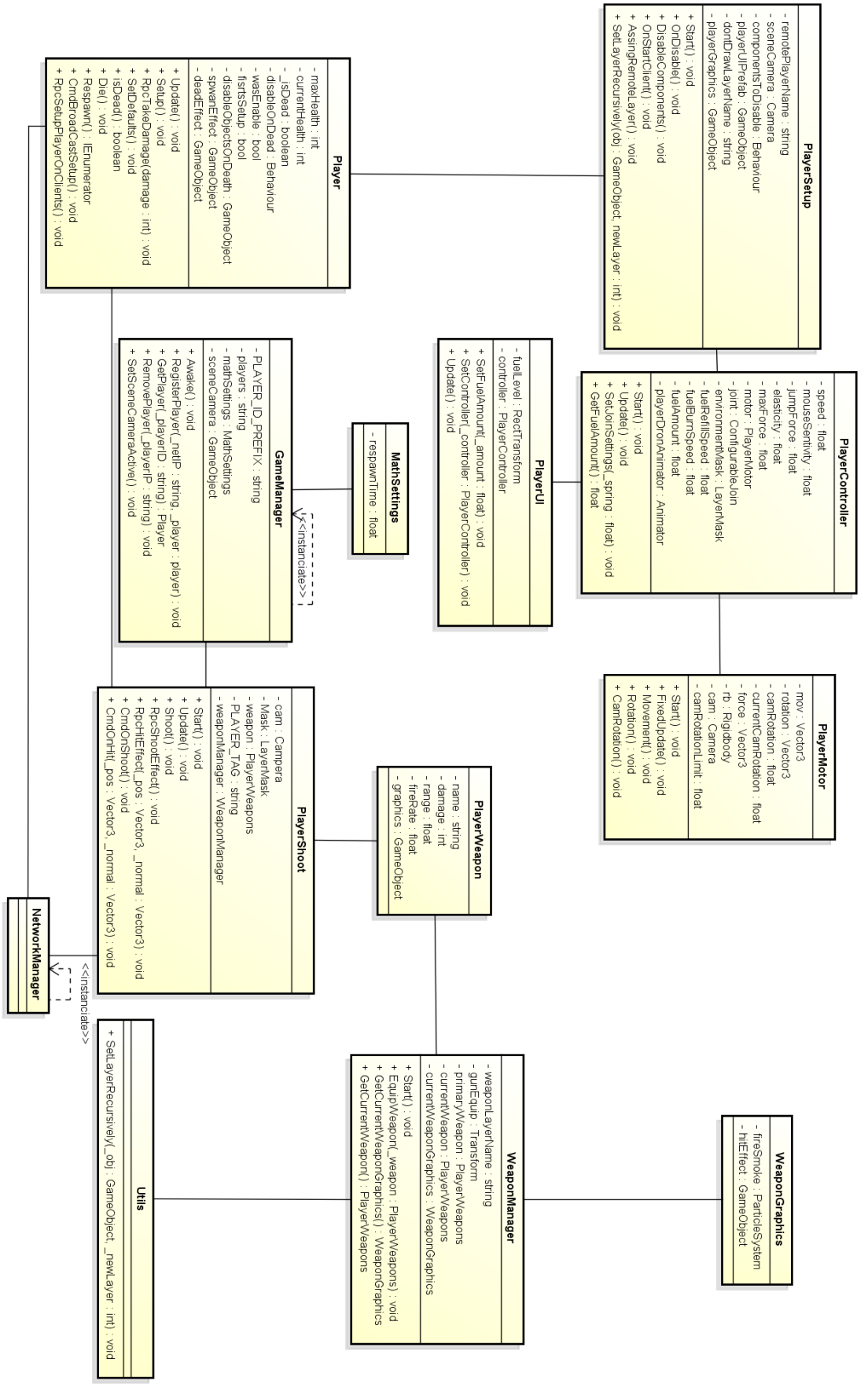


Imagen 37. Diagrama de clases de diseño Iteración 2

5.2.2 Modelo 3D del jugador y animaciones de movimiento

A través de la aplicación de modelado de Cinema 4D, se ha construido el Dron que será nuestro nuevo Mpersonaje.

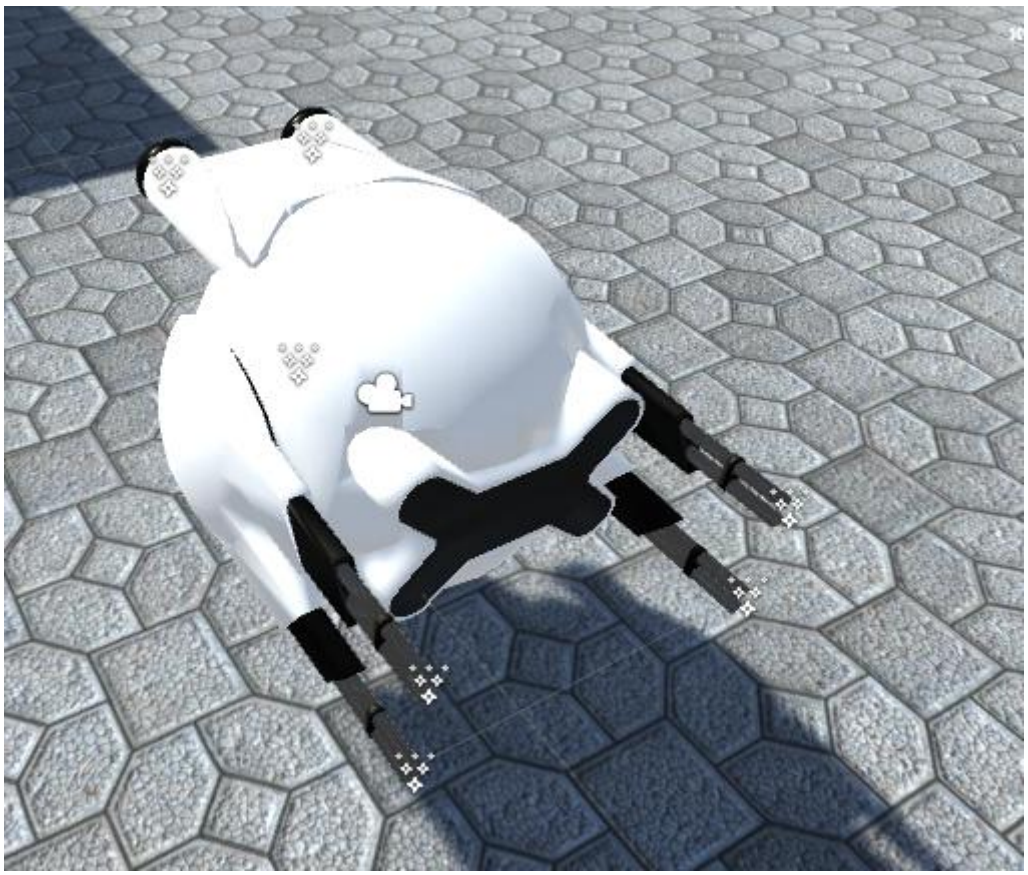


Imagen 38. Modelo 3D Dron

Para sustituir el anterior modelo, solo hemos de importar este nuevo objeto desde Unity y sustituir la esfera y arma que se había desarrollado como modelo prototipo inicialmente.

A continuación creamos los efectos especiales del propulsor. Para ello importamos los standard assets de Unity e implementamos el prefab "After bunner".

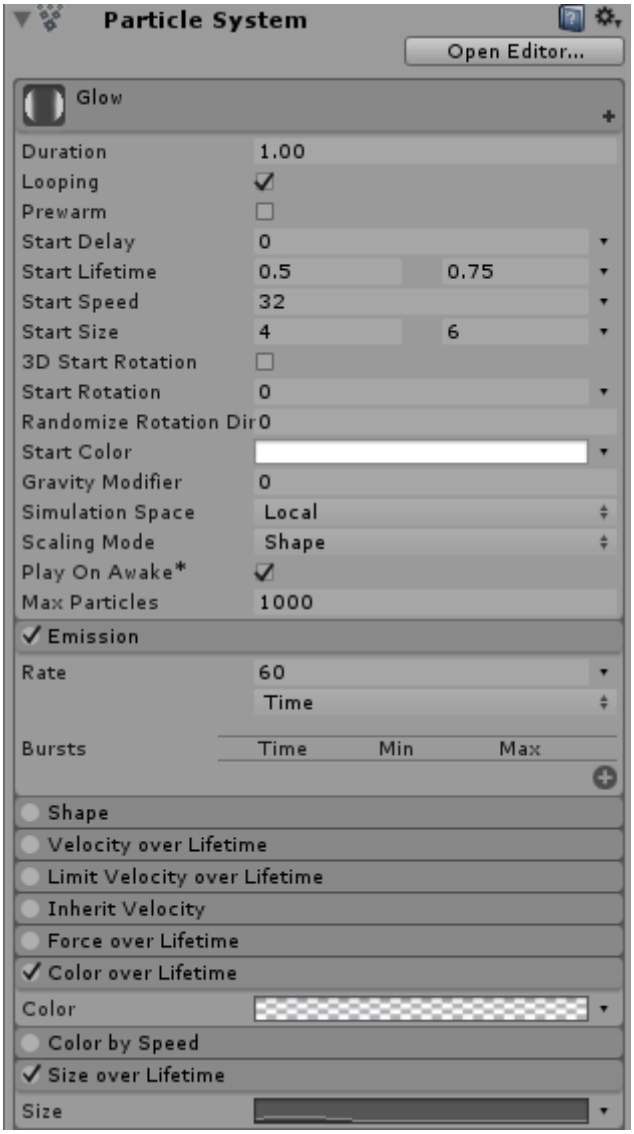


Imagen 39. Particle System

Aprovechando este prefab, borramos el resto de sistemas de partículas dejando únicamente la de "Glow" y modificamos sus parámetros has obtener el efecto deseado.

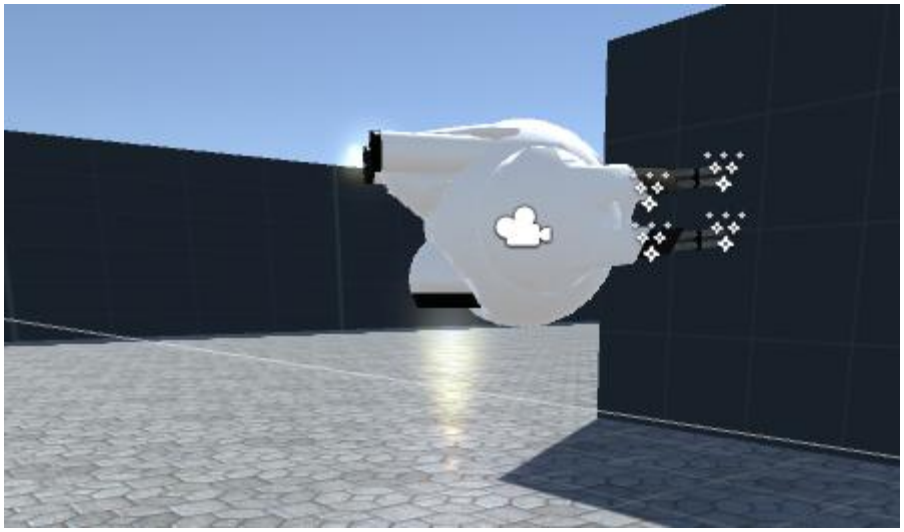


Imagen 40. Dron con "particle system" en los propulsores

El siguiente paso es encargarse de las animaciones del propulsor trasero del Dron. Para ello, haremos uso del animator de Unity, y a través del "Blend tree" establecemos las posiciones iniciales y finales del propulsor, dejando de esta manera a Unity para que calcule las intermedias.

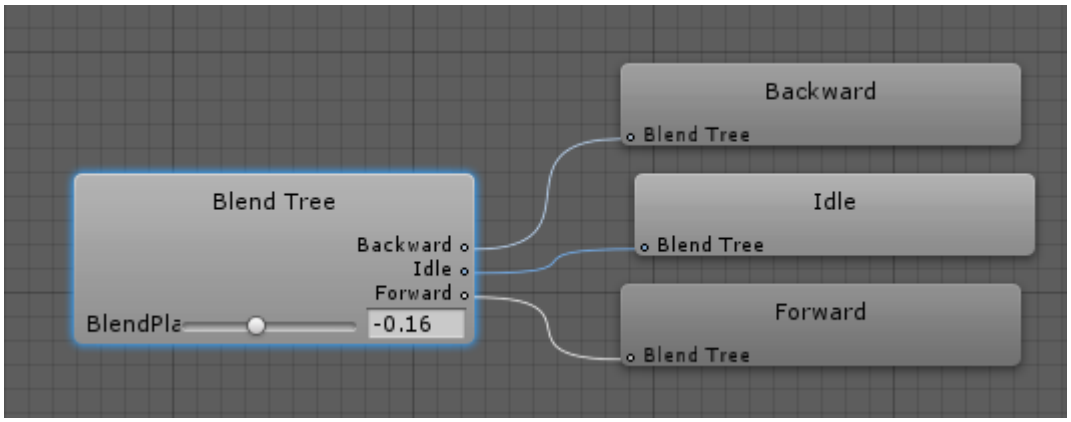


Imagen 41. Cita Blend tree

De esta manera conseguiremos que según el Dron se desplace hacia delante, atrás o este estacionado, la posición del propulsor variará.

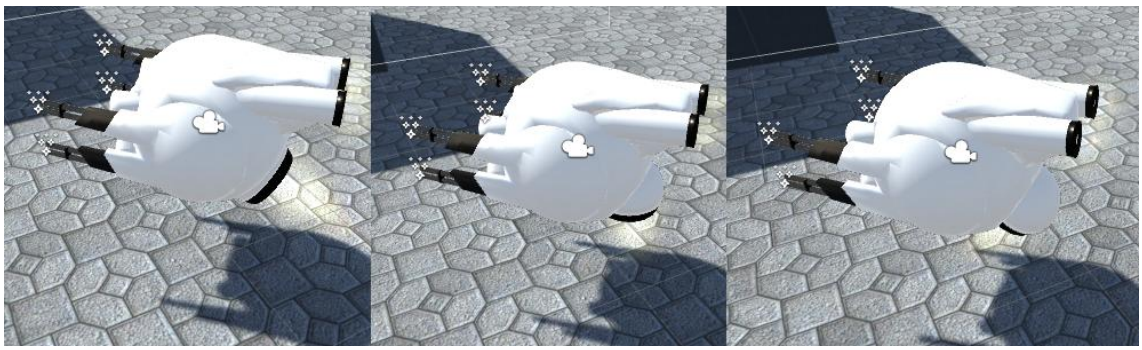


Imagen 42. Animaciones propulsor

Gracias a que hemos utilizado inicicialmente la función "getAxisRaw"para la captura de los movimientos horizontales, esta, a diferencia de otras funciones de captura no da los valores "1" y "-1" directamente al momento de pulsar las teclas correspondientes, sino que dentro de la unidad de tiempo de un segundo, establece los valores intermedios de manera incremental/decremental a partir del cero. Gracias a esto, nos simplifica enormemente el cálculo de las posiciones intermedias de la animación del propulsor, ya que lo que lo único que te tenemos que hacer, es almacenar dichos valores en una variable y aplicarlos sobre la animación la cual establece las tres posiciones del propulsor según los valores intermedios desde "-1" a "1".

```
playerDronAnimator.SetFloat("BlendPlayer", Z);
```

Tabla 63. Código animación propulsor

5.2.3 Mirilla

Como en todo juego de tipo FPS, es clave el tener un punto de mira para saber dónde estamos apuntando. Por eso, a través de photoshop se ha creado una cruceta, la cual colocaremos entre la cámara y el campo de visión del jugador.

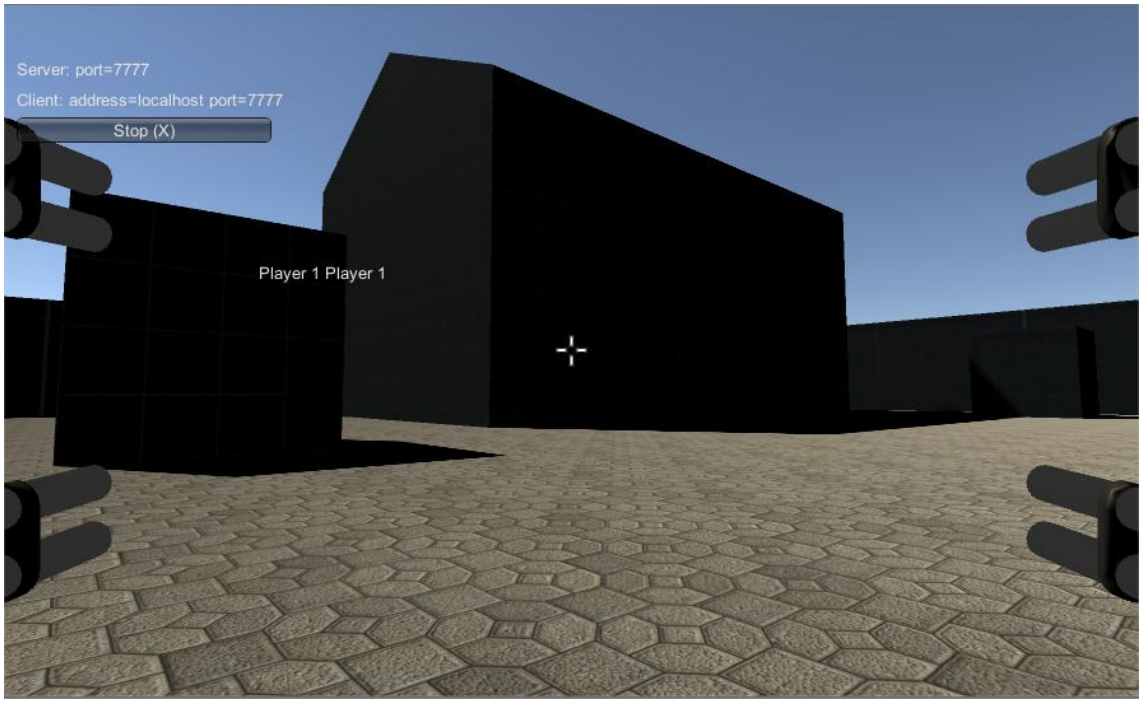


Imagen 43. Punto de mira

La colocación de esta cruceta se realiza por medio de un "Canvas" situado constantemente enfrente de la cámara.

Para evitar que este siempre visible, es necesario la creación de un script que habilite y deshabilite la visualización del Canvas según la situación o menú en el que se encuentre el jugador.

```
void Start () {
        if (!isLocalPlayer)
        {
            DisableComponents();
            AssingRemoteLayer();
        }
        else
        {
            playerUIInstance = Instantiate(playerUIPrefab);
            playerUIInstance.name = playerUIPrefab.name;

            PlayerUI ui = playerUIInstance.GetComponent<PlayerUI>();
            if (ui == null)
                Debug.LogError("No PlayerUI component on PlayerUI prefab.");
        }

       }
```

Tabla 64. GetComponent PlayerUI

De esta manera cargamos el prefab PlayerUI que contiene la cruceta una única vez desde el cliente local en el comienzo del juego así para deshabilitarla, únicamente tenemos que o bien deshabilitarla o destruir el objeto.

5.2.4 Capas de vista de la cámara

Llegados a este punto, podemos observar que las armas, debido a que carecen de un "rigidbody" para calcular las colisiones con otros objetos, estas traspasan cualquier objeto sólido con el que colisionan.

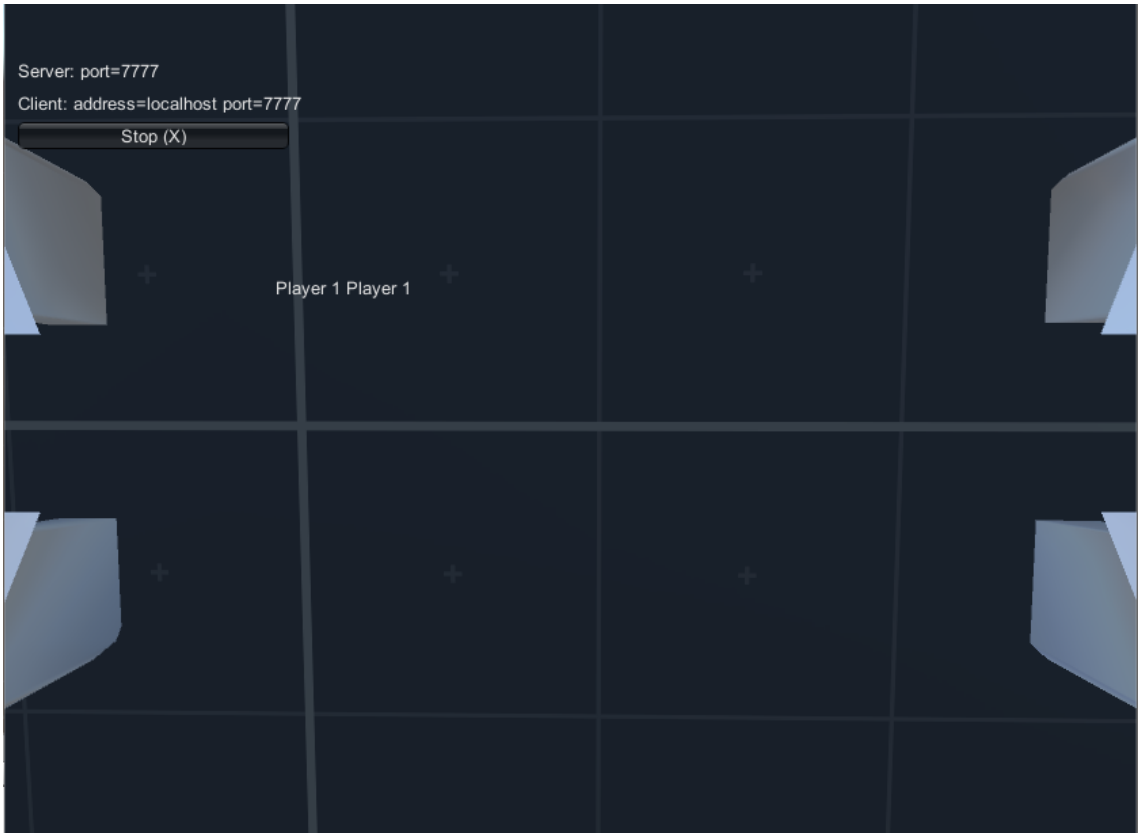


Imagen 44. Colisión Armas

Podríamos fácilmente establecer un sistema de colisiones para eliminar este problema, pero afectaría a la jugabilidad, ya que en situaciones como disparar cerca de una puerta o una esquina nuestro movimiento se vería afectado al chocar las armas con los marcos de las puertas o las paredes.

Por eso, una buena solución, es establecer dos cámaras de visión y superponerlas. Una encargada de mostrar los gráficos del nivel y otra que se encargue de mostrar únicamente las armas.

Imagen 45. Colisión Armas 2

De esta manera, por mucho que las armas se acerquen a una pared, estas no la atravesaran.

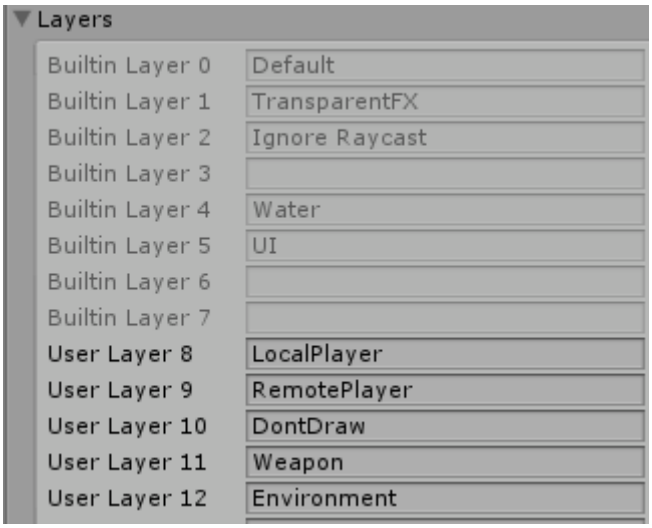


Imagen 46. Layers

Para realizarlo, definimos las diferentes capas que componen la visión del juego. De esta manera no solo podemos separar la visión de las armas sino que también podemos filtras los elementos que visualizan el jugador local y los jugadores remotos.

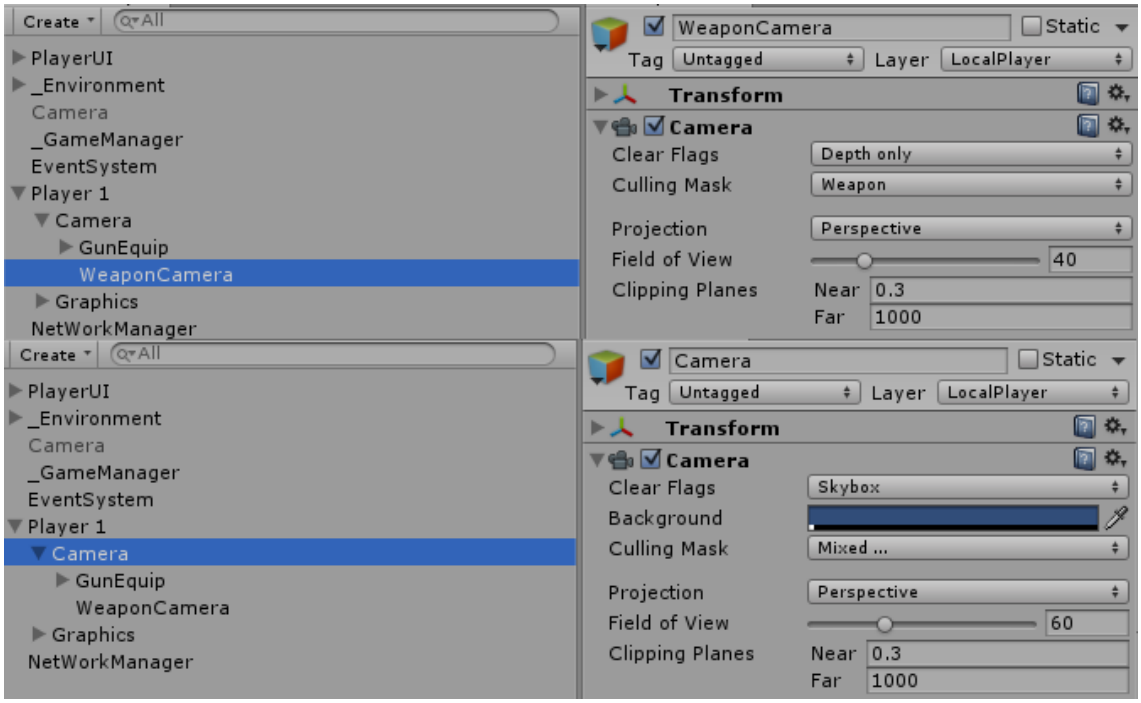


Imagen 47. Camera y WeaponCamera

De esta manera además podemos alterar el campo de visión y distancia del mismo de manera independiente, lo que es idóneo para los cambios entre diferentes tipos de armas con que se equipe el personaje.

5.2.5 Efectos de disparo y explosiones

Al igual que con los efectos visuales de los propulsores, usaremos los prefabs facilitados por Unity.

En el caso de las armas, a través del sistema de partículas podremos simular el humo y los destellos de disparo, y lo mismo podemos decir con los puntos de impacto donde aparte de humo y destello añadiremos también partículas incandescentes para dar la sensación de que las balas hacen saltar chispas sobre los objetos a los que se dispara.

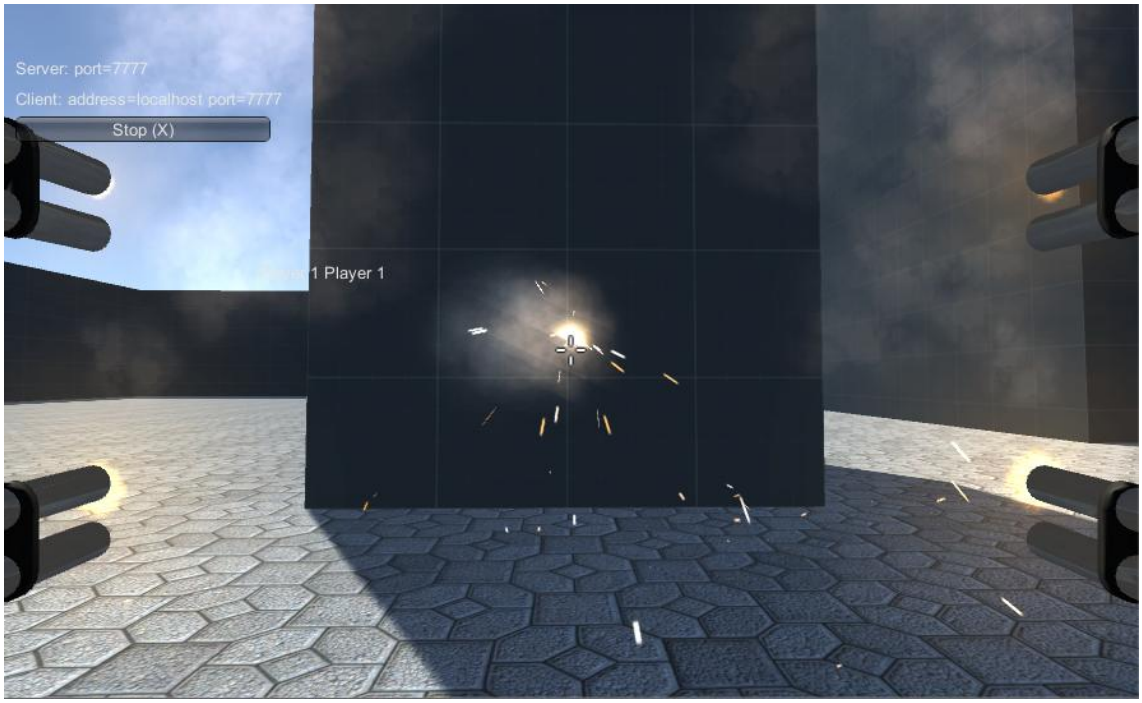


Imagen 48. Dron disparando

```
[ClientRpc]
void RpcShootEffect()
{
    for(int i = 0; i < weaponManager.GetCurrentWeaponGraphics().fireSmoke.Length; i++)
    {
        weaponManager.GetCurrentWeaponGraphics().fireSmoke[i].Play();
    }
}
[ClientRpc]
void RpcHitEffect(Vector3 _pos, Vector3 _normal)
{
    GameObject _hitEffect =(GameObject)Instantiate(weaponManager.GetCurrentWeaponGraphics().hitEffect, _pos, Quaternion.LookRotation(_normal));
    Destroy(_hitEffect, 1.5f);
}
```

Tabla 65. Script efectos disparo

Dado que a diferencia de los efectos del propulsor estos no están en constante ejecución, únicamente aparecen cuando se pulsa el botón de disparar, es necesario la incorporación de un script que se encargue de estos puntos:

Por un lado que en cada disparo aparezca evidentemente el efecto de este, y por otro el efecto de impacto, que en este caso necesitamos que se cree un objeto diferente en cada punto de impacto, reproduzca la animación y al cabo de un tiempo se destruya.

Para finalizar, añadiremos un efecto de explosión cuando un personaje muera y otro de reaparición.

Imagen 49. Explosión del Dron

5.2.6 Limitación del tiempo de vuelo

Dado que en la primera iteración añadimos la posibilidad de que el Dron pueda volar, esta característica, los jugadores pueden abusar de ella sin limitación alguna. Para solucionarlo, una posible medida es crear una barra que represente el tiempo o "combustible" disponible para poder volar.

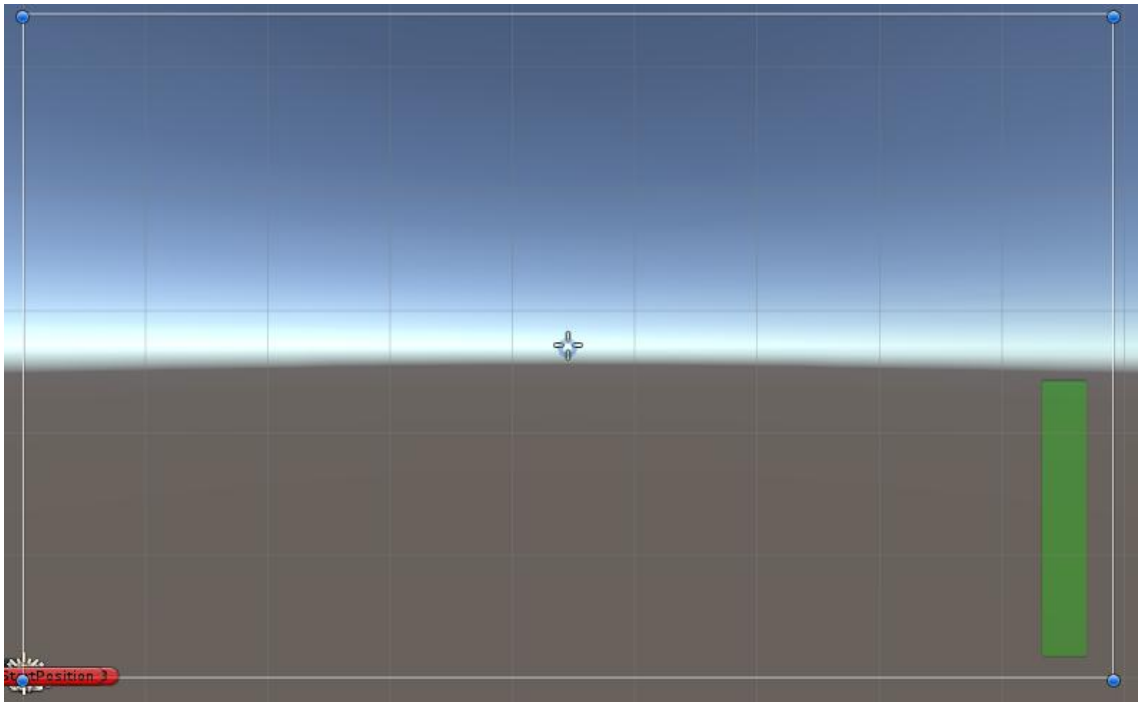


Imagen 50. Canvas PlayerUI

Aprovechando el "Canvas" que diseñamos para el punto de mira del arma, añadimos a este un panel compuesto de otro subpanel en el cual incluimos una imagen medio transparente con la que podamos indicar la cantidad de tiempo o combustible que le queda al Dron. De esta manera solo tenemos que "transformar" la altura de la imagen para representar la cantidad que dispone.

```
  void SetFuelAmount(float _amount)
    {
        fuelFill.localScale = new Vector3(1f, _amount, 1f);
    }
```

Tabla 66. SetFuelAmount

```
if (Input.GetButton("Jump")&& fuelAmount > 0f)
     {
      fuelAmount -= fuelBurnSpeed * Time.deltaTime;
      if (fuelAmount > 0.01f)
        {
         force = Vector3.up * jumpForce;
         SetJointSettings(0f);
        }
     }
     else
     {
      fuelAmount += fuelRefillSpeed * Time.deltaTime;
      SetJointSettings(elasticity);
     }
fuelAmount = Mathf.Clamp(fuelAmount, 0f, 1f);
motor.jumper(force);
```

Tabla 67. BurnSpeed

De esta manera, a través de una función que re-escala la imagen para indicarnos visualmente el combustible que nos queda y otra que vaya restando y sumando combustible en función de su uso, podemos limitar el uso del salto, permitiendo así a los jugadores que creen sus propias estrategias para sacarle el mayor provecho.

5.2.7 Bloqueo del ratón

Dado que no hemos establecido ningún bloqueo para el puntero del ratón, este se nos puede salir fácilmente de la pantalla del juego y sin querer pulsar sobre zonas que no debería.

```
if (Input.GetButtonDown("Esc"))
        {
            Cursor.lockState = CursorLockMode.None;
            Cursor.visible = true;
        }
        else if (Input.GetButtonDown("Fire1") && (Cursor.lockState ==
CursorLockMode.None))
        {
            Cursor.lockState = CursorLockMode.Locked;
            Cursor.visible = false;
        }
```

Tabla 68. CursorLock

Para ello, a través de un script, podemos establecer que por defecto el ratón este oculto y bloqueado y que a través del botón de escape este se desbloquee y que vuelva a bloquearse y desaparecer en el momento que se use el botón de disparo del juego.

5.2.8 Flotar sobre cualquier superficie

Tal y como estamos, en el momento en el que nuestro personaje baje a la posición "1" del eje Y en el espacio, este no bajará pode debajo de esa altura y permanecerá flotando. Sin embargo si lo situamos sobre cualquier otro objeto por encima de esa altura, este permanecerá al ras del suelo.

La solución que se ha llevado a cabo para poder hacer que nuestro personaje flote consiste en lanzar un raycast hacia el suelo desde el cuerpo del personaje y cada vez que detecte un objeto que tenga definido la capa "environmentMask", se establezca una nueva posición al "configurableJoin" a partir del punto devuelto por el raycast.

```
RaycastHit _hit;
        if (Physics.Raycast(transform.position, Vector3.down, out _hit, 100f,
environmentMask))
        {
            joint.targetPosition = new Vector3(0f, -_hit.point.y, 0f);
        }
        else
        {
            joint.targetPosition = new Vector3(0f, 0f, 0f);
        }
```

Tabla 69. código "flotar"

Este código lo deberemos situar dentro de una función update, para que de esta manera Unity compruebe continuamente si debe o no flotar el personaje en función de una altura data en relación al objeto que el personaje tiene justo debajo.

5.3 Tercera Iteración

Llegados por fin a la tercera iteración, podemos ir observando como el juego va empezando a tomar forma.

A partir de aquí necesitaremos añadir un servidor para que el juego no solo se pueda ejecutar en red privada sino también de manera online, un menú de juego para que no se empice a jugar directamente sin ni siquiera poder elegir nivel, acceso al listado de servidores para poder unirse a su partida y mapas de los diferentes niveles del juego. Por tanto los puntos a desarrollar serian:

- Matchmaking
- ServerList, hosting y joining
- Mapas
- Menú de pausa
- Menú de juego

5.3.1 Diagrama de clases de diseño

Los diagramas de clases de diseño correspondientes a esta tercera iteración son los siguientes:

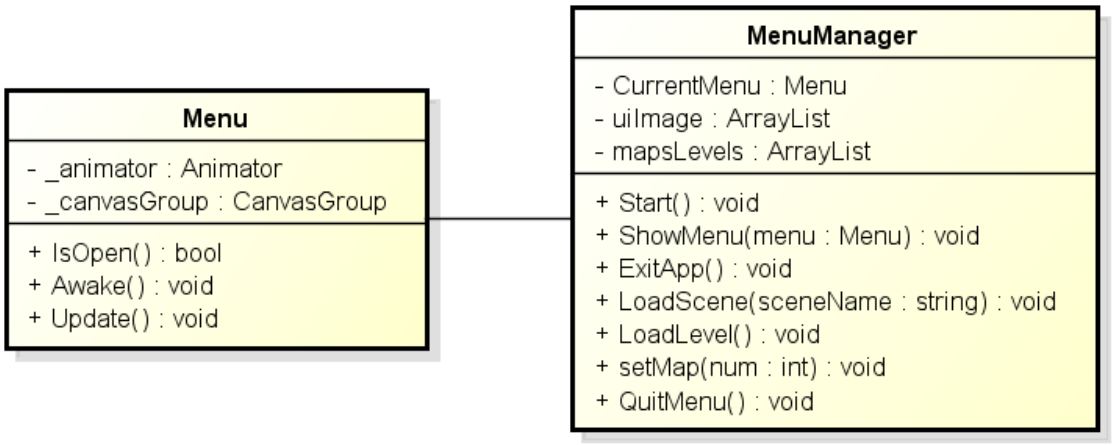


Imagen 51. Diagrama de clases de Diseño Iteracion 3: Menu

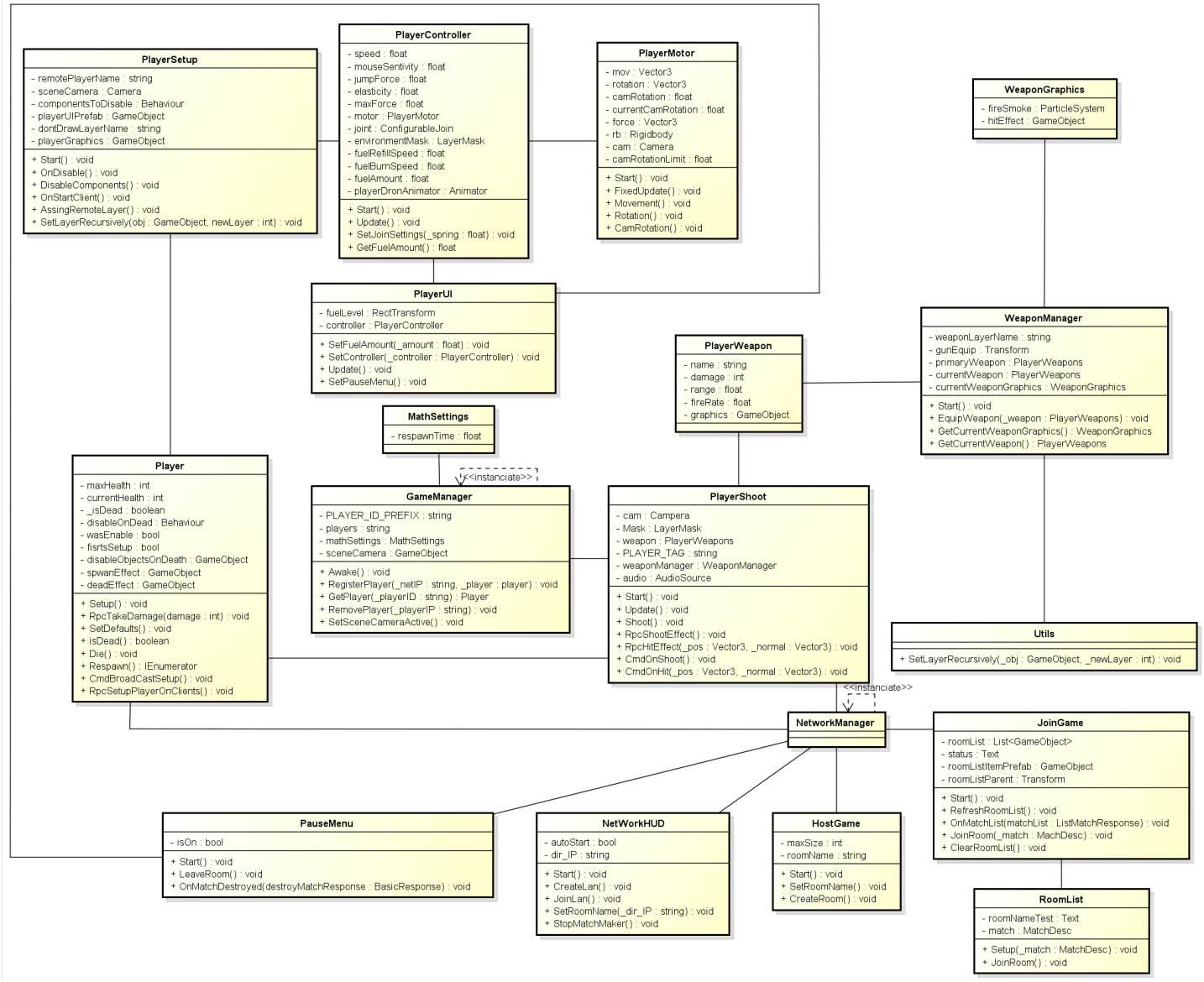


Imagen 52. Diagrama de clases de diseño Iteración 3: Game

5.3.2 Matchmaking

Como se ha mencionado previamente, en el estado actual de desarrollo de este juego, solo podemos realizar partidas en red privada o local. Para poder realizar partidas completamente Online necesitamos un servidor que nos aporte un servicio de Matchmaking.

Unity nos ofrece este servicio de manera gratuita con una limitación del número de usuarios que se pueden conectar de manera simultánea (si deseamos que se puedan conectar más es necesario pagar por estos servicios). Para activarlo, nos dirigimos a barra superior de Unity u seleccionamos Windows→Services.

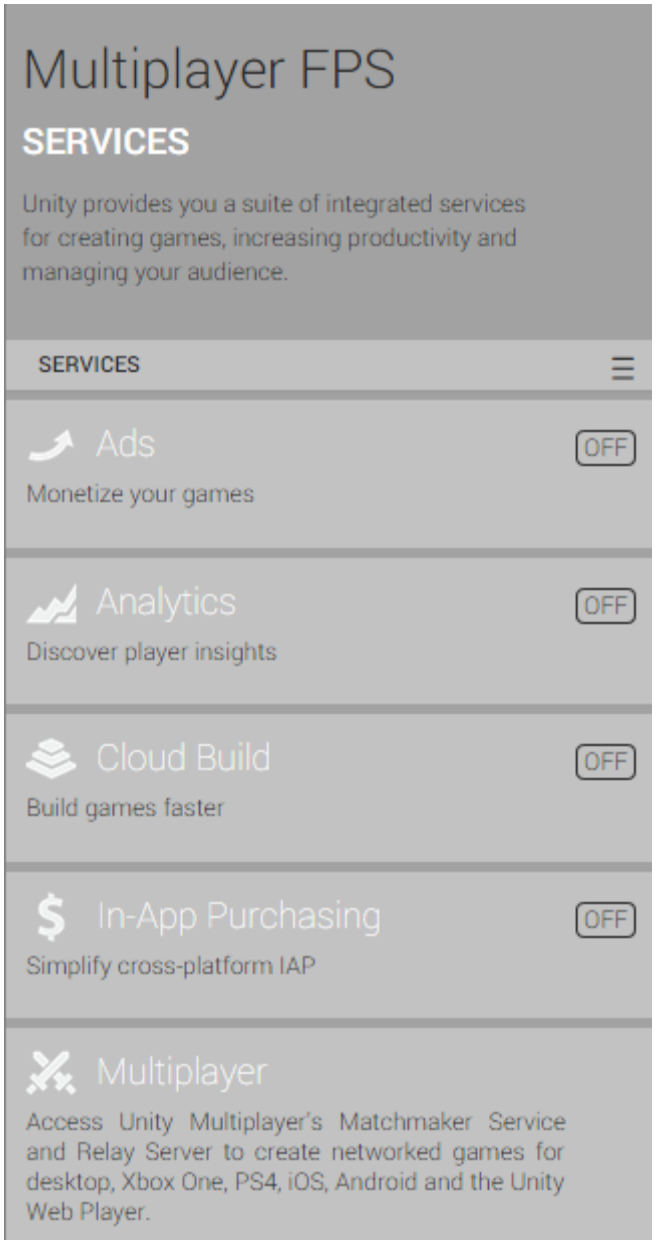


Imagen 53. Unity services

Desde aquí accederemos a nuestra cuenta registrada de Unity (si no disponemos de ella es necesario crear una) y elegiremos el servicio multiplayer.

Una vez pulsado, se abrirá en el navegador de internet la página de nuestra cuenta en Unity developer y nos dará la opción de crear el servicio de multijugador, el cual en función del número máximo de conexiones simultáneas de jugadores nos costará más o menos dinero. En este caso se establece en veinte jugadores, que es el número máximo de conexiones simultáneas que ofrece Unity de manera gratuita.

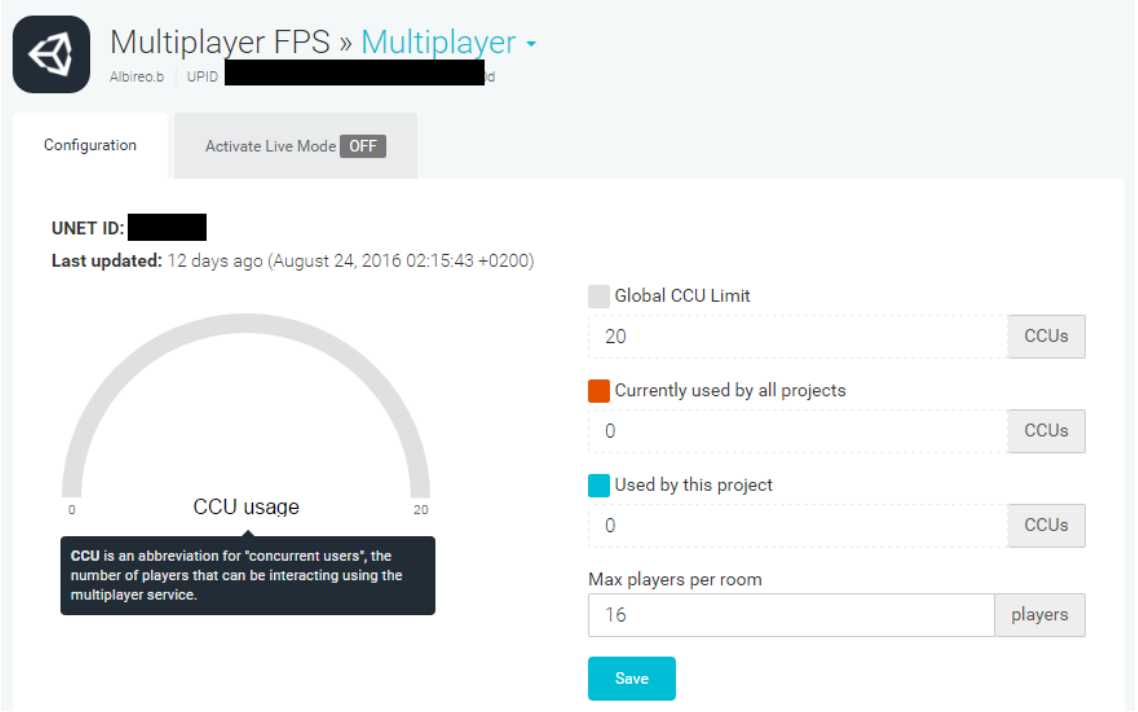


Imagen 54. Multiplayer Service

5.3.3 ServerList, hosting y joining

Creado el servicio multijugador el siguiente paso es conseguir interactuar con él, es decir, ser capaz de crear las "salas" online para jugar y una vez creadas que el resto de usuarios puedan visualizarlas y seleccionar una a su elección para unirse a dicha partida.

Empecemos por el host, necesitamos un menú desde que el usuario pueda introducir un nombre de sala y crear con ese nombre la partida Online.

```
    public void SetRoomName(string _roomName)
    {
        roomName = _roomName;
    }

    public void CreateRoom()
    {

        if (roomName != "" && roomName != null)
        {
            Debug.Log(roomName+"de tamaño" +maxSize+"creada");
            netkorkManager.matchMaker.CreateMatch(roomName,maxSize,true, "",
netkorkManager.OnMatchCreate);
        }
    }
```

Tabla 70. HostGame

Para ello creamos dos funciones, SetRoomName y CreateRoom. SetRoomName, se encargara de recorger el nombre de la sala introducido por el usuario y CreateRoom creará la sala bajo ese nombre.

Dado que cada sala tiene un identificador único que se genera automáticamente por Unity en el momento que se crea, no hay problema de que el nombre sea repetido.

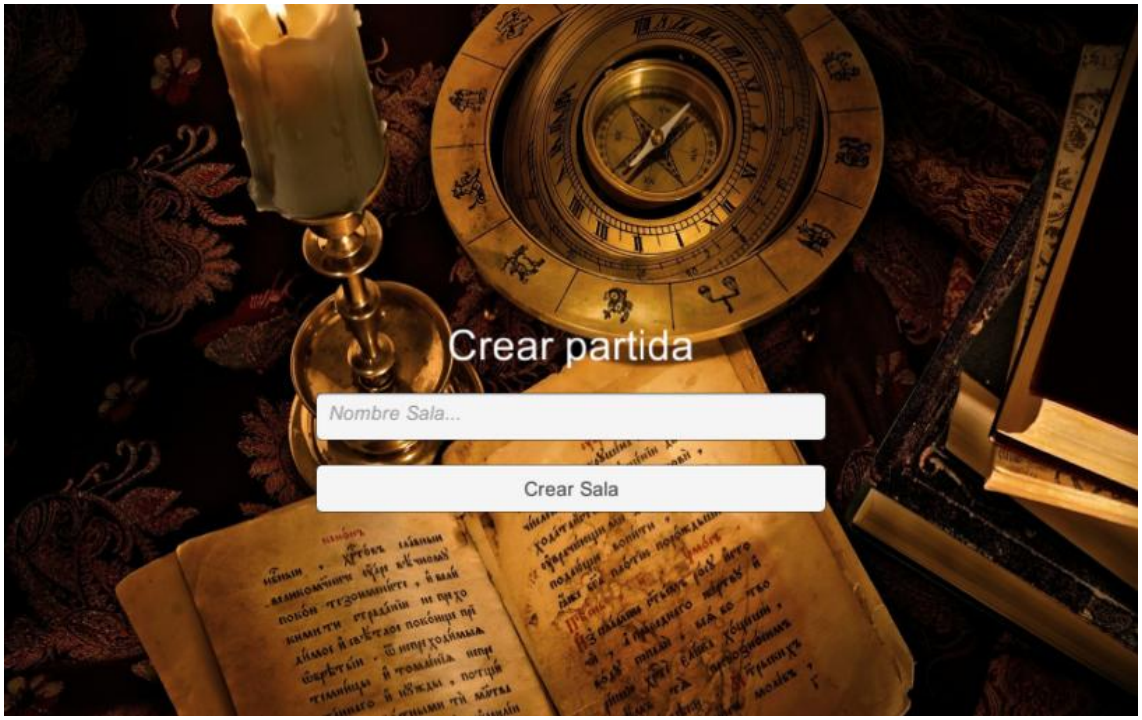


Imagen 55. Crear partida Online

Dicho esto, creamos un menú con un "InputField" para introducir el nombre de la sala y un botón para poder comenzar la partida Online.

Ahora pasamos a la creación de un menú que nos muestre el listado de salas disponibles para unirse a la partida. Como el número de salas es indeterminado, creamos un botón como prefab y de esta manera al acceder a la información de cada sala existente añadimos una réplica de este botón donde muestre el nombre de las diferentes salas de cada uno junto con el número de usuarios conectados a dicha sala en ese momento.

```
    public void Setup(MatchDesc _match, JoinRoomDelegate _joinRoomCallback)
    {
        match = _match;
        joinRoomCallback = _joinRoomCallback;

        roomNameText.text = match.name + " (" + match.currentSize + "/" +
match.maxSize + ")";
    }

    public void JoinRoom()
    {
        joinRoomCallback.Invoke(match);
    }
```

Tabla 71. RoomList

Para ello se crea una función que establezca el texto del botón y otra que al pulsarle se una a la sala cuyo nombre muestra dicho botón.

```
    public void RefreshRoomList()
    {
        ClearRoomList();
        networkManager.matchMaker.ListMatches(0,20,"", OnMatchList);
        status.text = "Loading...";
    }
 public void OnMatchList(ListMatchResponse matchList)
    {
        status.text = "";

        if (matchList == null)
        {
            status.text = "Couldn't get room list.";
            return;
        }

        foreach(MatchDesc match in matchList.matches)
        {
            GameObject _roomList = Instantiate(roomListItemPrefab);
            _roomList.transform.SetParent(roomListParent);

            RoomList _roomListItem=_roomList.GetComponent<RoomList>();
            if (_roomListItem != null)
               _roomListItem.Setup(match, JoinRoom);

            roomList.Add(_roomList);
        }
        if (roomList.Count == 0)
            status.text = "No rooms at the moment";
     }

    public void JoinRoom (MatchDesc _match)
    {
        networkManager.matchMaker.JoinMatch(_match.networkId, "",
networkManager.OnMatchJoined);
        ClearRoomList();
        status.text="Accediendo...";
    }
```

Tabla 72. JoinGame

Como ya se ha mencionado, la lista de salas a las que puede acceder el jugador es de tamaño variable, por lo que necesitamos unas funciones que refresquen dicha información y añadan y eliminen los botones correspondientes de cada sala además de realizar la conexión a la sala correcta al momento de pulsar alguno de los susodichos botones.

Para realizarlo, a partir de la función "RefreshRoomList", solicitarmos al servidor la información referente a cada sala y esta llamará a la función "Setup" que se encargará de recorrer el listado y crear cada botón con la información facilitada por el servidor. Finalmente la función "JoinRoom" se encargará de conectar con la partida del botón correspondiente seleccionado.

Imagen 56. Menú unirse a partida

Realizada toda la parte de scripting, solo nos queda añadir el menú, para ello se crea un nuevo canvas con un panel el cuál se irá rellenando con los correspondientes botones de cada sala existente en el servidor de Unity y un botón de refrescar para actualizar la información de las salas existentes.

5.3.4 Mapas

Como se pudo comprobar en la construcción del modelo 3D del personaje, modelar y crear objetos lleva mucho tiempo y dedicación. Por eso para la realización de los mapas de la aplicación hemos partido, por una parte de modelos ya existentes que nos facilita Unity en su Asset Store como es el "Angry Bot" a partir del cual se ha construido un nivel completamente nuevo en base a los prefabs (objetos 3D y efectos visuales) que incluía.

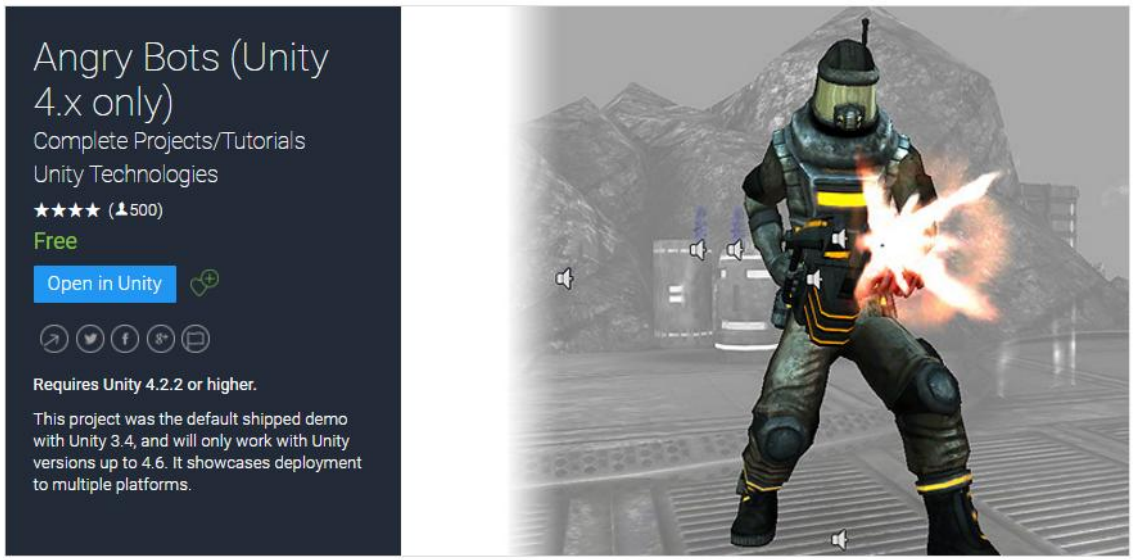


Imagen 57. Asset Angry Bots

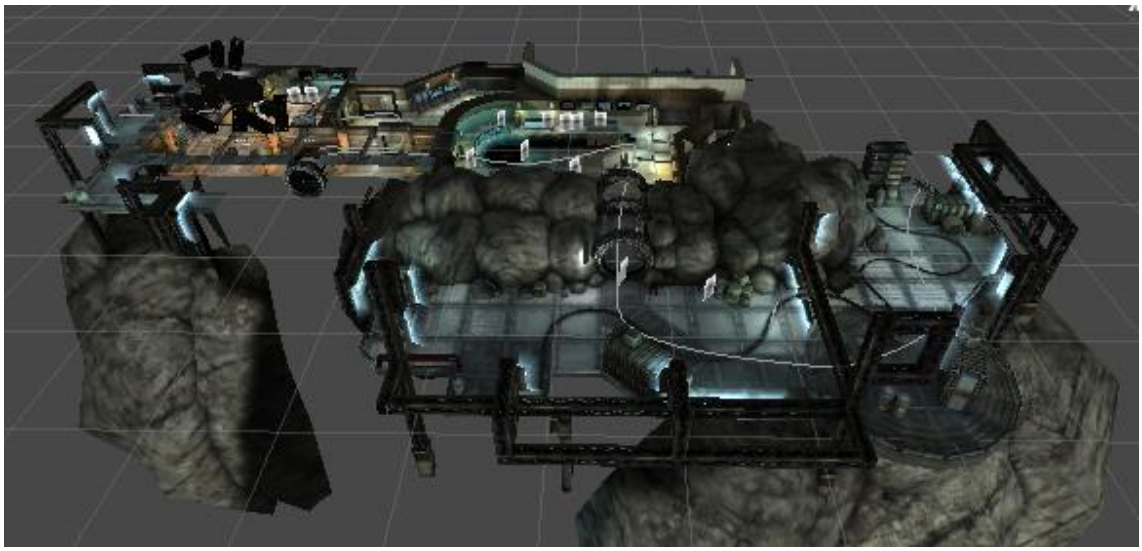


Imagen 58. Nivel Original de Angry Bots

Imagen 59. Nivel Syfy desarrollado a partir de los prefabs de Angry Bots

Debido al gran tiempo que ha requerido el desarrollo de este nivel, el siguiente nivel (destroyed city) se ha obtenido directamente a partir de un mapa ya existente (también gratuito) al que se le han ido realizando algunos pequeños cambios y modificaciones para adaptarlo a las necesidades de la aplicación.

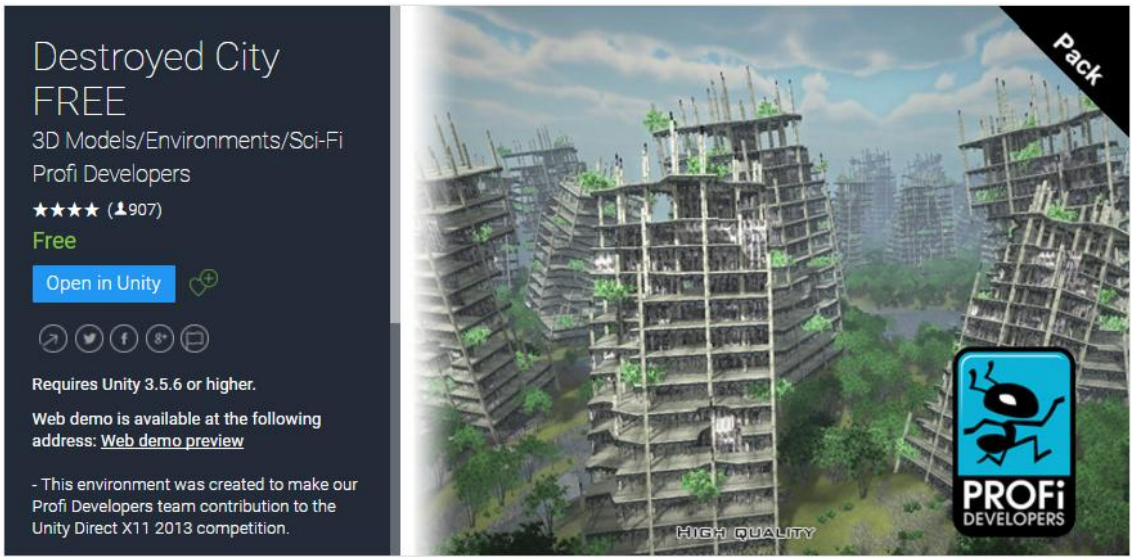


Imagen 60. Destroyed City

5.3.5 Menú de pausa

Para el menú de pausa se ha aprovechado el prefab "PlayerUI" ya creado que actualmente utilizamos para mostrar el combustible que le queda al jugador y la mirilla.

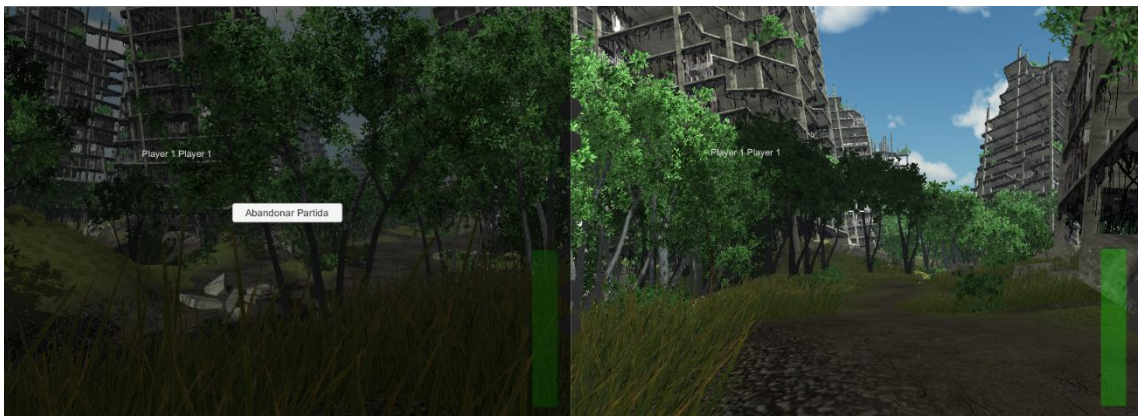


Imagen 61. Menú de pausa

Añadimos al prefab "Player UI" una nueva imagen con base de color negro medio transparente y el botón de "Abandonar Partida" y dejamos estos componentes por defecto deshabilitados.

```
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
            SetPauseMenu();
    }

    void SetPauseMenu()
    {
        pauseMenu.SetActive(!pauseMenu.activeSelf);
        PauseMenu.IsOn = pauseMenu.activeSelf;
    }
```

Tabla 73. Activar Menú de pausa

A través del siguiente script establecemos de esta manera que al pulsar la tecla "Escape" el juego entre en pausa. Ahora bien, en el momento en el que entra en pausa, la única acción que hace es oscurecer la pantalla y mostrarnos un botón. Necesitamos que el jugador sea incapaz de mover el personaje mientras el menú de pausa está activo. Para ello, declaramos un booleano que se ha denominado "IsOn", con el que comprobaremos si el menú está activo o no antes de dejar al usuario utilizar el resto de controles del juego.

```
if (PauseMenu.IsOn)
            return;
```

Tabla 74. Comprobación de si el menú pausa está activo

Con este fragmento de código añadido en los scripts de "PlayerShoot" y "PlayerController" justo antes de comprobar las pulsaciones de teclado, podemos establecer fácilmente que se bloqueen los controles del personaje en el momento que se active el menú de pausa.

El siguiente paso es dar la funcionalidad al botón de abandonar partida, para el cual le asignamos la función que se muestra en la siguiente tabla.

```
    public void LeaveRoom()
    {

       if (networkManager.matchMaker!=null)
        {
            MatchInfo matchInfo = networkManager.matchInfo;
            networkManager.matchMaker.DropConnection(matchInfo.networkId,
matchInfo.nodeId, OnMatchDestroyed);
        }
            networkManager.StopHost();
            NetworkManager.Shutdown();

    }
```

Tabla 75. LeaveRoom

Con este código en caso de tratarse de una partida Online, antes de detener el networkManager, desconecta la conexión con el servidor de Unity, por lo que así podemos usar la misma función en los diferentes escenarios (es decir, nos vale tanto para partidas de tipo LAN como las de tipo WAN).

5.3.6 Menú de juego

Finalmente llegamos a otro de los puntos clave del juego que afecta en las primeras impresiones que pueda tener un usuario al iniciar por primera vez la aplicación, el menú de juego.

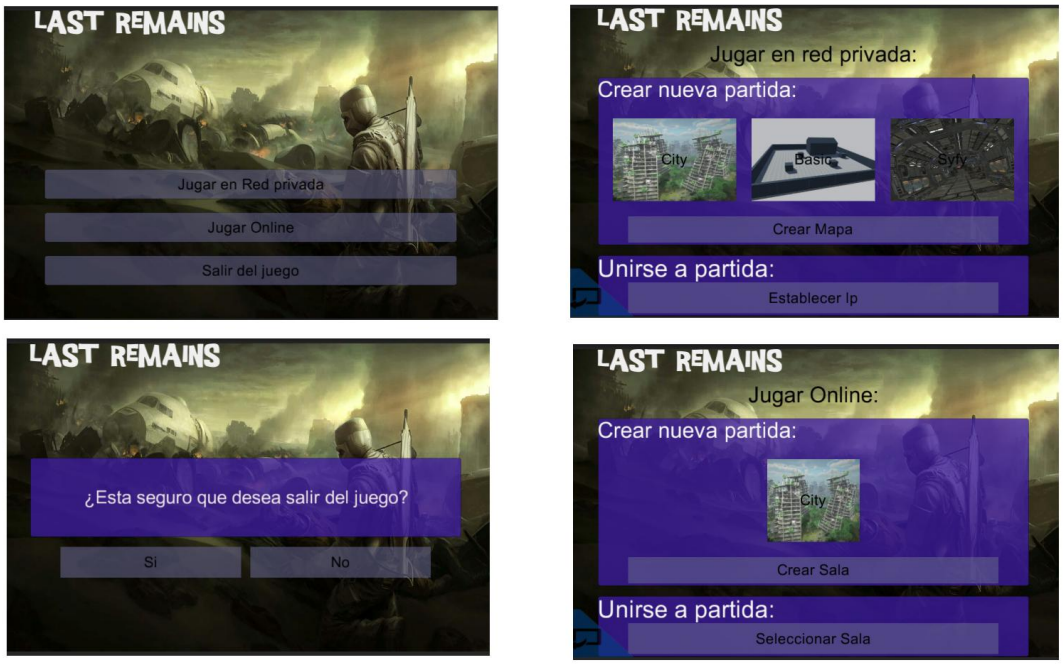


Imagen 62. Menú principal

Como se puede observar en la imagen anterior el menú principal del juego se ha diseñado con cuatro paneles que va desde el menú de inicio, menú de acceso LAN, menú de acceso WAN y el menú de salir.

Estos paneles se están en un mismo scenario, por lo que para mostrarles correctamente, la mejor manera es a través del animator. Declaramos por medio de este que el menú actual o seleccionado sea visible e interactuable, mientras que el resto pasan a estar ocultos.

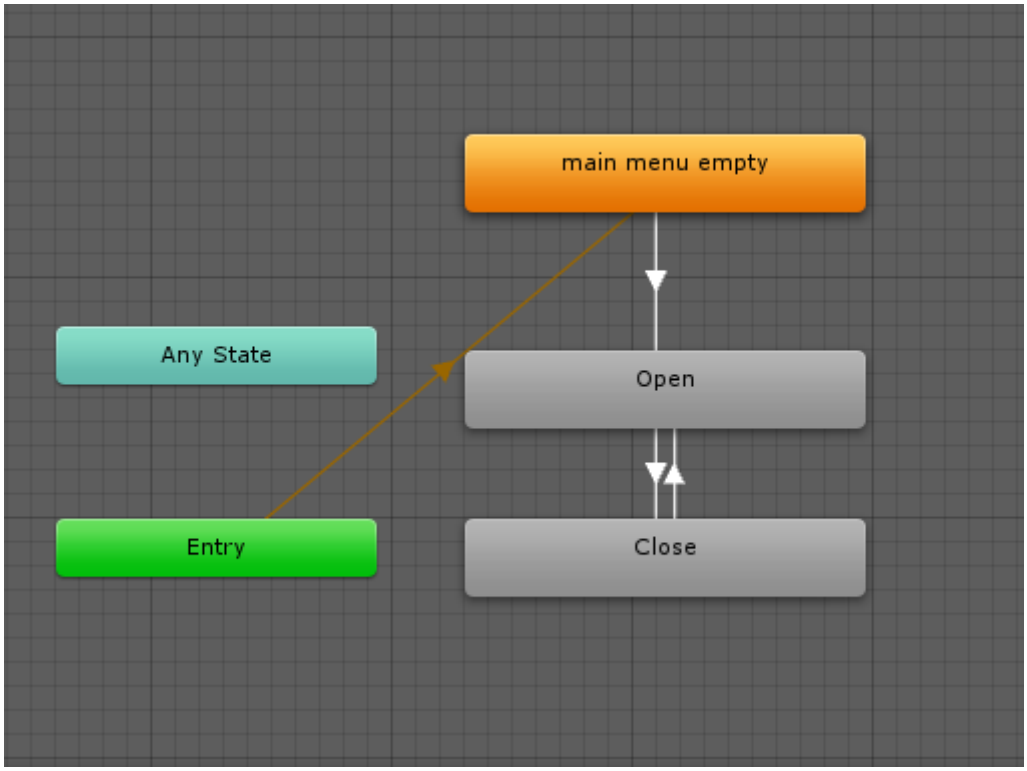


Imagen 63. Animator Main Menu

De esta manera, solo tenemos que añadir a botón encargado del cambio de escena un script que se encargue de habilitar el menú al que se accede y que deshabilite el resto.

```
        public void ShowMenu(Menu menu)
        {
                if (CurrentMenu != null)
                        CurrentMenu.IsOpen = false;

                CurrentMenu = menu;
                CurrentMenu.IsOpen = true;
        }
```

Tabla 76. ShowMenu

Con esta función, partiendo de un menú visible o en este caso "abierto", en el momento que la llamemos, la pasamos el nuevo menú que queremos mostrar por pantalla, así el anterior menú se "cierra" y se abre el nuevo menú seleccionado.

```
    public void ExitApp()
    {
        Application.Quit();
    }

    public void LoadScene(string sceneName)
    {
        SceneManager.LoadScene(sceneName);
    }
```

Tabla 77. LoadScene y ExitApp

Finalmente con la función LoadScene, podemos cargar el escenario que deseemos del juego en base a su nombre y con la función ExitApp cerrar la aplicación.

6 Conclusiones

En este capítulo se exponen las dificultades encontradas en el desarrollo de este proyecto, los objetivos alcanzados, las posibles líneas de trabajo futuro y las conclusiones extraídas.

6.1 Dificultades encontradas en el transcurso del proyecto

- **Falta de tiempo:**
  Este proyecto, a diferencia de lo que se había pensado en un primer momento es muchísimo más extenso de lo que se creía.
  Realizar un juego de FPS, por muy simple que sea, necesita de muchos más puntos a considerar de los que originalmente se plantearon para su desarrollo y que por una u otra razón, en estas primeras etapas de planificación pasaron desapercibidos y/o no se consideraron importantes.
  Por eso ha sido necesario acortarle eliminando múltiples funcionalidades como podrían ser por ejemplo lucha por equipos, diferentes modelos de jugadores, personalización de personajes, sistema de rankings, cambio de armas, modo historia… y debido a esta limitación de tiempo de la que se disponía para su desarrollo y entrega a tiempo no ha sido posible realizarlo.
- **Organización del proyecto:**
  Como se ha podido observar en la planificación estimada, en las primeras etapas de este proyecto aún no se tenía una idea completamente clara de todos los puntos necesarios para el desarrollo del juego, traduciéndose en una estimación temporal demasiado optimista, por no mencionar de que se partía del supuesto base de que no iba a realizarse la más mínima interrupción temporal en el trabajo diario del mismo.
- **Falta de conocimientos:**
  Pese a que se realizó un estudio previo del manejo y funcionalidades de Unity, en muchas ocasiones debido al desconocimiento de las características que ofrece este, el desarrollo de varios de los puntos desarrollados del juego llevaron un tiempo de dedicación muy superior a la dificultad que planteaban. Como por ejemplo el uso de las animaciones que originalmente se intentó realizar a través de scripts en vez de la herramienta de animación que nos facilita el entorno o los métodos de asignación del contenido de un objeto a una variable el cual no era necesario la búsqueda del componente vía scripts sino que se le podía asignar directamente el contenido desde el propio inspector sin tener que buscar dicho objeto.
- **Actualizaciones del entorno Unity:**
  Durante el tiempo que ha llevado desarrollar esta aplicación se ha actualizado Unity en dos ocasiones. Debido a esto, pese a que se solucionaron algunos errores gracias a las actualizaciones (como el que funcionase correctamente la función "match.movetowardangle" o que el servicio de networking dejara de funcionar al reentrar al mismo mapa), algunas de las funciones que se usaban en ese momento se quedaban obsoletas, siendo necesario sustituirlas por otras similares o actualizarlas a las nuevas necesidades (como en el caso del componente de sonido "AudioSource", cuya manera de implementarse cambió drásticamente con la nueva versión de Unity.

6.2 Objetivos alcanzados

Como se puede observar tras haber repasado el plan de iteraciones, todos los objetivos planteados al comienzo de esta memoria han podido realizarse por completo.

Los objetivos a cumplir eran los siguientes:

- Recogida de información y aprendizaje sobre el uso de las herramientas que se van a usar (Unity 3D, Cinema 4D y Adobe Photoshop).
- Modelado de diferentes modelos 3D (como los personajes jugables y elementos decorativos del mapa) a través de la herramienta "Cinema 4D".
- Modelado y creación de mapas a través de "unity 3D", con la complementación de "Cinema 4D".
- Desarrollo de los diferentes Scripts en C# para establecer la mecánica de funcionamiento del juego, junto con las diferentes interacciones, eventos, colisiones y resto de reglas que permitan el funcionamiento correcto del juego.
- Sincronización de los movimientos y acciones de los diferentes jugadores online entre cliente y servidor.
- Creación de la interfaz y menús de la aplicación.

Aunque algunos no se han podido profundizar todo lo que se había visionado en un principio, como es por ejemplo el modelado de objetos 3D que se pretendía originalmente diseñar todos los modelos 3D del proyecto y al final solo ha dado tiempo al desarrollo del modelo del personaje ya que el resto se ha partido de modelos ya existentes.

6.3 Líneas de trabajo futuro

Como ya se ha mencionado, este proyecto es muchísimo más extenso de lo que se creía en un primer momento, por lo que algunas de las líneas de trabajo podrían ser las siguientes:

- Establecer un menú de rankings de muertes y derribos entre los diferentes jugadores, el cual sea accesible en cualquier momento de la partida.
- Dar la posibilidad de que el mapa en el que se realiza la partida cambie al pasar cierto tiempo o algún jugador haya alcanzado cierto número de derribos de los contrincantes.
- Incluir un repertorio de armas variadas y limitación de la munición, siendo esta última accesible a partir de la recogida de objetos situados en ciertos puntos del mapa.
- Establecer un menú de armas que se puedan adquirir mediante el intercambio de puntos obtenidos al ganar a lo largo de los diferentes mapas.
- Implementación de ruidos y sonidos según la superficie sobre la que se muevan los personajes.
- Sistema de batalla por equipos o bandos.
- Implementación de múltiples modelos de personajes según el bando y que además sean personalizables.
- Posibilidad de que el jugador pueda definir su propio Nick online.
- Menú de personalización de controles, sonido y resolución.
- Generador de mapas propios.

6.4 Conclusiones

Llegados a este punto, se puede concluir que la creación de un videojuego por muy simple que parezca, es una tarea ardua que requiere mucha dedicación y tiempo.

El motor de desarrollo de Unity hace que el desarrollo de este sea más ágil y sencillo siempre y cuando no nos salgamos de las funcionalidades que este nos ofrece y la orientación con las que están destinadas para su uso. Pese a ello, es una herramienta que nos facilita en gran medida el desarrollo de juegos con unos resultados muy buenos y a diferencia de otras está muy bien documentada. Dispone de toda una comunidad de usuarios y una web de manuales donde podemos localizar las explicaciones de las diferentes funciones y funcionalidades que podamos necesitar y no sepamos su manejo y en gran medida está siempre actualizada con las últimas versiones de la herramienta.

En cuanto al modelado de objetos y el diseño de texturas es muchísimo más tedioso y lleva una gran tiempo de dedicación del que se había planteado en un principio. Ya que es necesario tener una idea clara del modelo a construir donde es necesario el diseño de bocetos y además la herramienta de modelado no siempre consigues los resultados y apariencia deseados teniendo que rehacer de nuevo el modelo. De ahí que haya sido necesario obtener algunos modelos ya construidos sobre los que trabajar en los mapas del juego.

Por último, este proyecto ha sido una experiencia muy positiva ya que me ha permitido aprender sobre el proceso de desarrollo de un videojuego y del reto que conlleva conseguir crear un juego decente que pueda salir de cara al público.

7 Contenido del DVD

El DVD adjunto contiene los siguientes archivos:

- Memoria del trabajo fin de master.
- Planificación estimada y real
- Diagramas UML en formato png.
- Código fuente del proyecto en sus diferentes fases de iteración.
- Aplicación compilada.
- Instalable de la última versión de Unity 3D actual.

8 Bibliografía

[1] Otakufreaks.com. (2016). Historia de los Videojuegos: El Origen y los Inicios. [online] Available at: http://www.otakufreaks.com/historia-de-los-videojuegos-el-origen-y-los-inicios/ [Accessed 11 Sep. 2016].

[2] Es.wikipedia.org. (2016). Framework. [online] Available at: https://es.wikipedia.org/wiki/Framework [Accessed 6 Sep. 2016].

[3] Paul, P., Goon, S. and Bhattacharya, A. (2012). HISTORY AND COMPARATIVE STUDY OF MODERN GAME ENGINES. [online] ResearchGate. Available at: https://www.researchgate.net/publication/236590476_HISTORY_AND_COMPARATIVE_STUDY_OF_MODERN_GAME_ENGINES [Accessed 11 Sep. 2016].

[4] Jahmel Coleman. (2013). Game Engines (Types & Purposes). [online] Available at: https://jahmelcoleman.wordpress.com/games-development/game-engines/ [Accessed 6 Sep. 2016].

[5] Gamecareerguide.com. (2016). What is a Game Engine?- GameCareerGuide.com. [online] Available at: http://www.gamecareerguide.com/features/529/what_is_a_game.php?page=1 [Accessed 6 Sep. 2016].

[6] Wikipedia. (2016). Unity (game engine). [online] Available at: https://en.wikipedia.org/wiki/Unity_(game_engine) [Accessed 6 Sep. 2016].

[7] Wikipedia. (2016). Unreal Engine. [online] Available at: https://en.wikipedia.org/wiki/Unreal_Engine#Unreal_Development_Kit [Accessed 6 Sep. 2016].

[8] Wikipedia. (2016). CryEngine. [online] Available at: https://en.wikipedia.org/wiki/CryEngine [Accessed 6 Sep. 2016].

[9] Epf.eclipse.org. (2016). OpenUP. [online] Available at: http://epf.eclipse.org/wikis/openup/ [Accessed 6 Sep. 2016].

[10] Anon, (2016). [online] Available at: http://repositorio.espe.edu.ec/bitstream/21000/6316/1/AC-SISTEMAS-ESPE-047042.pdf [Accessed 6 Sep. 2016].

[11] Gallagher, Brian P.: Software Acquisition Risk Management Key Process Area (KPA) — A Guidebook, Version 1.02. Carnegie Mellon University. HANDBOOK CMU/SEI-99-HB-001. 1999.

[12] Social Point Engineering Blog. (2014). Model View Controller pattern for Unity3D User Interfaces. [online] Available at: http://engineering.socialpoint.es/MVC-pattern-unity3d-ui.html [Accessed 11 Sep. 2016].

[13] Technologies, U. (2016). Unity - Manual: Unity Manual. [online] Docs.unity3d.com. Available at: http://docs.unity3d.com/Manual/index.html [Accessed 6 Sep. 2016].

[14] Unity3d.com. (2016). Unity - Learn - Modules. [online] Available at: https://unity3d.com/es/learn/tutorials [Accessed 6 Sep. 2016].

[15] Blog.csdn.net. (2016). 原文：http://blog.teotigraphix.com/2011/05/17/unity3d-uml-gameobject-cheat-sheet/ - 明明 - 博客频道 - CSDN.NET. [online] Available at: http://blog.csdn.net/a351945755/article/details/36201733 [Accessed 6 Sep. 2016].

[16]Marmoset. (2014). PBR In Practice. [online] Available at: http://www.marmoset.co/toolbag/learn/pbr-practice [Accessed 6 Sep. 2016].

[17]Anon, (2016). [online] Available at: http://http.maxon.net/pub/r12/doc/QuickstartC4DR12ES.pdf [Accessed 6 Sep. 2016].

9 Anexo

9.1 Manual básico de Unity 3D

Unity 3D está disponible tanto en versiones de pago, como una gratuita, siendo las funcionalidades de cada una las siguientes:

Compara los planes	Personal	Plus	Pro	Enterprise
Todas las prestaciones del motor	✓	✓	✓	✓
Todas las plataformas	✓	✓	✓	✓
Actualizaciones continuas	✓	✓	✓	✓
Sin regalías	✓	✓	✓	✓
Pantalla de inicio	Pantalla de inicio MWU	Animación personalizada o ninguna	Animación personalizada o ninguna	Animación personalizada o ninguna
Capacidad de ingresos	$100 mil	$200 mil	Ilimitado	Ilimitado
Unity Analytics	Analytics Personal	Plus Analytics	Analytics Pro	Analytics personalizado
Cloud Build de Unity	Cola estándar	Cola de prioridades	Compilaciones simultáneas	Agentes de compilaciones dedicados
Unity Multiplayer	20 usuarios simultáneos	50 usuarios simultáneos	200 usuarios simultáneos	Multiplayer personalizado
Unity Ads	✓	✓	✓	✓
Acceso a Beta	✓	✓	✓	✓
Editor UI Skin de la versión Pro		✓	✓	✓
Informes de ejecución		✓	✓	✓
Gestión de puestos flexible		✓	✓	✓
Software educativo para certificación de Unity		Acceso de 1 mes	Acceso de 3 meses	Acceso por 3 meses
Kits de assets		20 % de descuento	40 % de descuento	40 % de descuento
Acceso al código fuente			$	$
Soporte Premium			$	$

Imagen 64. Tabla comparativa de planes que ofrece Unity 3D

Como se puede observar, las funcionalidades ofrecidas únicamente por las versiones de pago no son impedimento alguno para desarrollar juegos potentes y de calidad con la versión gratuita, únicamente nos limita en cuanto a la facilidad de desarrollo, ya que al carecer de ciertas funcionalidades, de necesitarlas Unity 3D no nos impide desarrollarlas por cuenta propia.

9.1.1 Instalación de Unity 3D:

La instalación es muy sencilla, no requiere de configuración alguna y se puede usar no solo en Windows, sino también en Linux y MaxOs. Basta con descargar el paquete de instalación desde su página, ejecutarlo y seleccionar si junto con el motor de Unity deseamos también descargar el proyecto de ejemplo y el editor de código.

https://store.unity.com/es/download?ref=personal

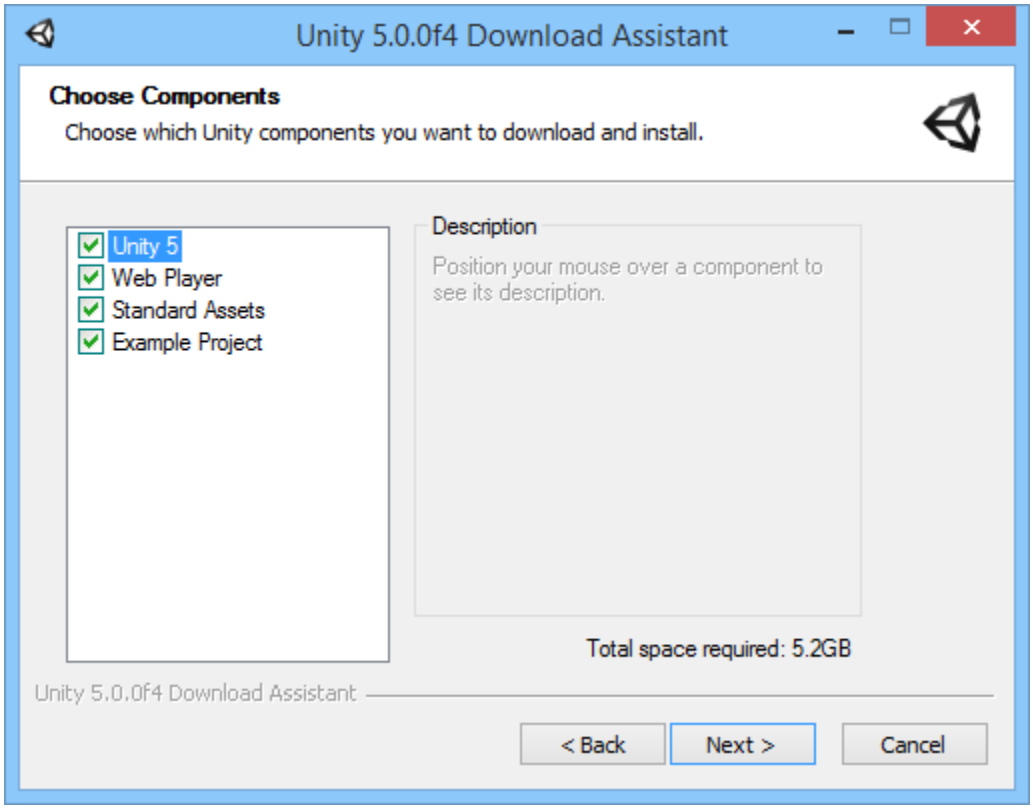


Imagen 65. Instalación de Unity: Selección de componentes a Instalar

Una descargado y ejecutado el asistente de instalación de Unity, este nos preguntará que componentes queremos instalar. Entre estos destacan los "standard assets" y el proyecto de ejemplo lo cuales contienen material y ejemplos básicos que nos son de gran ayuda para comenzar a desarrollar nuestro juego.

Tanto en la versión "pro", como la gratuita, Unity no dispone de un sistema de backups y de versionado, por lo que es recomendable crear los proyectos en un directorio de "Dropbox" o "Gmail" para poder disponer de un historial de versiones.

9.1.2 Estructura de un juego en Unity

Todos los juegos que se diseñan en Unity están compuestos de escenarios. Estos a su vez se componen por un conjunto de objetos los cuales a su vez están formados por un conjunto de componentes que definen su comportamiento y apariencia.

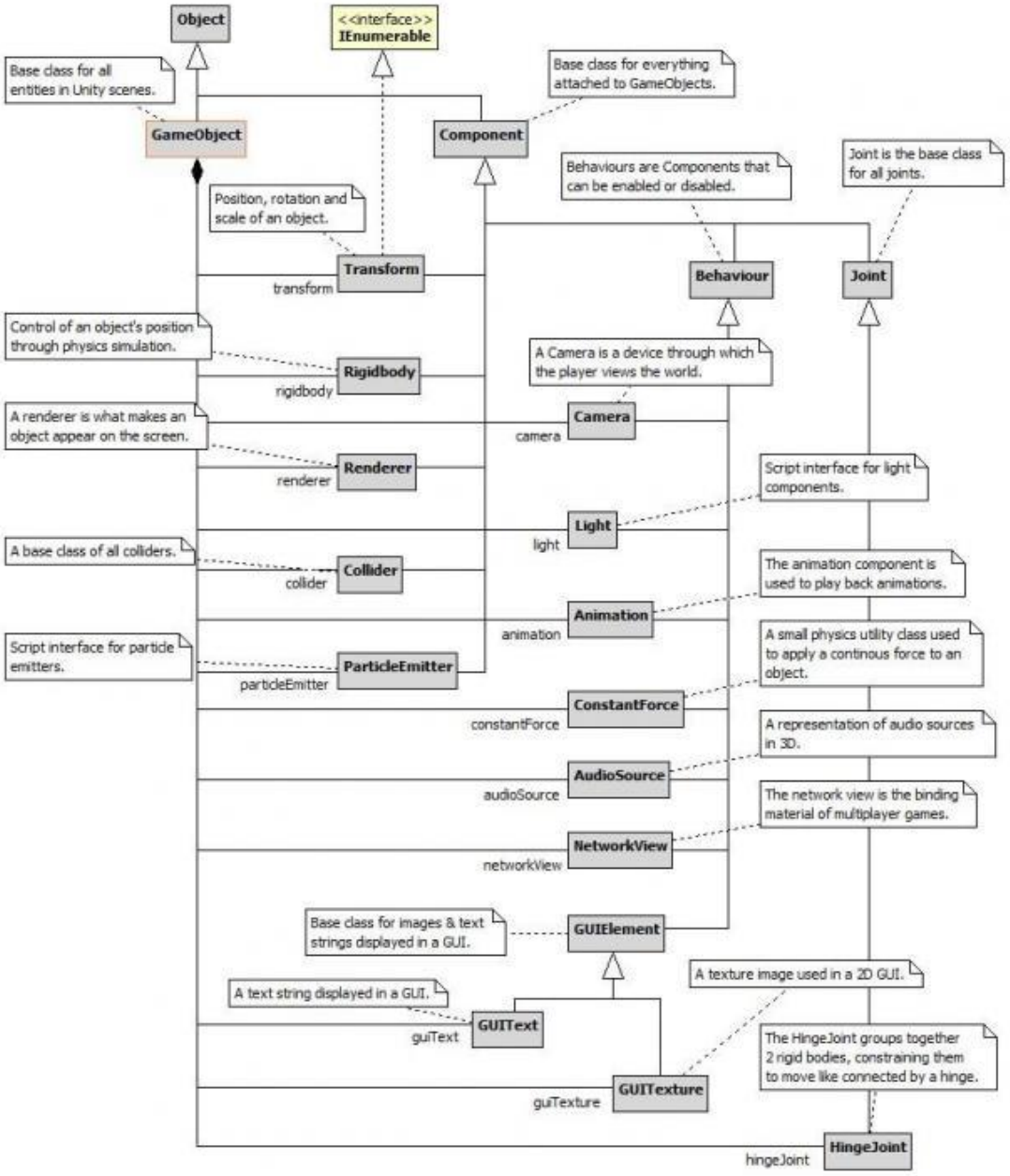


Imagen 66. Diagrama de clases de Unity Engine: Game Objects [15]

Por tanto dentro de cualquier "scene" o escenario que forma el nivel o menú de un juego de Unity, existe como mínimo un "game object" el cual posee uno o varios componentes para que al cargar el escenario estos realizan alguna acción o comportamiento durante la interacción del usuario (o con el paso del tiempo si se trata de una acción temporizada) con el juego.

9.1.3 Un vistazo rápido a la interfaz de usuario:

Una vez iniciado un nuevo proyecto en Unity, vemos una interfaz compuesta por varios paneles [13] o áreas completamente personalizables, las cuales podemos agrupar y colocar donde deseemos.

Normalmente este entorno se personaliza con no más de 5 paneles que son los más utilizados de los 10 que posee en total.

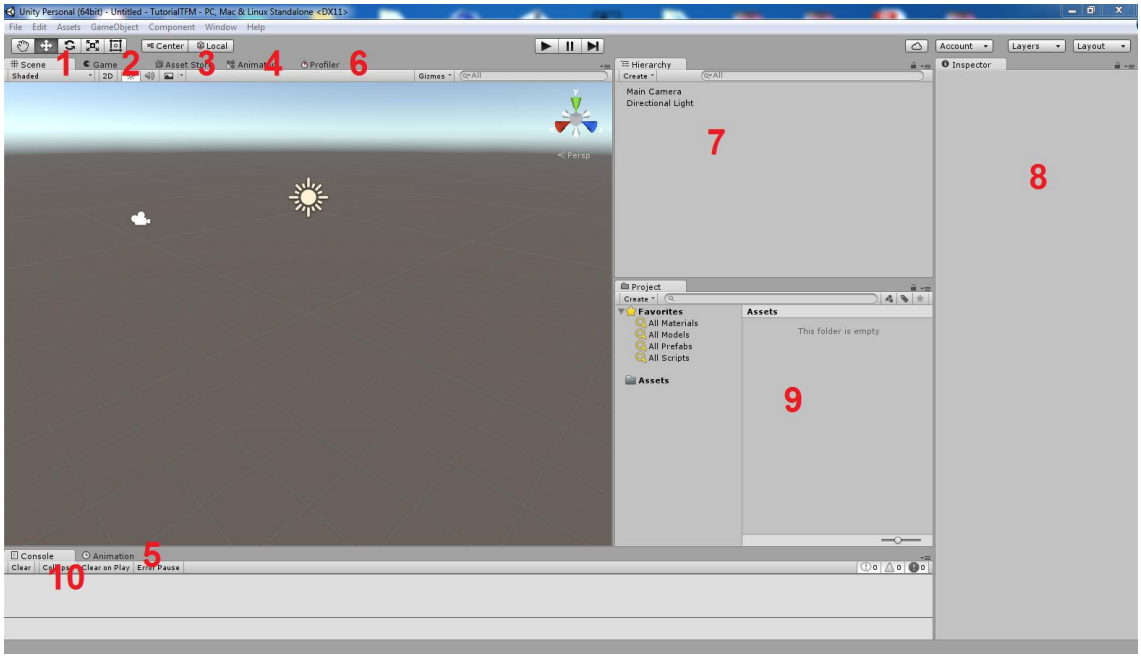


Imagen 67. Interfaz visual de Unity 3D

Cada uno de estos paneles dispone de estos símbolos que permiten maximizar, cerrar, añadir un panel y bloquear el panel para evitar cambiar su contenido actual.

Estos paneles sirven para lo siguiente:

1- **Scene:**

Muestra el "mundo virtual". Es el lugar donde construimos el escenario donde el juego va a desarrollarse. Desde este panel, colocamos y observamos el lugar donde va la interfaz, paisajes, objetos, luces y otros elementos del juego.

Dentro de este panel aparecen diferentes botones y menús situados en la parte superior:

Imagen 68. Barra de control del panel Scene

El primero (situado en la izquierda superior), se conoce como "Draw Mode". Nos permite controlar el modo de dibujo, renderizado e iluminación de la escena.

Los siguientes cuatro botones son respectivamente, "2D" para fijar la cámara desde vista frontal, "scene lighting" habilita y deshabilita la iluminación de la escena actual, "Audio" habilita y deshabilita el audio, "scene effects" que habilita y deshabilita los efectos de "skybox", niebla y llamaradas. Por último el desplegable de "gizmos" que son

iconos que nos aparecen en el editor de escena y no indican el tipo de objeto o recurso que representan.

Para poder mover, rotar, seleccionar y transformar los objetos que observamos y navegar por el interior de este mundo virtual, disponemos de los botones de control y de un gizmo que representa los diferentes ejes de coordenadas de una vista 3D cuyas funciones son las siguientes:

Imagen 69. Botones de control y gizmo de vistas 3D

-  Permite desplazarse alrededor de la vista de escena:
  - ALT : Rota la cámara
  - CTRL+COMMAND: Zoom
  - SHIFT: incrementa la velocidad de movimiento
-  Permite mover el "game object" seleccionado dentro de la escena.
-  Permite la rotación del "game object" seleccionado.
-  Permite escalar cualquier eje (x,y,z) el "game object" seleccionado.
-  Permite escalar únicamente la vista frontal (ejes x, y) del "game object" seleccionado.
-  Permite ver la escena desde diferentes perspectivas (vista superior, frontal, derecha e inferior)

2- Game:

Desde este panel, podemos ejecutar el juego sin necesidad de salir del editor ni compilarlo, además, este dispone de los siguientes controles situados en la parte superior con las siguientes funcionalidades:

Imagen 70. Barra de control del panel Game

- Display 1: Permite elegir la visualización a mostrar del juego (en caso de que haya varias)
- Free Aspect: Cambia el formato del ancho y alto de la pantalla de juego.
- Maximize on Play: Una vez pulsado, permite maximizar a pantalla completa la vista del juego.
- Mute audio: Deshabilita/habilita el audio del juego.
- Gizmos: Activa y desactiva la visualización de Gizmos dentro del Panel game.
- Stats: Muestra las estadísticas de uso de los gráficos, audio y red del juego en tiempo real.

Para ejecutar el juego, detenerlo, pausarlo o avanzar "paso a paso" y poder visualizarlo en este panel, es necesario el uso de estos tres botones:

Imagen 71. Botones de control para ejecutar juego

3- Unity Asset Store:

Aparece como panel en versiones de Unity iguales o superiores a la 5. Permite a cualquier persona registrada al Unity Asset Store buscar, comprar, descargar e instalar cualquier Asset sin necesidad de salir del editor.

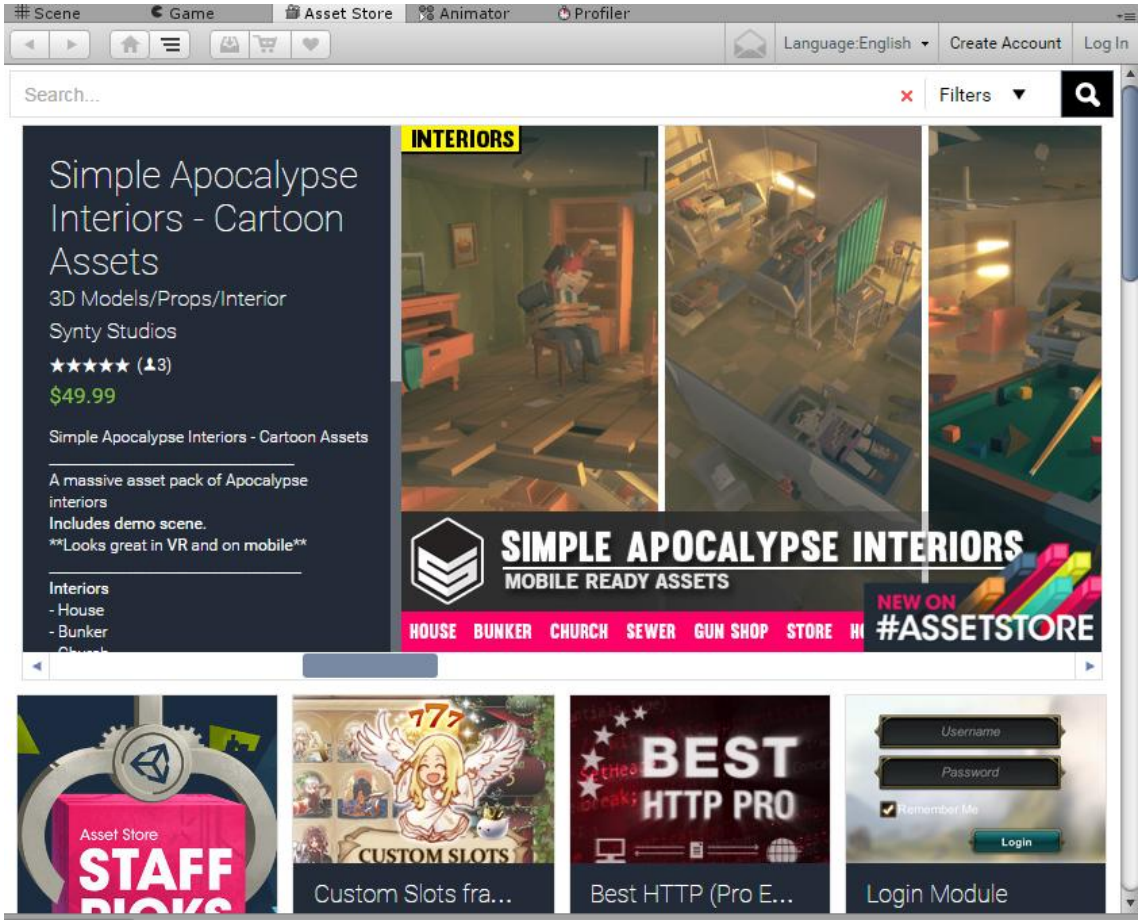


Imagen 72. Panel del Asset Store

4- Animator:

Mediante diagramas de flujo permite generar los diferentes estados de animación de un objeto o personaje de una manera simple y fácil, y además es capaz de interpolar los estados intermedios de dichas animaciones, es decir, es una interfaz para controlar la mecánica de animación del sistema.

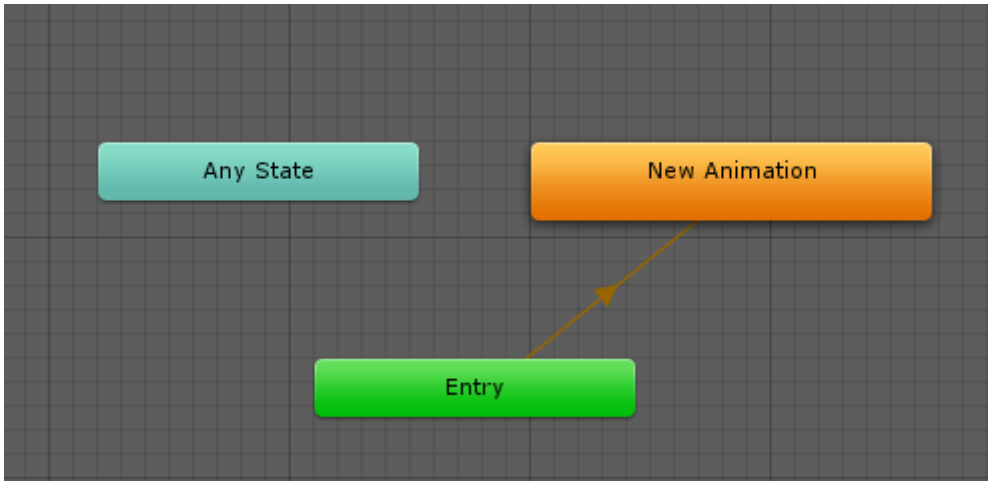


Imagen 73. Contenido de ejemplo del panel Animator

5- Animation:

Este se utiliza para crear y reproducir animaciones. Con él, puedes crear clips de animación de los diferentes "game objects" de un escenario, desplazarles, rotarles transformarles, cambiarles de color…

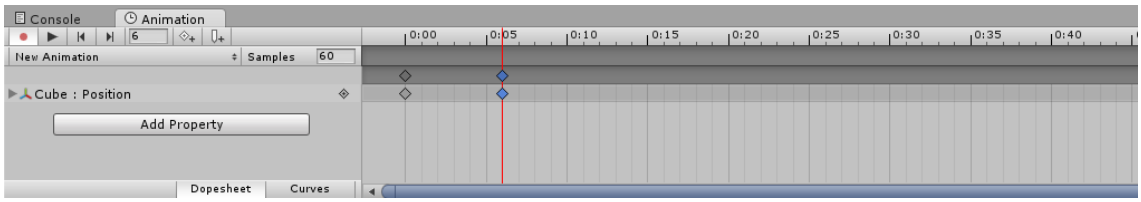


Imagen 74. Panel Animator

A través de los botones creación de clips de secuencia y grabación puedes establecer los estados iniciales y finales de la animación que desees desarrollar y ya Unity se encarga de extrapolar los estados intermedios.

6- Profiler:

Este panel nos ayuda a optimizar el juego. Nos reporta el uso de CPU, GPU y memoria en renderizado, cálculo de físicas, "scripts", sombras, transparencias... y demás lógica del juego.

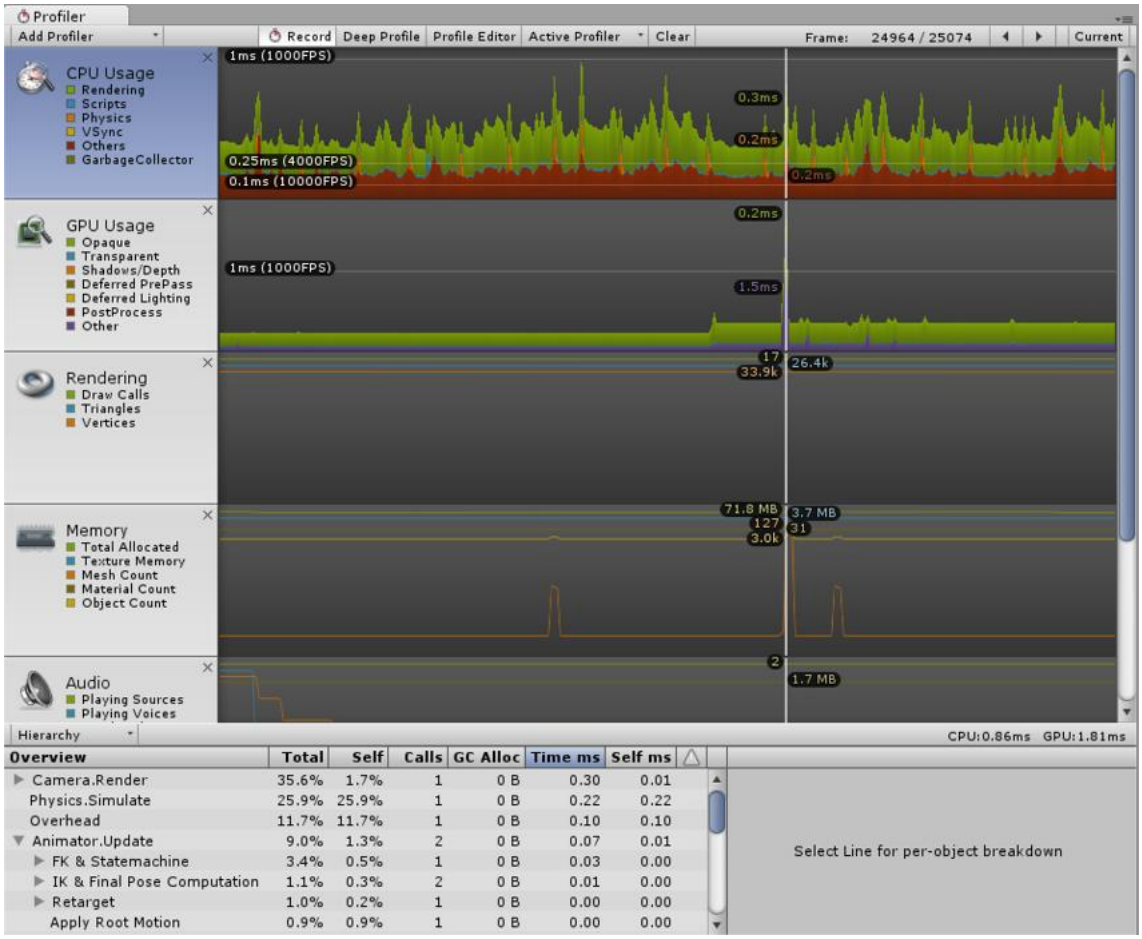


Imagen 75. Panel Profiler

Mientras jugamos de manera normal al juego, el "profiler" nos irá grabando los datos del rendimiento de este, para que una vez "finalizada la partida" podamos observar a través de una línea temporal los resultados.

7- Hierarchy:

Contiene una lista de todos los "game object" que hemos colocado en la escena actual del juego y nos muestra las relaciones jerárquicas que existen entre los diferentes objetos permitiéndonos además, copiar, eliminar y modificar los diferentes "game objects" junto con su jerarquía.

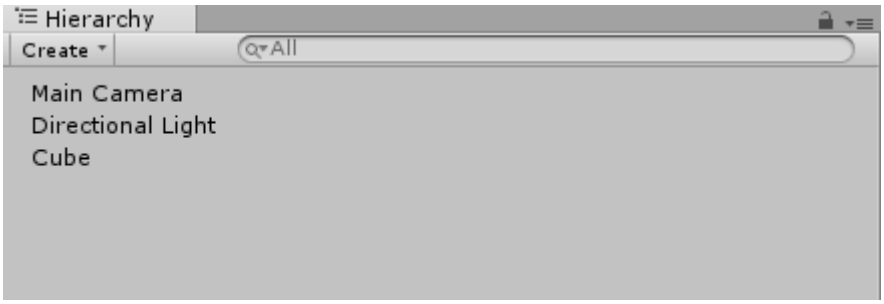


Imagen 76. Panel hierarchy

Al seleccionar un "game object" desde este panel, nos aparece sus propiedades y características en el panel de "Inspector" y además podemos modificar su ubicación desde el panel de "scene".

8- Inspector:

Se usa para ver y editar las propiedades de un "game object".

Cuando seleccionamos desde el "hierarchy" un "game object", el "inspector" nos muestra todos los componentes y materiales de este objeto y nos permite editarlos y añadir nuevos componentes al mismo.

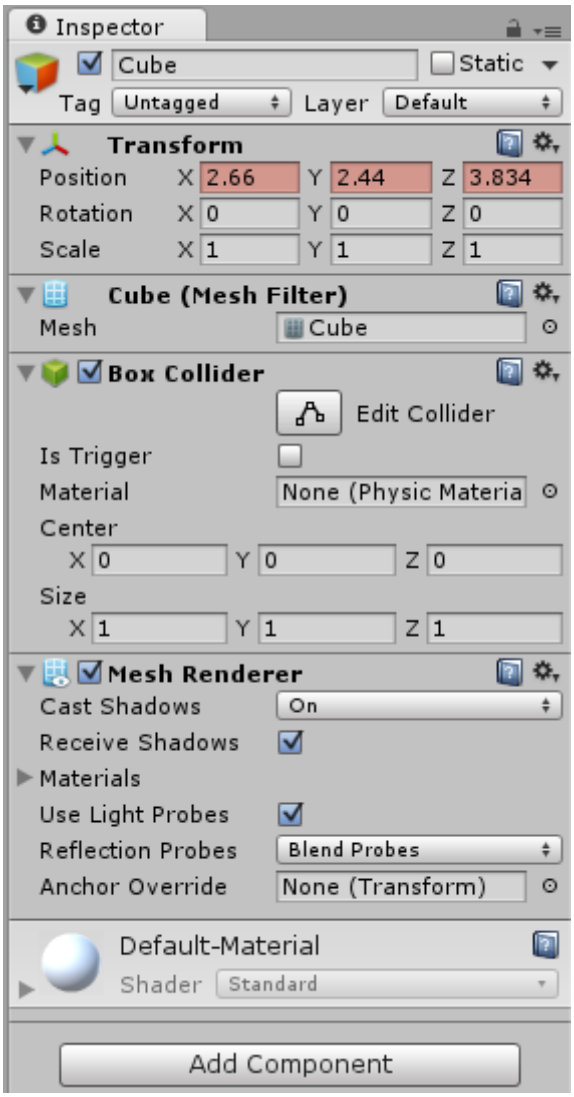


Imagen 77. Panel inspector

Desde este panel, podemos además añadir a los "game objects" código personalizado ("scripts") para que estos objetos reaccionen de la manera que deseemos dentro del juego.

9- Project:

Esta panel se divide a su vez en dos ventanas de navegación; La ventana izquierda, que muestra la estructura de carpetas del proyecto como una lista jerárquica, y la ventana del lado derecho que muestra el contenido seleccionado de la anterior junto con la dirección completa de la ruta en la que se encuentra dicho elemento seleccionado.

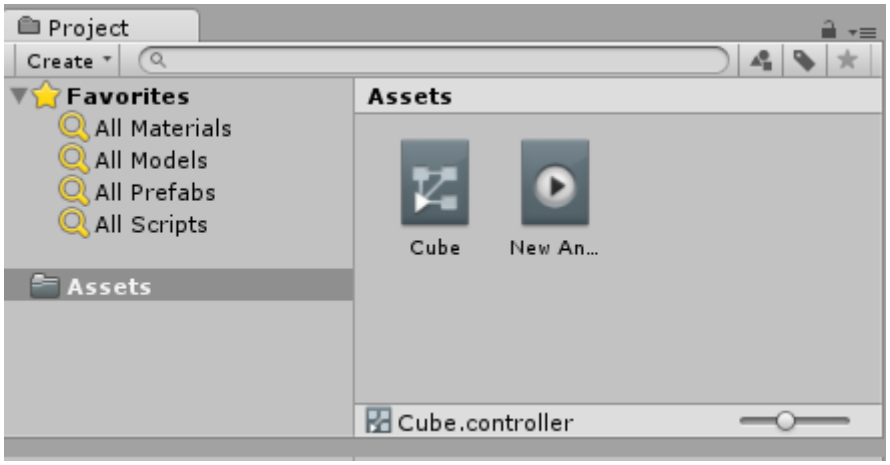


Imagen 78. Panel inspector

Los "assets" se muestran en el panel derecho con iconos que nos indican su tipo (script, materia, subcarpeta…) y pueden ser redimensionados de tamaño usando el deslizador que se sitúa en la parte inferior izquierda del panel.

En la esquina superior de la ventana izquierda, existe una sección de favoritos. Aquí podemos colocar los elementos que usemos de manera más frecuente para tener un acceso rápido. Para ello, simplemente arrastramos un elemento a la carpeta correspondiente de favoritos para así generar su acceso rápido.

El navegador dispone además de un buscador. Es especialmente útil para ubicar "assets" y filtrar contenidos de acuerdo al tipo y las etiquetas del asset establecidas en el "inspector".

Los filtros de búsqueda funcionan a través de los siguientes términos:

- "t:" filtra por el tipo de "asset".
- "l:" filtra por etiqueta.

Finalmente si se selecciona en la ruta del navegador el "Asset Store", se pueden realizar búsquedas del Store de Unity y añadirlas al nuestro proyecto de manera directa.

10- Console:

Muestra errores, advertencias y otros mensajes que nos ayudan a solucionar fallos y debugear nuestro juego (además podemos añadir nuestros propios mensajes de debug por medio de la función "Debug.Log").

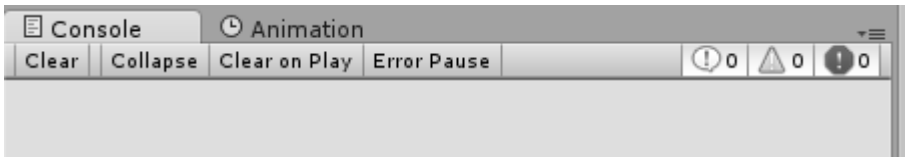


Imagen 79. Panel console

La barra de herramientas de este panel contiene una serie de controles que afectan a cómo se muestran los mensajes:

- Clear: elimina cualquier mensaje generado desde el código y mantiene únicamente los errores de compilación.
- Clear on Play: una vez seleccionado, limpia automáticamente los mensajes antiguos antes de correr el juego.

- Collapse: hace que se muestre solamente la instancia de un mensaje de error que sigue recurriendo.
- Error Pause: pausa la generación de mensajes de error y congela la reproducción del juego en ese punto específico.

9.1.4 Creando un nuevo proyecto:

La creación de un nuevo proyecto se puede realizar de dos maneras, abriendo Unity 3D y seleccionando "New", o una vez iniciado seleccionando File→New Project.

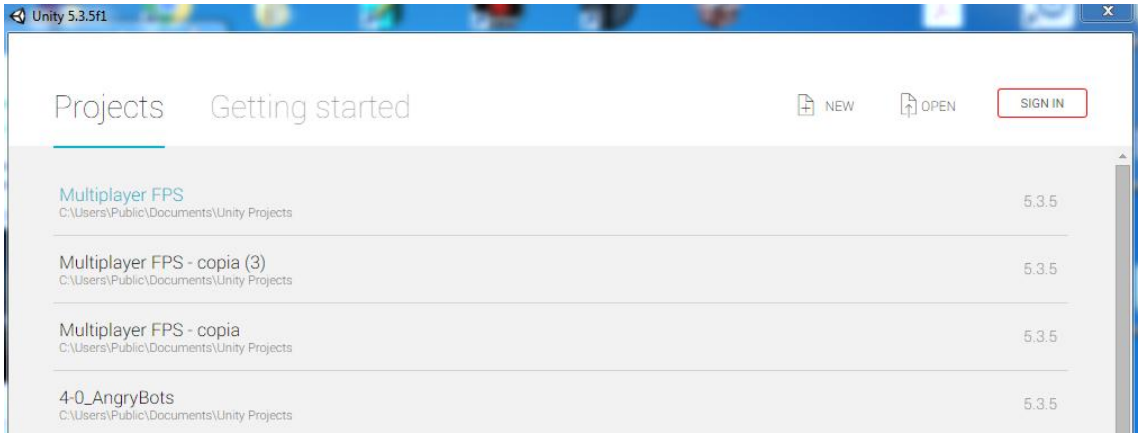


Imagen 80. Ventana de inicio de Unity: Permite abrir o crear nuevos Proyectos.

Una vez realizado esto, nos aparecerá una ventana de diálogo bajo el nombre de "Asset packages". Esta nos facilita paquetes predefinidos por Unity 3D o que hayamos comprado en el Asset Store los cuales poseen "Assets" que no pueden servir para facilitarnos la creación y desarrollo de nuestro videojuego.

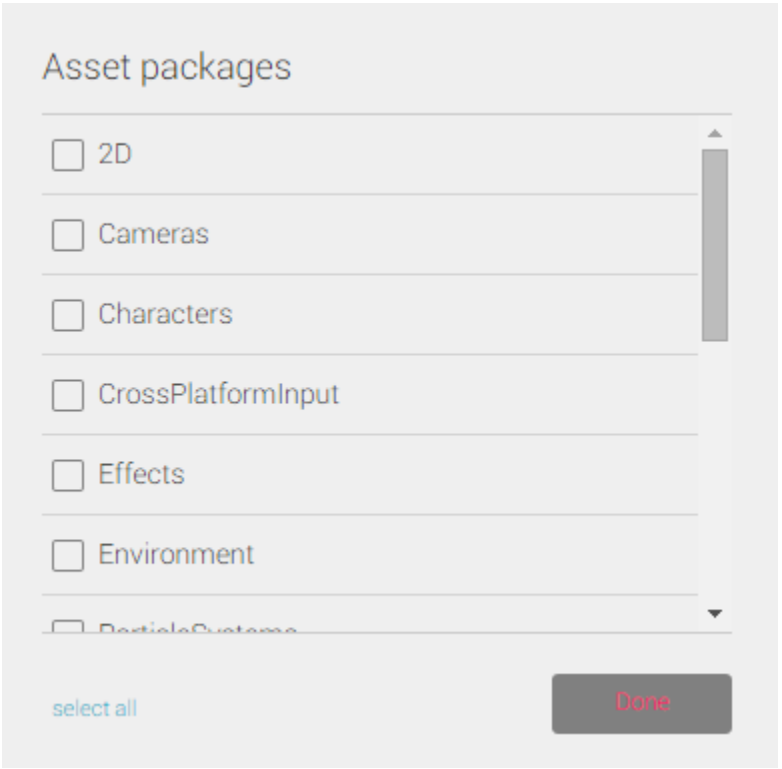


Imagen 81. Ventana de Asset packages con los paquetes predeterminados.

Algunos de los paquetes más importantes que trae Unity 3D por defecto son los siguientes:

- 2D: Colección de sprites, ejemplos y resources para ayudar a construir juegos en dos dimensiones.
- Characters: Prefabs de personajes y scripts para poder moverles a través del mapa.
- Effects: Efectos de borrosidad en la cámara, corrección del color, visión cromática…
- Environment: texturas de terrenos, skyboxes para creación de cielos, agua, arboles…
- ParticlesSystem: Efectos de partículas como humo, fuego o explosiones.
- Vehicles: Vehículos como coches y aviones, junto con scripts para poder moverles en el mapa.

9.1.5 Crear una nueva escena:

Unity 3D funciona a través de lo que denomina "scene" o escenas. Tal y como denota la palabra, son escenarios donde podemos introducimos desde menús a mapas completos.

Según Unity, estos escenarios tienen un entorno 3D infino por lo que podría crear un juego entero usando únicamente una única escena.

Para crear nuevas escenas, basta con ir a File→New Scene y para guardarla escena creada, File→Save Scene. Para cargar una escena con la que queramos trabajar solo tenemos que seleccionar el archivo de escena que nos aparece visible en el panel "project".

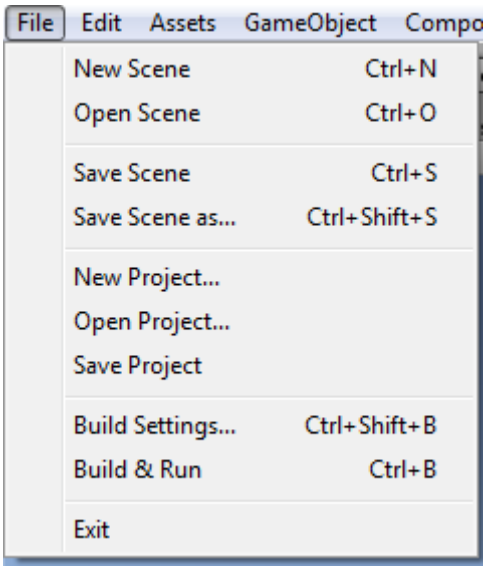


Imagen 82. Menú "File" de Unity 3D

9.1.6 Creación de objetos:

Unity 3D puede trabajar con modelos 3D de cualquier forma y tamaño aunque hayan sido creados con diferentes software de modelado. Para importarlos a Unity, basta con presionar el botón derecho del ratón dentro del panel de "Project" y seleccionar "Import new Asset".

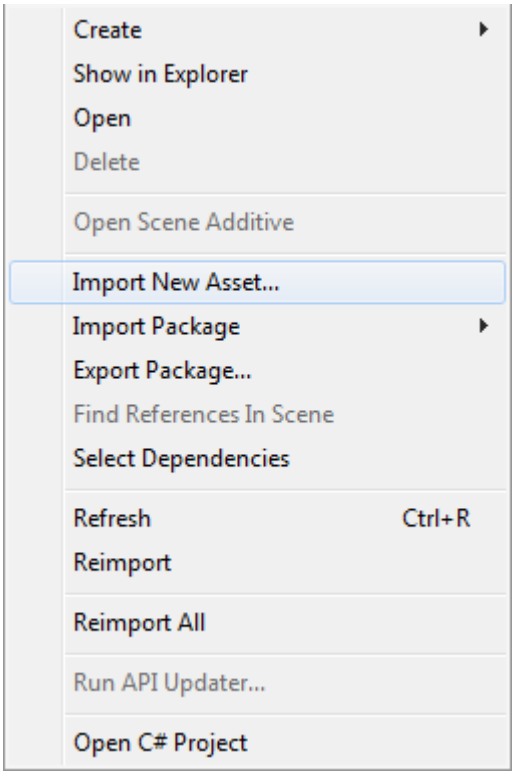


Imagen 83. Menú de opciones del panel "Project"

Por otro lado, existen ciertos tipos de objetos básicos que pueden ser creados directamente desde Unity como son el cubo, esfera, capsula, cilindro, plano y cuadrángulo.

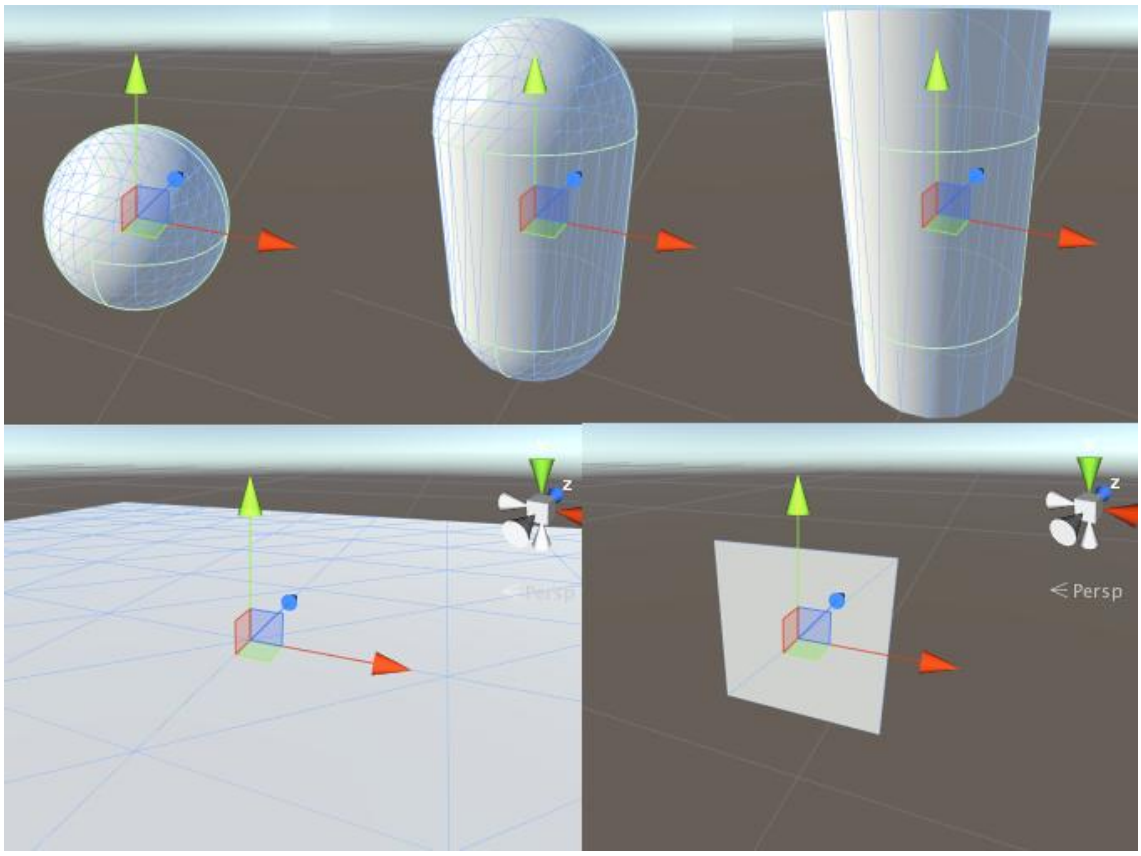


Imagen 84. Objetos básicos de Unity

Aparte de estos, Unity considera también como "objetos" los siguientes componentes:

- Sprite (objeto bidimensional para juegos 2D)
- Light (para establecer los puntos de iluminación del juego)
- Audio (son objetos invisibles los cuales emiten una pista de audio)
- UI (componentes de menús como botones, imágenes, canvas…)
- Particle System (sistema de partículas, para realización de efectos como humo, fuego, movimiento de agua…)
- Camara (a través de la cual el usuario visualiza el mapa del juego)

Todos estos objetos se pueden escalar, posicionar, añadirles scripts y otros componentes a través del panel hierarchy, y a excepción de la cámara, el resto de objetos se les puede añadir texturas y shaders.

9.1.6.1 Manipulación de objetos

Una vez creados o importado un objeto, tras insertarlo en la "scene" del juego, este podemos modificar su tamaño, posición y forma para así conseguir el resultado que deseado.

Para ello podemos realizarlo de dos maneras, una vez que seleccionemos el objeto a manipular, podemos modificarle mediante estos botones de control y mediante el inspector de dicho objeto:

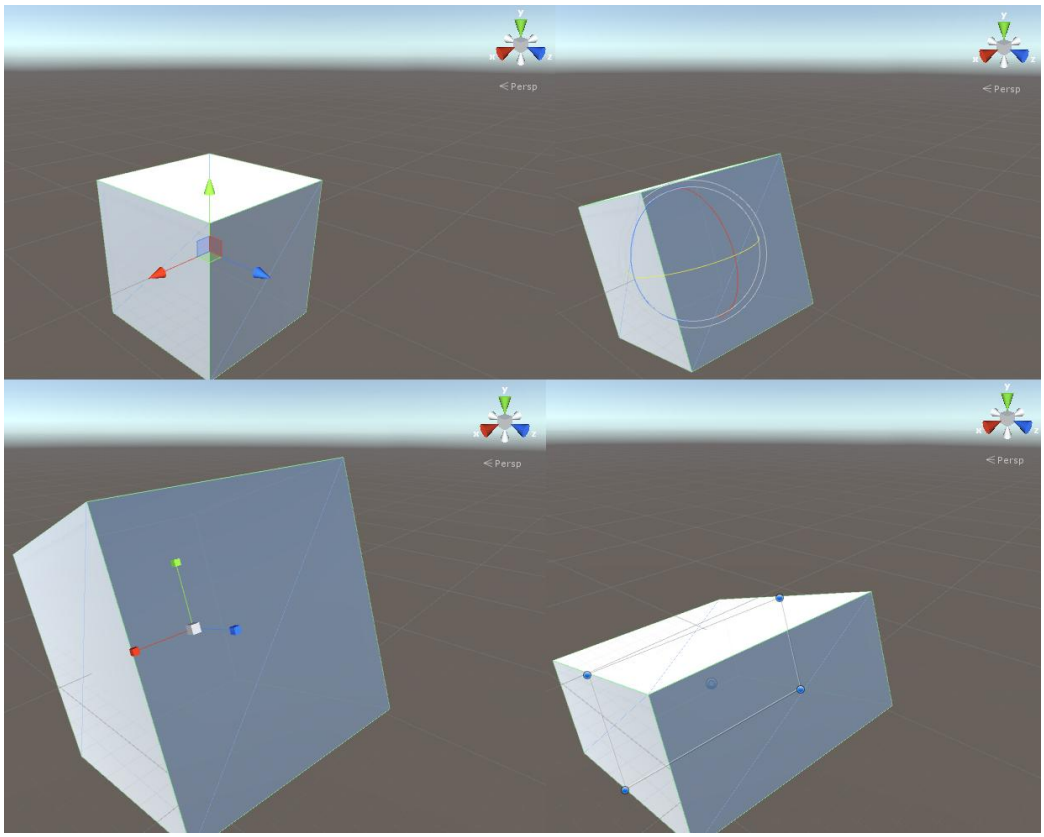


Imagen 85. Modificación de la posición, rotación y escalado de un objeto.

Como se puede observar en la imagen anterior, mediante los botones de control del panel de scene, podemos manipular fácilmente la posición, rotación y escalado de un objeto. Y una vez

ubicado y transformado, podemos manipular estos mismos atributos con mayor precisión desde el inspector de dicho objeto.

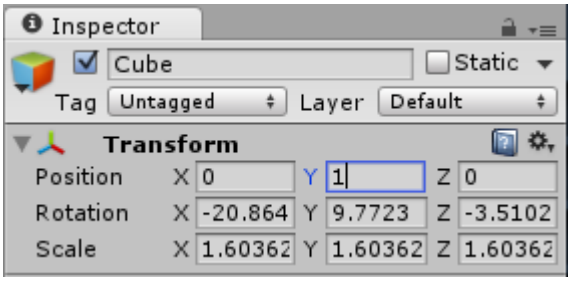


Imagen 86. Inspector del objeto cube

El componente "transform" del objeto cube nos permite cambiar la posición, rotación y escala del mismo a través de la introducción de valores numéricos.

9.1.6.2 Uso de Scripts en objetos

Si ahora mismo colocamos diferentes objetos delante de una cámara y ejecutamos el juego, podremos observar dichos objetos. Sin embargo, no hay ningún movimiento ni interacción ya que no hemos establecido lógica alguno a dichos objetos.

Para ello debemos crear un script. Situamos el ratón sobre el panel de Project y pulsamos el botón derecho del ratón, seleccionamos Create→Script (en C# o JavaScript), le damos un nombre descriptivo y lo abrimos.

Al abrirlo, Unity, cargará el entorno de Visual Studio o el Mono Develop, según tengamos configurado por defecto. A continuación nos aparecerá nuestro script con las funciones por defecto de Start() y Update():

- Start(): Se ejecuta al iniciar el programa o cuando se crea el objeto que contiene el script.
- Update(): Es invocada cada vez que se refresca el juego, entre 30 a 60 veces por segundo en función de la computadora en la que se ejecute el juego.

Si queremos por ejemplo que un objeto se mueva al pulsar una tecla (digamos la tecla "space), habría que hacer lo siguiente:

1) Crear un script.
2) Añadir el script al objeto que queremos que se mueva. Para ello, seleccionamos el objeto y una vez seleccionado, desde el hierarchy pulsamos sobre "Add Component" y añadimos dicho script.
3) Introducimos el siguiente código en la función update() de dicho script:

```
If(Input.GetButton("Space"))
{
Transform.Translate(0,0,1*Time.deltaTime);
}
```

De esta manera, desplazamos el objeto en el eje Z a una velocidad constante de "1" cada vez que mantenemos pulsada la tecla Space y gracias a Time.deltaTime conocemos el tiempo que ha transcurrido en la tasa de refresco desde la última vez que se ejecutó update() con lo que nos asegurarnos de que el objeto realiza un desplazamiento uniforme.

No todos los objetos deben por regla general existir ya introducidos dentro del escenario. Personajes, tesoros y otros objetos, pueden ser creados a la hora de iniciar el juego o durante el mismo desarrollo de este. Para ello Unity dispone de las funciones Instantiate() y Destroy() que nos permiten copiar y destruir objetos respectivamente:

```
void CmdDisparar()
  {
    //Crea las balas a partir del prefab "bullet"
    GameObject bullet = (GameObject)Instantiate(bulletPrefab, bulletSpawn.position, bulletSpawn.rotation);
    //Añade velocidad a bullet
    bullet.GetComponent<Rigidbody>().velocity = bullet.transform.forward * 6.0f;
    //Destruir bullet tras 2 seg
    Destroy(bullet, 2.0f);
  }
```

Con esta función, podemos crear una copia del objeto "bullet" y ubicarla en la posición que deseemos, que avance hacia adelante cierta velocidad y que al cabo de 2 segundos, dicho objeto desaparezca.

Como se puede observar en el anterior ejemplo, el objeto "bulletPrefact" no contiene dicho script, sino al revés el script quien "contiene" a dicho objeto. Unity, posee dos maneras de asignar un objeto ya existente a una variable:

- La primera consiste en crear una variable de tipo GameObject que sea pública o serializable, de esta manera al guardar el script, podremos ver en el inspector estas variables y asignarles dicho objeto a través de la interfaz visual de Unity.

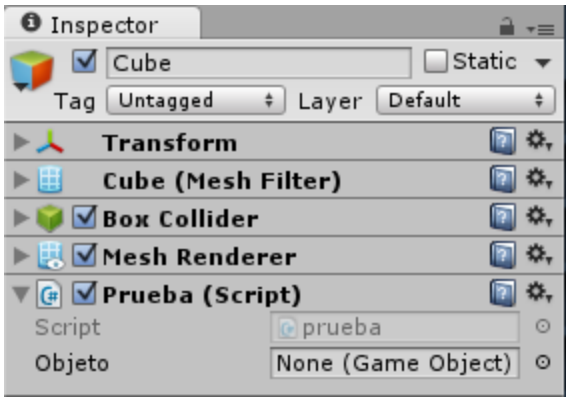


Imagen 87. Panel Inspector con un componente "script"

- La segunda consiste en usar la función "GameObject.Find" para localizar el objeto:

```
GameObject objeto;
```

```
objeto=GameObject.Find("nombre");
```

Como ya se ha mencionado previamente, Unity 3D soporta varios lenguajes de programación. Pese a ello, existen un gran número de palabras y funciones reservadas propias de Unity, por lo que es recomendable echarle un vistazo a su documentación en la siguiente página web para consulta y creación de scripts avanzados:

```
http://docs.unity3d.com/Manual/index.html
```

9.1.7 Sistema de Física

Unity utiliza la API PhysX de Nvidia como motor de física y gracias a su interfaz de fácil manejo, nos permite utilizar todo lo relacionado con el motor de física sin tener que trabajar en exceso con la programación.

En concreto la versión 5 de Unity (que es la que usamos para este proyecto) utiliza PhyX3.3 (a diferencia de las versiones 4 y anteriores que usaban PhyX2).

Debido a que PhysX3 ha sido rediseñada completamente desde cero, es posible que muchos proyectos realizados con versiones antiguas de Unity 3D no sean del todo compatibles o den fallos a la hora de ejecutarlos.

9.1.7.1 "Collider", "Rigid body", "Tigger" y "Ray cast"

Casi todos los juegos a día de hoy requieren de la simulación de colisiones. Realizar estas colisiones de forma precisa y sobre superficies irregulares requiere michos cálculos y muchas ocasiones un tiempo de ejecución que puede ralentizar el juego.

Por eso, en vez de utilizar la geometría real de los objetos a colisionar, se hacen uso de geometrías sencillas para obtener una verificación rápida y simple. Unity hace uso de cuatro tipos diferentes de geometría para calcular las colisiones; con forma de caja, de cápsula, de esfera y malla.

Tipos de Colliders de Unity:

1. Box collider:
   La mayoría de objetos creados en Unity, tienen por defecto este componente incluido. El box collider es tal y como denota su nombre una caja la cual envuelve completamente el objeto que posee este componente y es invisible de cara a la visión del juego. Esta se puede ajustar sus dimensiones de manera manual para que encaje lo máximo posible al objeto que envuelve.
2. Capsule collider:
   Se utiliza principalmente para los personajes de videojuego. Es igual al box collider con la diferencia que esta tiene forma de capsula en vez de caja.
3. Sphere collider:
   Tiene forma de esfera y su cálculo de colisión es el más rápido y sencillo de realizar. Desgraciadamente no es apto para objetos delgados ya que la superficie del collider con la del objeto dista demasiado, dando lugar a la detección de falsas colisiones entre objetos.
4. Mesh collider:

Es el más complejo, en vez de usar una geometría sencilla como los anteriores, utiliza una copia de la malla del objeto que envuelve, aumentando por tanto la complejidad del cálculo de la colisión en base a la forma que tenga el objeto.

Aparte de los colliders, es necesario otro componente para poder hacer frente a una collisión: los rigid body.

Los rigid body son componentes que permiten añadir a un objeto física tal como la gravedad, inercia, rigidez (desde solido a intangible), masa… De esta manera podemos hacer que por ejemplo al lanzar una piedra contra una pared, el juego no solo calcule la colisión sino que aplique la física necesaria para hacer que dicha piedra rebote o atraviese la pared según la fuerza aplicada y velocidad de la piedra.

Hasta aquí hemos explicado como el motor de físicas, calcula las colisiones y simula el comportamiento de los objetos cuando son sometidos a fuerzas externas y de choque como por ejemplo la gravedad, pero nos falta lo más importante, ¿Cómo detectamos estas colisiones?

Para la detección de estas, Unity nos da dos métodos, mediante triggers y a través de un ray cast.

El sistema de scripting puede detectar cuando sucede una colisión e instanciar acciones a través de la función OnCollisionEnter, o usar simplemente el motor de física para detectar cuando un collider entra en el espacio de otro sin necesidad de crear una colisión (es decir que un objecto pasa a través de otro ya que no disponen de un rigid body) a través de la función OnTriggerEnter.

Por otro lado el scripting a través de la función Raycast() consiste en disparar un "rayo" invisible como si se tratase de un puntero laser en cierta dirección. Si este "rayo" se le interponga o choque con un objeto en su dirección, devolverá el booleano "true" y en caso de que no choque con nada "false".

9.1.7.2 Joints

En Unity, podemos unir un objeto a otro o con un punto fijo en el espacio a través del componente "Joint" (articulación). De esta manera podemos mover dicho objeto obligándole a cumplir ciertas restricciones de movimiento en torno al objeto o punto al que le hemos unido.

Los Joints también permiten la activación de ciertos efectos, como por ejemplo el de romperse la articulación cuando la fuerza aplicada a este sobrepasa cierto umbral (como se da en el juego de World of Goo).

9.1.8 Manejo de la Cámara

La cámara en Unity, es un objeto que nos permite observar el escenario del juego. Se puede integrar en un personaje para que se mueva acorde a sus movimientos, ser fija o incluso omitir diferentes componentes de cara a la visualización del escenario y se podemos poseer un número ilimitado de cámaras en escena.

Podemos por tanto, considerar a este objeto un componente esencial ya que es el encargado de "mostrarnos" el mundo virtual del juego que creemos. Debido a esto Unity lo ha provisto de múltiples propiedades como modos de vista, selección de capas a mostrar, campo de visión y otras muchas para de esta manera reducir la carga de trabajo que nos requeriría el realizar estas

opciones a través de programación de scripts mediante la simple configuración de unos parámetros.

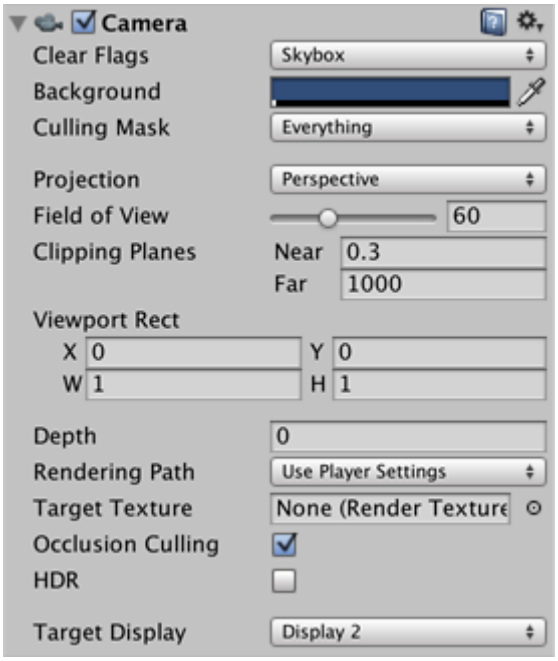


Imagen 88. Inspector de Camera

Propiedades de la "cámara":

- Clear Flags: Determina que partes de la pantalla deben visualizarse. Se utiliza por ejemplo en juegos tipo FPS donde el jugador ve solo los brazos de su personaje, mientras que el resto ven el muñeco entero.
- Background: Color de fondo del escenario.
- Culling Mask: Establece que capas de los objetos deben de ser renderizadas.
- Projection: Activa o desactiva la capacidad de la cámara para simular perspectiva.
- Size (si en projection se selecciona el modo orthographic): Modifica el tamaño de la vista gráfica de la cámara.
- Field of view (si en projection se selecciona perspective): Cambia el ancho del ángulo de vista de la cámara.
- Clipping Planes: Establece las distancias para empezar y terminar de renderizar según lo lejos y cerca que estén los objetos de la cámara.
- Viewport Rect: Establece los dos puntos iniciales donde se situa la cámara, junto con el ancho y alto que abarcará.
- Depth: En caso de existir varias cámaras superpuestas, establece el orden de "dibujo" en función de la profundidad que ocupa cada cámara.
- Rendering Path: Muestra diferentes opciones para definir los métodos de renderizado que utilizará la cámara.
- Target Texture: activar esta opción deshabilita la capacidad de la cámara para renderizar por pantalla todo los objetos que no diponga de la textura referenciada en esta opción.
- Occlusion Culling: Deshabilita el renderizado de aquellos objetos que no son visualizados por la cámara.
- HDR: Activa el renderizado High Dynamic Range para esta cámara.

- Target Display: En caso de disponer varias cámaras, para poder asignar un monitor diferente a visualizar desde la vista de scene.

9.1.9 Manejo de materiales y sprites

Por regla general un objeto 3D, de cara a los videojuegos u otras aplicaciones, su forma es una aproximación poco detallista del objeto que representa, siendo los detalles finos de esta suministrados por las texturas. Una textura es un bitmap que se aplica sobre la superficie de la malla del objeto 3d. Podemos pensar de las texturas como si se tratasen de pegatinas con una imagen impresa adheridas a un objeto.

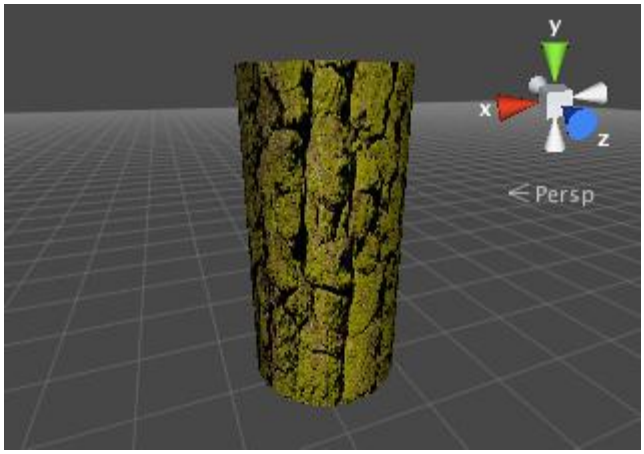


Imagen 89. Cilindro con textura de corteza

9.1.9.1 Aplicando texturas en modelos 3D

Las texturas en Unity se aplican a los objetos a través de los materiales (Material). Para crear un nuevo material desde el panel de Project pulsamos botón derecho→ Create→ Material y una vez creado, arrastramos el material sobre la superficie del objeto a aplicar.

Al seleccionar el inspector del material, podemos acceder a sus parámetros desde dicho panel y añadir diferentes tipos de texturas.

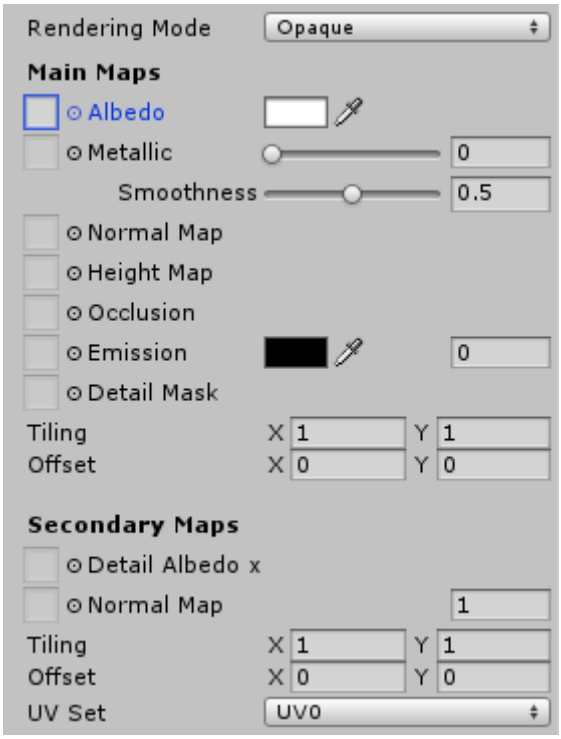


Imagen 90. Inspector de Material

Como se puede observar en el inspector de un material, Unity soporta texturas PBR [16] (Physically based rendering), las cuales, mediante el uso de modelos matemáticos, aproxima los diferentes comportamientos de la iluminación (reflexión y refracción). De esta manera, añadiendo las siguientes texturas, podemos hacer que el objeto 3D apenas sea imperceptible de si es o no real.

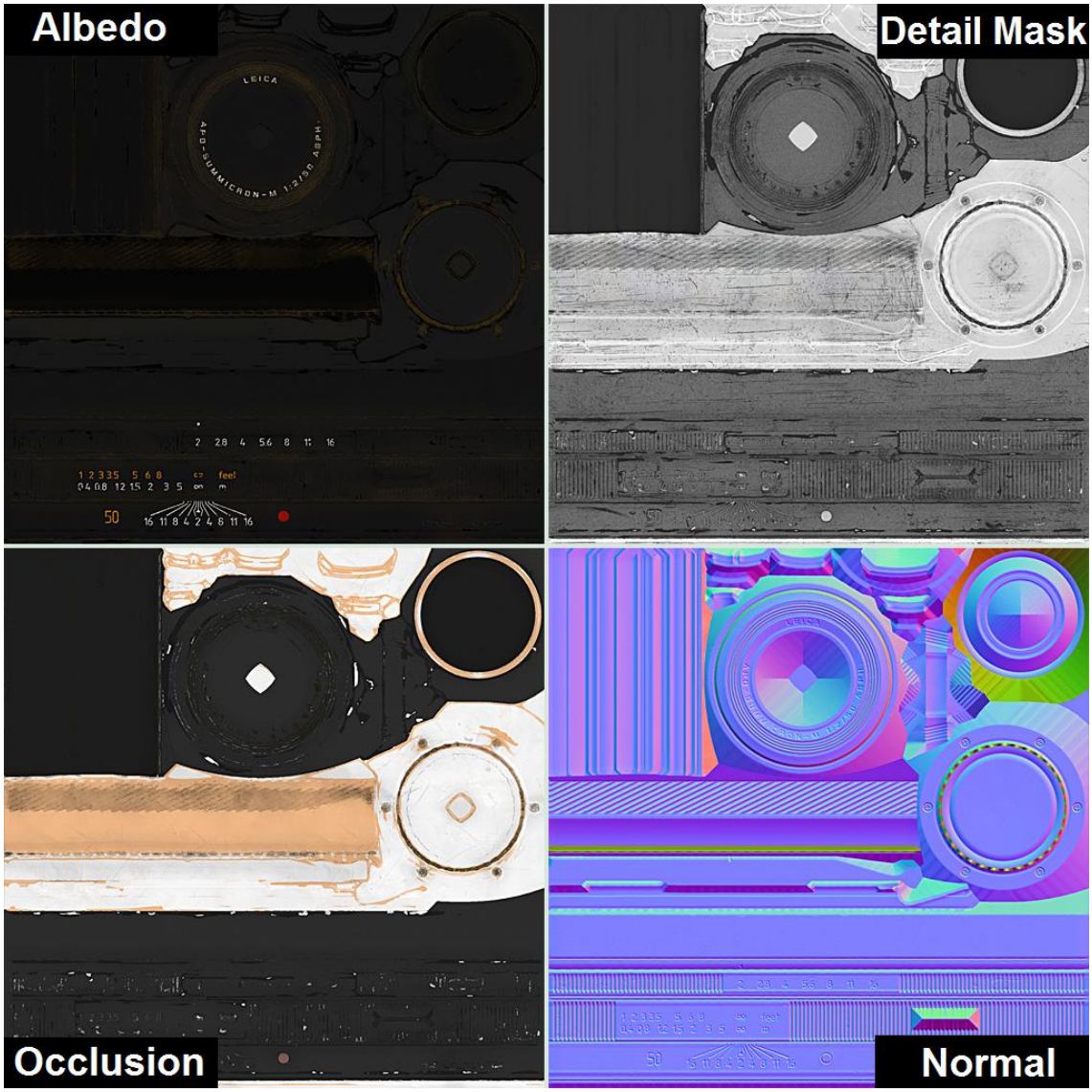


Imagen 91. Texturas PBR

Imagen 92. Objeto 3D con texturas PBR

Podemos hacer uso desde texturas básicas normales, como la del ejemplo anterior del cilindro con la corteza de árbol, a texturas PBR las cuales consiguen un efecto hiperrealista sin apenas de GPU.

9.1.9.2 Uso de sprites

En caso de crear un nuevo proyecto en Unity para juegos 2D, en vez de materiales, se hace uso de sprites, texturas aplicadas a objetos planos.

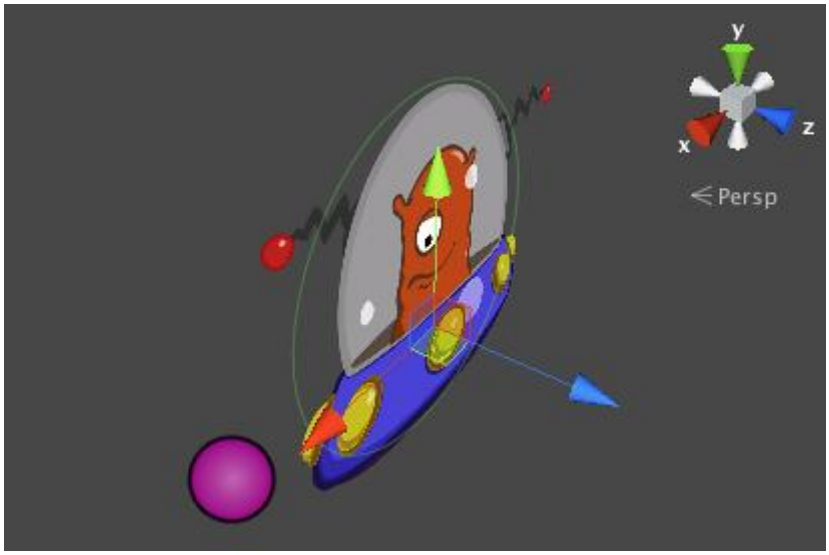


Imagen 93. Sprite visto en perspectiva

9.1.10 Editor de terrenos

Unity posee desde la versión 2 un completo editor de terrenos muy potente que permite la creación de estos de manera rápida y simple. Además, incluye prefabs de árboles, plantas y agua y texturas básicas como las de hierba y piedra.

Imagen 94. Ejemplo de terreno creado con Unity

A través de un conjunto de herramientas, presentadas bajo la forma de "pinceles", podemos modelar el terreno bajando y subiendo diferentes partes de la superficie además de usar también otro tipo de pinceles para aplicar texturas sobre las diferentes superficies por las que pasemos los pasemos.

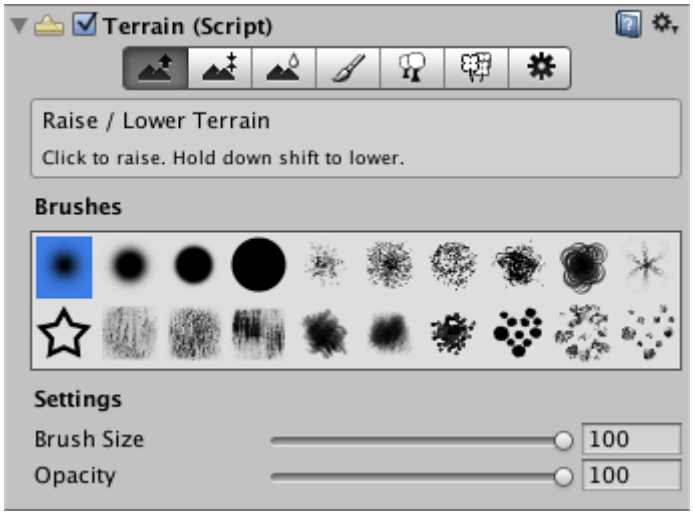


Imagen 95. Herramientas de edición de terreno

9.1.10.1 Crear terreno y uso de atajos de teclado

El primer paso es agregar un objeto tipo "Terrain" al escenario sobre el que estemos trabajando para poder empezar a trabajar sobre este. Para ello, pulsamos sobre GameObject→ 3D OBject→ Terrain y de esta manera obtendremos un asset básico de terreno.

A continuación y no lo hemos añadido al crear nuevo proyecto, pulsamos botón derecho sobre el panel de Project y damos sobre Import Package→Environment y seleccionamos los diferentes assets por defecto que nos ofrece Unity relacionados con el terreno como texturas, agua, y árboles.

Antes de continuar, conviene aprenderse los atajos de teclado para la edición de terrenos ya nos ahorran tiempo de cara a al correcto uso de las herramientas y selección de las mismas:

- Al pulsar la techa Shift junto con una tecla comprendida desde la Q a la Y, se selecciona la herramienta de terrenos correspondiente a dicho atajo (por ej, Shift+Q selecciona la herramienta Raise/Lower)
- La tecla coma "," junto con el punto ".", recorren los pinceles disponibles.
- Shift+"," y shift+"." Recorren los objectos disponibles para árboles y texturas.

Además el atajo F de Unity funciona de forma ligeramente diferente para los terrenos. Normalmente Unity enmarca un objeto seleccionado cuando el ratón esta sobre el scene view. Sin embargo, ya que los terrenos son muy grandes, pulsando F este centrará el scene view en el área de terreno sobre el que esté situado en ese momento el ratón.

9.1.10.2 Configuración básica de un terreno

Una vez que tenemos un objeto terreno en el escenario, el primer paso es definir sus propiedades básicas como el tamaño y el nivel de detalle que debe poseer.

Para realizarlo, seleccionamos el objeto terreno desde el panel de vista jerárquica para que de esta manera obtener sus diferentes componentes en el panel inspector.

Entre los diferentes componentes, nos centramos en el componente "Terrain", seleccionamos el botón de configuración, y entre las diferentes opciones de configuración, nos entramos en la de "Resolution":

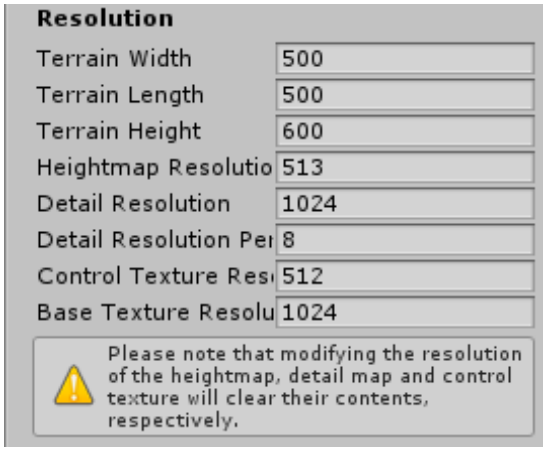


Imagen 96. Terrain Resolution

- Terrain Width: Establece el ancho total en metros que tendrá el terreno.
- Terrain Legth: Longitud total en metros del terreno.
- Terrain Height: Altura máxima en metros que podría alcanzar el terreno al editarle.
- Heightmap Resolution: Resolución del HeightMap (cada punto del terreno se representa como un valor y se almacena en una imagen escalada a gris conocida como heightMap)
- Detail Resolution: Cuanta más resolución más detallista a la hora de dibujar y alterar la superficie del terreno.
- Control Texture Resolution: Resolución de las texturas aplicadas sobre el terreno.
- Base Texture Resolution: Resolución de la textura primaria del terreno.

9.1.10.3 Herramientas de edición de la Altura

Las tres primeras herramientas de la barra del componente Terrain sirven para "pintar" cambios de altura sobre el terreno:

Imagen 97. Herramientas de edición de altura del terreno

De izquierda a derecha, el primer botón activa la herramienta Raise/Lower Height. Cuando se utiliza un pincel con este botón activado, la altura se incrementa a medida que se desplaza el ratón por encima del terreno. Al igual que las herramientas de aerógrafo de los editores de imagen, la altura se acumulará si se mantiene el ratón en su lugar. Si se mantiene pulsada la tecla Shift, la altura se reducirá. Según el pincel que se elija, se pueden obtener una gran variedad de efectos.

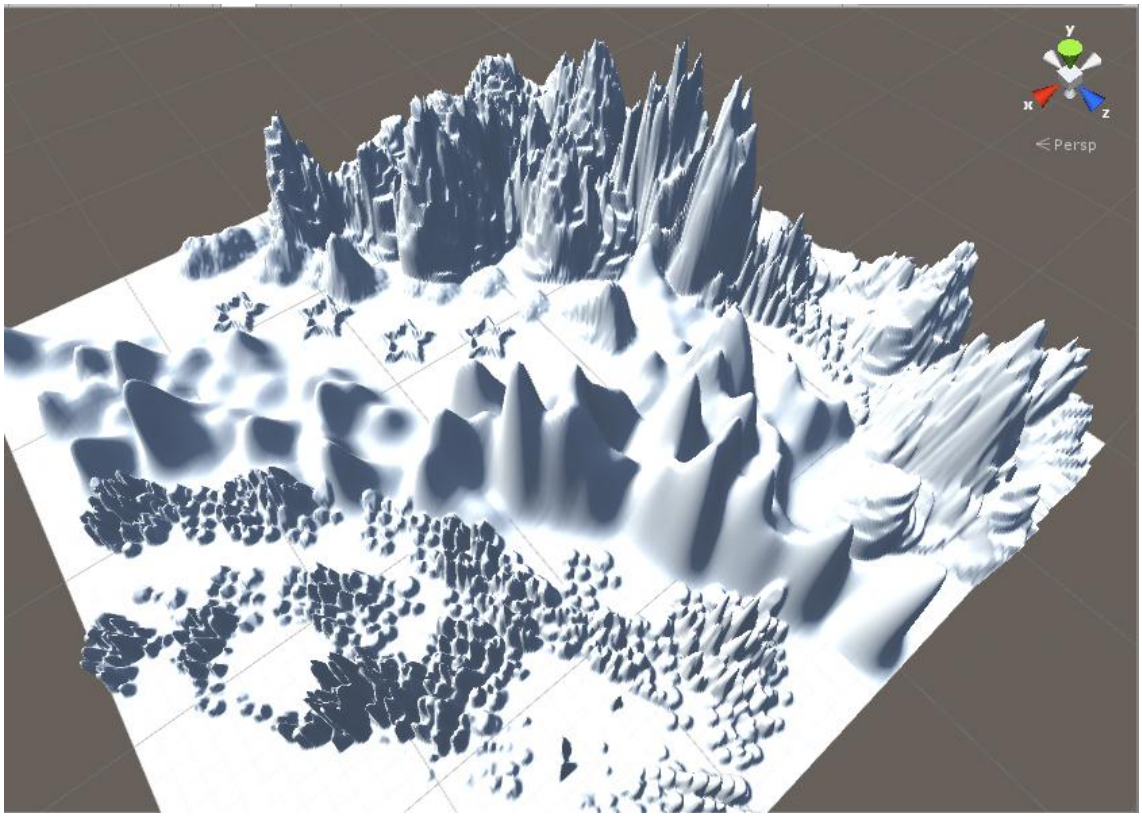


Imagen 98. Terreno de prueba realizado con Raise/Lower Height

La segunda herramienta es Paint Height. Es similar a la anterior, pero tiene la propiedad adicional de establecer la altura objetivo. Cuando se pinta con esta herramienta seleccionada, el terreno se reducirá en zonas por encima de la altura establecida y aumentará en las áreas que estén por debajo de esa altura. Para establecer la altura se puede realizar de dos maneras, la primera es a través del parámetro Height y la segunda es presionando la tecla Shift y hacer click sobre el terreno para tomar una muestra de la altura de este según donde se posicione el ratón.

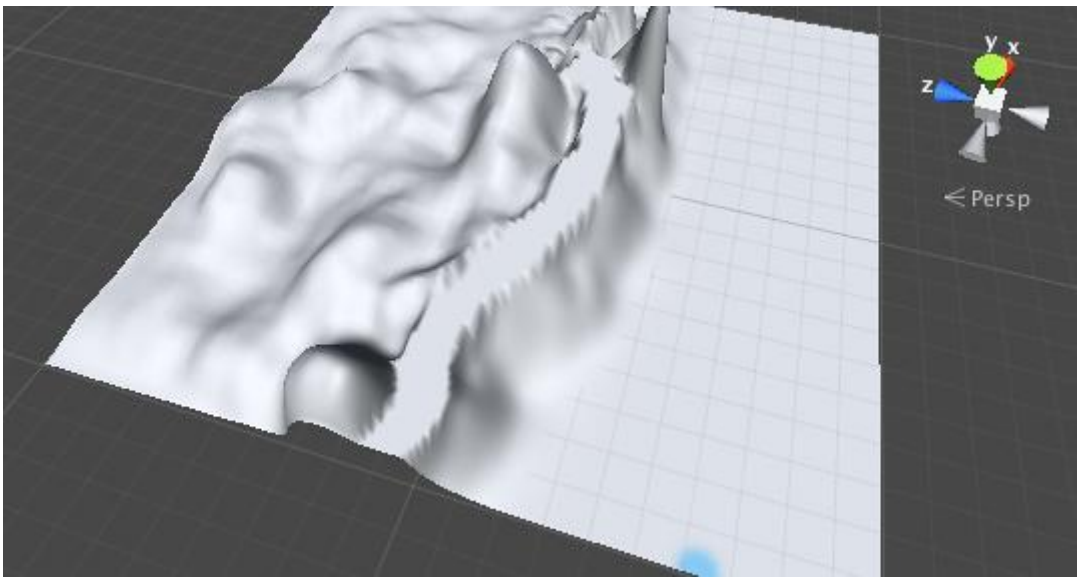


Imagen 99. Terreno de prueba haciendo uso de la herramienta Paint Height

La tercera herramienta, Smooth Height promedia la altura del terreno en base a las áreas cercanas. De esta manera, podemos suavizar el paisaje y reducir la aparición de cambios abruptos en el terreno.

Otra alternativa para la edición del terreno es la importación de un heightmap, si hacemos click sobre el botón de settings, en su menú podemos encontrar las opciones de importar y exportar heightmaps y así cargar o guardar lo datos geográficos del terreno.

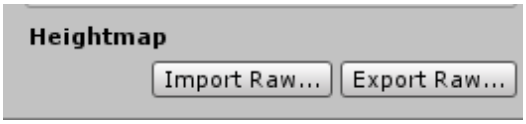


Imagen 100. Importación y exportación de heightmap

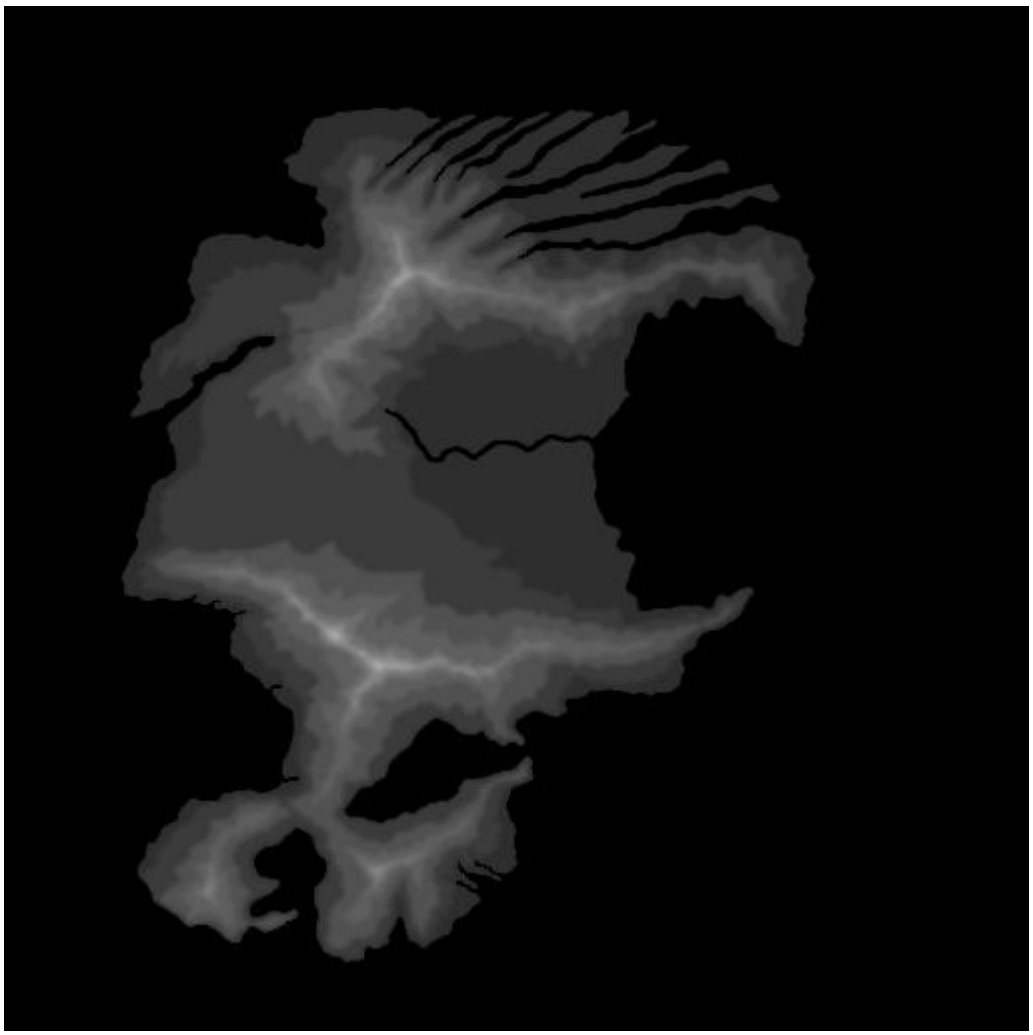


Imagen 101. Ejemplo de heightmap

9.1.10.4 Uso de texturas sobre el terreno

Ya que un terreno es un objeto que por regla general tiene un tamaño muy grande, es una práctica habitual utilizar texturas que se repitan a lo largo de su superficie, ya que dicha repetición no es perceptible desde el punto de vista de un personaje cerca del suelo.

Como ya comentamos anteriormente, Unity admite el uso de texturas PBR (Physically based rendering), las cuales, mediante el uso de modelos matemáticos, aproxima los diferentes

comportamientos de la iluminación (reflexión y refracción). Gracias a este tipo de texturas, podemos simular por ejemplo las hendiduras entre los ladrillos de una pared, cuando está en realidad es completamente plana.

Imagen 102. Textura PBR de ladrillo de piedra

Dado que inicialmente el terreno no tiene ninguna textura asignada, la primera que asignemos, se establecerá por defecto en todo el mapa. Para realizarlo, sobre el botón con la imagen de pincel(paint texture)→ Edit textures→ Add Texture.

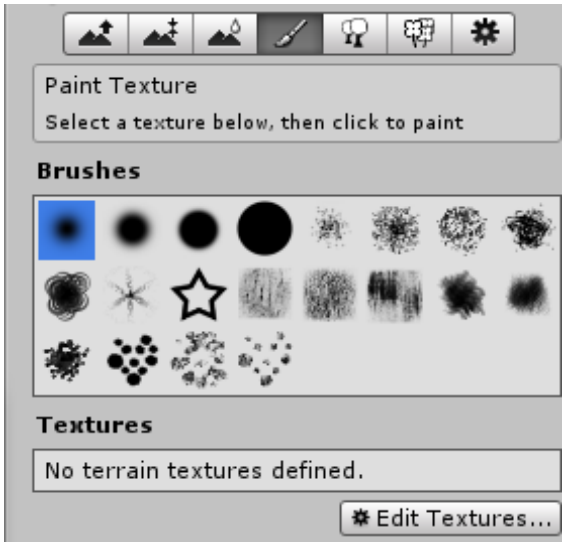


Imagen 103. Paint Texture

Y añadimos la textura que deseemos y establecemos su tamaño.

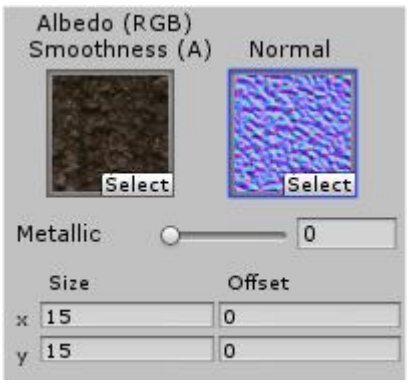


Imagen 104. Textura de barro rocoso

Asignando de esta manera dicha textura a todo el mapa de terreno.

Imagen 105. Vista del mapa de terreno con la textura de barro rocoso

Si añadimos más texturas en el panel de Paint texture, estas ya no aparecerán aplicadas en el terreno como cuando añadimos la primera, sin embargo podemos pintarlas sobre este mismo seleccionando ducha textura y un pincel. De esta manera todas las zonas del terreno sobre las que pasemos el pincel pasaran a tener la textura seleccionada.

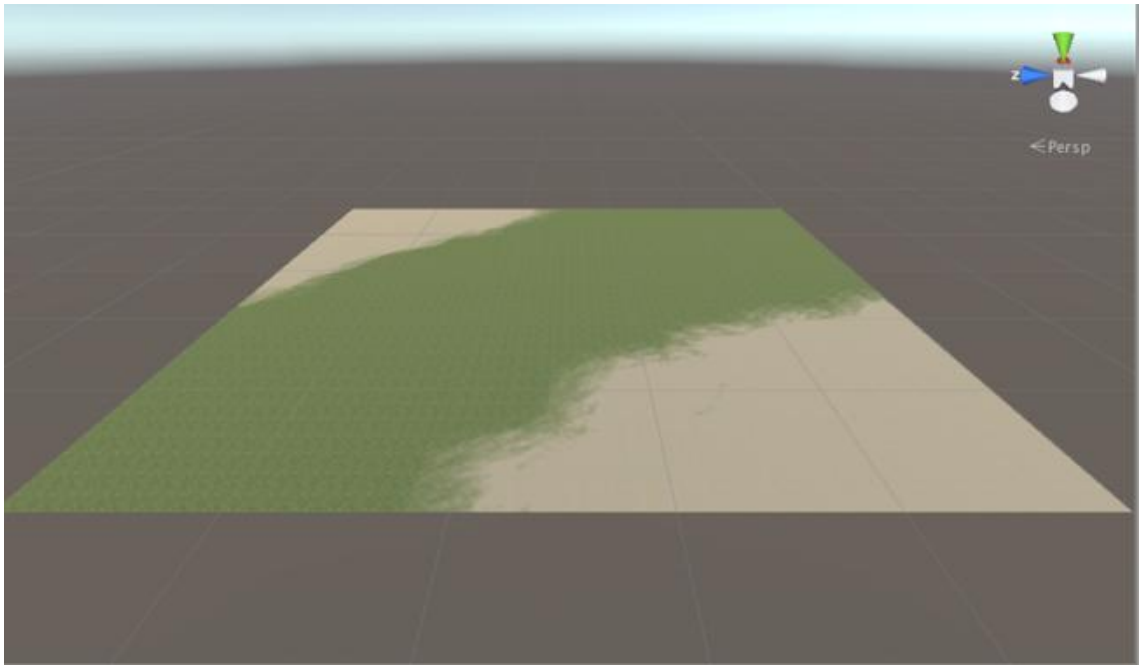


Imagen 106. Terreno de césped con la textura de tierra pintada en las esquinas

9.1.10.5 Árboles

Una vez diseñado el terreno, este lo podemos poblar rápidamente a través del botón de árboles en la barra de herramientas.

Imagen 107. Botón Place Trees

Suponiendo que ya disponemos de assets de árboles (como los incluidos en los estándar assets de Unity), al hacer clip sobre el botón Edit Trees-> Add tree podemos añadir los árboles que usaremos sobre el terreno.

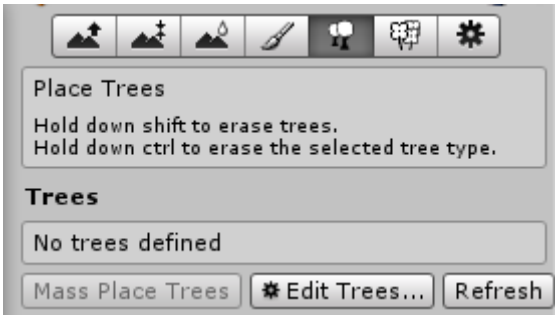


Imagen 108. Place Trees

Una vez añadidos, solamente tenemos que seleccionar el tipo de árbol que queremos colocar y al igual que con las herramientas de edición de terrenos, podemos igualmente "pintar" los árboles. Mediante las opciones Tree density, Tree height, Random Tree Rotation y Mass Place Trees controlamos la densidad de árboles por superficie, su altura, su orientación y además los podemos colocar de manera masificada.

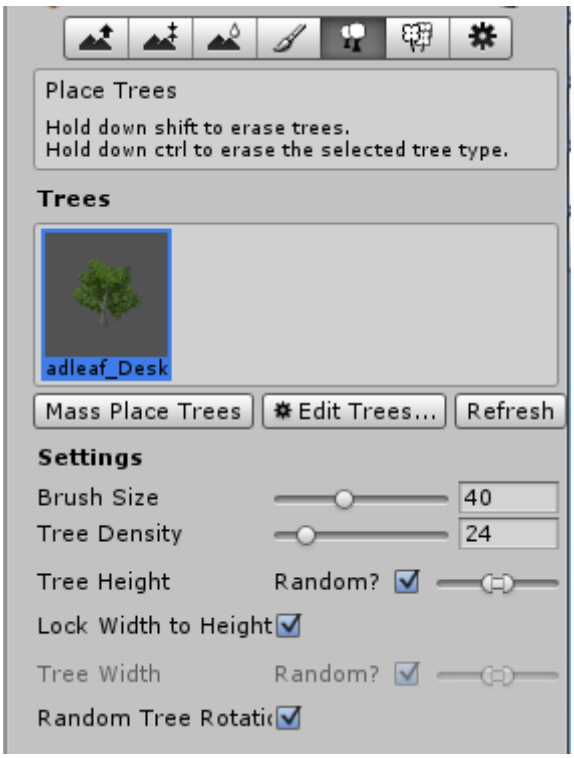


Imagen 109. Ventana Place Trees con un asset de árbol añadido

9.1.10.6 Hacer que los árboles se muevan con el viento

El primer punto esencial es asegurarnos que los árboles están configurados para curvarse. Desde el inspector del terreno, ir al Place Trees→Edit tres→ Edit tree y establecemos el valor de curvatura del árbol a 1.

El siguiente paso es crear una zona de viento añadiendo un objeto del tipo Wind Zone. Para ello, desde el menú pulsamos en Game Object→ Create General → Wind Zone.

Por defecto se puede observar que los árboles de mueven de manera muy violenta. Para corregir esto basta simplemente con disminuir la turbulencia del viento (wind turbulence) bajando su parámetro en torno al 0.2.

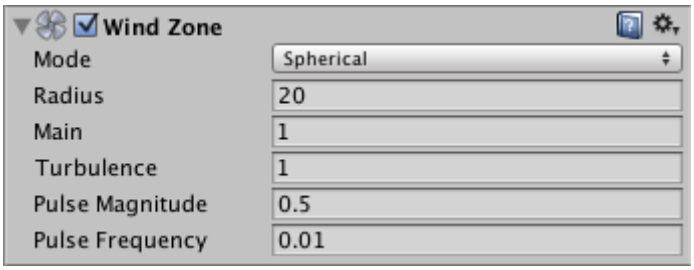


Imagen 110. Inspector de Wind Zone

9.1.10.7 Cesped y otros detalles

Un terreno puede tener además césped y otros pequeños objetos como piedras. El césped es renderizado usando imágenes 2D para representar las matas individuales mientras que el resto de detalles son generados por mallas 3D.

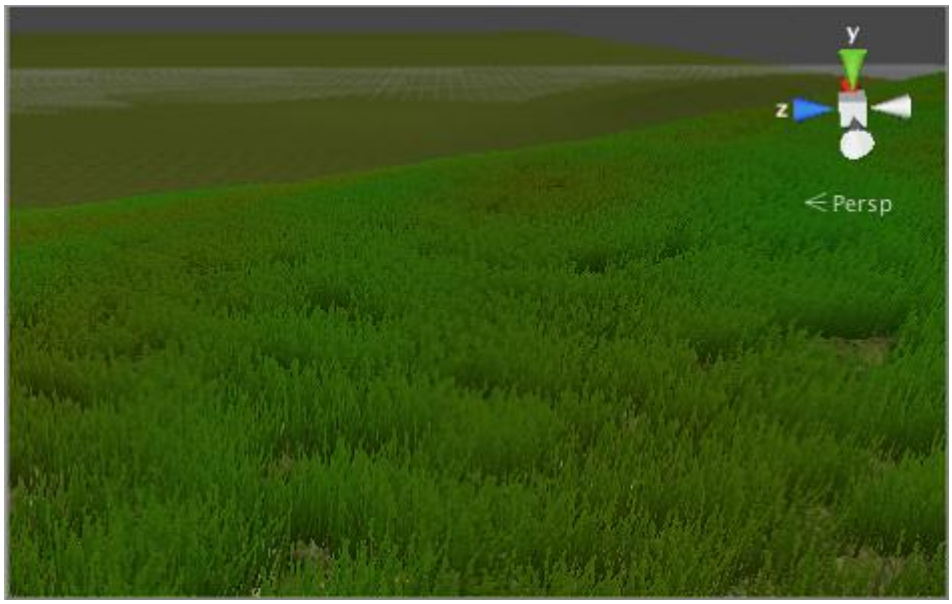


Imagen 111. Terreno con césped

El botón con el dibujo de flores situado en el cuadro de herramientas del terreno se conoce como Paint Detail y es a través de él con el que podemos "pintar" el césped y otros detalles similares.

Imagen 112. Botón Paint Detail

Funciona de manera similar al editor de árboles, para comenzar debemos añadir las texturas de césped pulsando sobre Edit details→ Add grass texture y Add Detail Mesh.

Una vez añadidos, mediante el uso del pincel "pintamos" las zonas que deseemos con césped.

9.2 Manual básico de Cinema 4D

Cinema 4D es un software de creación de gráficos y animación 3D desarrollado por la compañía Maxon para los sistemas basados en el Commodore y amiga y portado posteriormente a plataformas como Windows y Macintosh.

Por lo general, Cinema 4D es un software de desarrollo de pago, aunque permite el manejo de una versión demo y es posible la obtención de una licencia de estudiante sin coste alguno.

Para su instalación, no requiere conocimiento alguno, basta con descargar el ejecutable, seguir los pasos que nos indica y una vez finalizados, introducir la licencia.

9.2.1 Interfaz de usuario

La interfaz de usuario de Cinema [17] es libremente configurable. Podemos crear nuestras propias paletas de iconos e incluso editar los menús.

Se pueden definir varios entornos y conmutar entre ellos a nuestro antojo. Por ejemplo, puede ser útil crear un entorno para modelado y otro para animación dado que estas tareas las realizan diferentes gestores.

Dado que este programa en nuestro caso se utilizará únicamente para el modelado de los diferentes objetos 3D, la creación de sus texturas y "body paint", utilizaremos únicamente dos entornos; el entorno por defecto para modelado y el entorno de "body paint".

9.2.1.1 Manejo básico de la interfaz

El uso de esta aplicación es muy intuitivo, aunque es recomendable el uso de una Tablet en vez de un ratón para navegar, ya que puede hacerse muy engorroso el trabajar con fluidez al no disponer las herramientas de trabajo adecuadas.

Imagen 113. Editor por defecto de Cinema 4D

El primer punto de comienzo es la paleta de comandos. Esta nos da objetos 3D básicos, realización de primitivas booleanas (como intersección, unión y substracción), aplicación de nurbs para redondear objetos y otras varias.

Manteniendo pulsado sobre el Icono con forma de cubo de la paleta de comandos, podemos elegir los diferentes objetos 3D básicos a partir de los cuales procederemos a construir el objeto que tengamos en mente.

Imagen 114. Objetos 3D de la paleta de comandos

Una vez elegido el objeto, a través del gestor de coordenadas y del gestor de atributos podemos ubicarle dentro del entorno 3D y además modificar su tamaño en relación a los ejes de coordenadas.

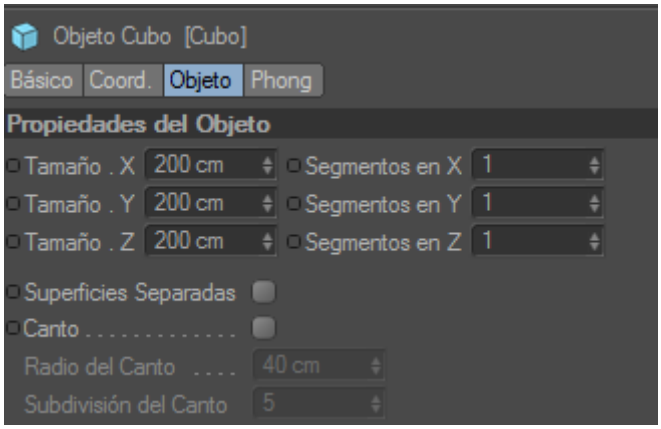


Imagen 115. Gestor de atributos

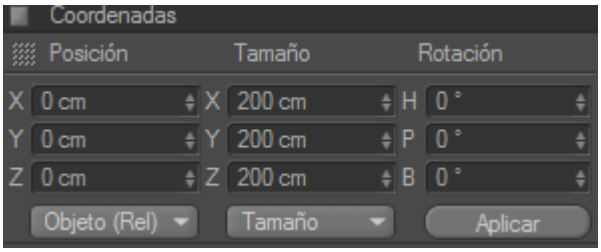


Imagen 116. Gestor de coordenadas

Obtenido y colocado el objeto 3D a partir del cual vamos a modelar, el siguiente paso es permitir su edición. Uno objeto paramétrico por defecto solo puede modificarse en su totalidad y no sus superficies individuales (con excepción de los "deformadores" especiales del menú de paletas en el submenú "Deformador")

Para ello, seleccionamos el objeto con el que estamos trabajando y para hacerlo editable pulsamos la tecla "c" del teclado, o en su defecto sobre el siguiente icono de la paleta de modos:

Imagen 117. Botón de edición

Una vez que es editable, a través de la paleta de modos para poder seleccionar puntos, aristas o caras del objeto, por medio del menú "estructura", podemos deformarlas, eliminarlas, subdividirlas y otras muchas opciones para así obtener la forma deseada.

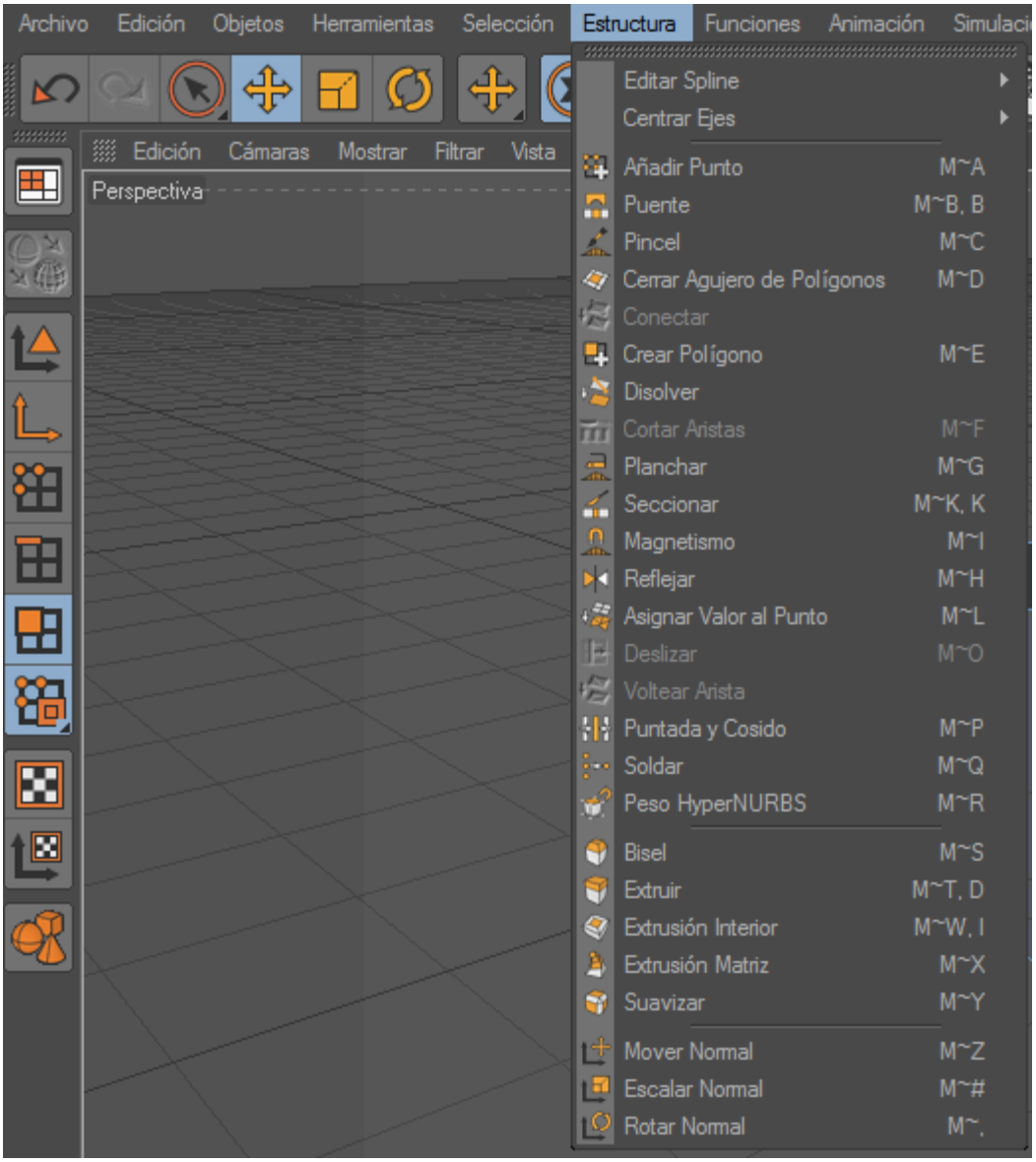


Imagen 118. Menú estructura

A través únicamente de lo mostrado hasta ahora, es posible realizar casi cualquier tipo de objeto básico que no requiera una estructura demasiado compleja.

Una vez finalizado nuestro trabajo, solo tenemos que guardar o exportar el objeto desde el menú archivo→guardar/guardar como.

9.2.1.2 Navegación

La navegación para visualizar el objeto con el que estemos trabajando desde diferentes puntos de vista, se lleva a cabo a través de los siguientes controles:

Imagen 119. Controlas de navegación

Empezando de izquierda a derecha, el primer icono, haciendo clic sobre él y manteniendo el botón del ratón o el boli de la Tablet pulsado, mueve la vista. El segundo icono (con la imagen de la doble flecha en perspectiva) permite carcar y alejar la vista. El tercero (las flechas curvas

con un punto en el centro), rota la escena. El último divide todo el panel de vista en cuatro perspectivas diferentes (superior, frontal, lateral y en perspectiva), cada cuadro tiene su propio icono que al pulsar sobre el convierte ese cuadro en la vista principal.

Los tres primeros iconos, se pueden también manejar mediante atajos de teclado que corresponden respectivamente a las teclas "1", "2", y "3".

9.2.1.3 Uso de los NURBS

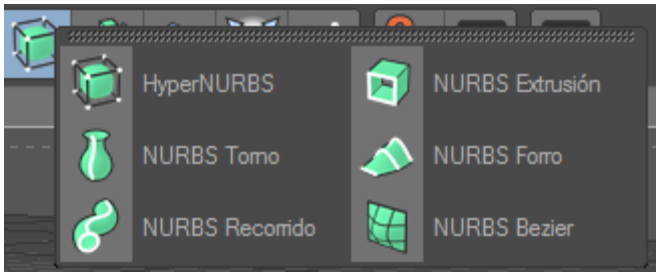


Imagen 120. Menú de objetos NURBS

Si un objeto poligonal, a través del gestor de objetos lo convertimos en un sub-objeto de un HyperNURBS, este se subdividirá virtualmente en todos sus polígonos. Visualmente estará compuesto de una mayor grado de polígonos más pequeños y tendrá un aspecto más suave y redondeado.

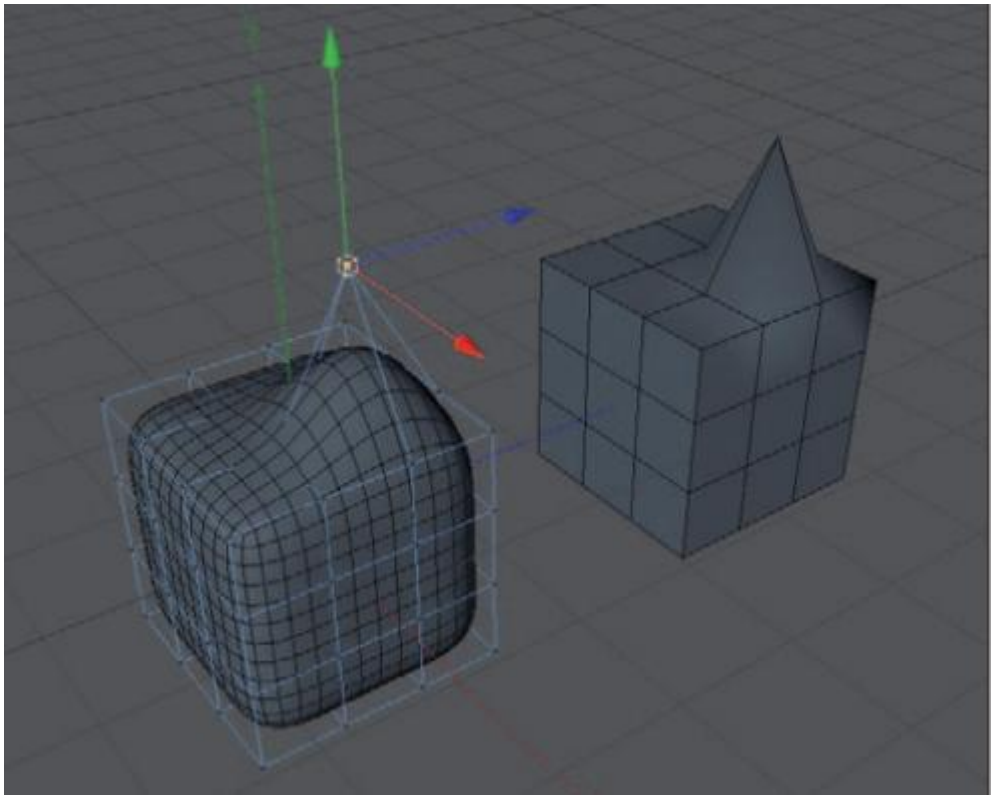


Imagen 121. Objeto 3D con hyperNURBS

Las ventajas del uso de los NURBS son obvias, un objeto poligonal formado por divisiones demasiado pequeñas y finas es muy difícil de modelar, sin embargo a través de los NURBS, podemos simplemente mover unos pocos puntos del objeto consiguiendo la forma deseada ni tener que alterar la posición de una gran cantidad de polígonos.

9.2.2 Creación de materiales

Por muy bien que este modelado un objeto, este puede dar una impresión mediocre si no se usan las texturas adecuadas.

Una textura proporciona a un objeto 3D, color, destellos, y el aspecto o apariencia que puede tener su superficie además de que también puede proveerle de una superficie irregular con relieve sin necesidad de alterar la estructura geométrica del objeto. De esta manera podemos imitar a la perfección superficies irregulares como paredes de piedra, barro, pieles de frutas, etc.

Para crear un nuevo material en Cinema 4D, solo tenemos que dirigirnos al panel de materiales y pulsar archivo→nuevo material.

Imagen 122. Creación de un nuevo material

De esta manera, se creará un nuevo material básico. Al hacer clic sobre este material, aparecerán sus propiedades den el gestor de atributos y en la pestaña "Básico" se pueden definir los canales y propiedades a activar para dicho material.

A continuación se explican los canales o propiedades que se pueden aplicar a cada material individual:

- Color: Establece el color base de la textura o material.
- Difusión: Este canal permite crear "irregularidades" sobre la textura. Mediante la aplicación de un shader de ruido, el objeto al que se le aplique el material recibe un aspecto de suciedad y/o polvo.
- Luminosidad: Dota al material de una propiedad luminosa que se tiene en cuenta en los cálculos de iluminación global.
- Transparencia: Establece la opacidad del material.

- Reflexión: Dota al material de características reflexivas.
- Entorno: Textura que simula la reflexión.
- Niebla: Permite aplicar una propiedad de niebla al material.
- Relieve: Usa un efecto óptico para recrear una superficie sin necesidad de alterar el objeto sobre el que se aplique dicho material, simulando arrugas, arañazos o similares.
- Normal: Este canal permite que un objeto de baja resolución poligonal posea un aspecto de alta resolución al aplicarse texturas RGB con ciertas propiedades. De esta manera se puede trabajar con objetos de baja resolución que ofrezcan un resultado visual idéntico a los de alta resolución y ahorrar en tiempo de renderizado.
- Alfa: Este canal determina la transparencia del material por las zonas claras y oscuras de una textura. El negro equivale a transparente (100%) y el blanco a opaco (0%).
- Especular: Establece las propiedades especulares del material.
- Color Especular: Determina el color de la expecularidad del material y puede establecerse con una textura
- Fosforescencia: Dota al objeto de fosforescencia.
- Desplazamiento: Deforma un objeto usando valores claros y oscuros (imita una superficie irregular).

Una vez creado, para aplicarlo tenemos diferentes maneras. La primera y más sencilla que afecta al objeto por completo, consiste en arrastrar el material sobre el nombre del objeto 3D que aparece en el gestor de objetos.

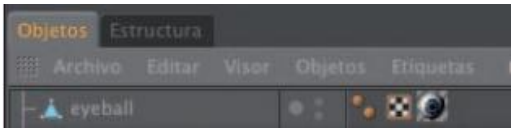


Imagen 123. Textura aplicada sobre el objeto eyeball

Y la segunda, consiste en seleccionar de manera manual los polígonos sobre los que se desea aplicar la textura y una vez seleccionados, arrastrar el material sobre estos.