Proyecto fin de carrera

Grado de Ingeniería Informática

Facultad de matemáticas en la Universidad de Barcelona

Demo de un videojuego 2.5D en unity3d con blender

Por **Rubén Monedero Plaza**

Director: Eloi Puertas Prats
Realizado en: Departamento de Matemática Aplicada y Análisis. UB
Barcelona,14 de Junio de 2013

INDICE MEMORIA

RESUMEN

Demo de un videojuego 2.5D, usando Unity3d y Blender.

Se ha creado un videojuego usando el programa Unity3d, un motor de videojuegos multiplataformas de alto nivel, complementado, ademas, para la parte de modelado y animación con el programa Blender. En el desarrollo del protipo, se ha seguido un modelo similar al modo iterativo e incremental aplicado en empresas para estructurar el diseño, y se ha conseguido realizar un juego de plataformas, con posible distribución para Windows, Mac y Linux, ademas de para servicio web.

RESUM

Demo de un videojoc 2.5D, fent servir Unity3d i Blender.

S'ha creat un videojoc fent servir el programa Unity3d, un motor de videojocs multiplataformes d'alt nivell, complementat, a més a més, per la part de modelat i animació amb el programa Blender. En el desenvolupament del prototip, s'ha seguit un model similar al mode iteratiu i incremental, que s'aplica a empreses. Així doncs, s'ha aconseguit realitzar un joc de plataformes, el qual es distribuïble tant per Windows, Mac, Linux com per a web.

ABSTRACT

Demo of a game 2.5D, using Unity3d and Blender.

It was created a game using unity3d program, a high level multiplatform videogame engine, supplemented with the modeling and animation program Blender. It was followed a similar of iterative and incremental model that companies use to the structure design, and has managed to make a platform game, with possible distribution for Windows, Mac and Linux, in addittion to Web service.

AGRADECIMIENTOS

Primero agradecer la colaboración de mi tutor del proyecto Eloi Puertas Prats, por mantener contacto conmigo, darme la posibilidad de realizar este proyecto, y ayudarme dándome las indicaciones de manera simple y la implicación necesaria para poder acabar el proyecto de manera satisfactoria.

También me gustaría agradecer a mis amigos y familia por el apoyo y paciencia que me han dado, y sobretodo por las opiniones objetivas que me han podido dar, al ir probando las versiones que iba sacando, y dándome ideas de cosas que podía cambiar o mejorar, ya que al tratarse de un videojuego es algo muy visual hasta para gente que no entiende del tema de informática. Vuestra ayuda me ha sido muy importante, gracias.

CAPITULO 1: INTRODUCCIÓN

En este tema se explican las motivaciones del proyecto, los objetivos de este, ademas del tipo de videojuego propuesto a realizar y el esquema final propuesto para la memoria.

1.1. Motivación del proyecto

Actualmente la industria de los videojuegos es el sector económico involucrado en el desarrollo, distribución, mercadotecnia, la venta de videojuegos y el hardware asociado. Esto engloba una docena de disciplinas de trabajo actualmente, y emplea miles de personas en el sector a realizar estas pequeñas obras de arte.

Esta industria a crecido de manera muy elevada estos últimos años, debido al desarrollo de la computación, capacidad de procesamiento, imágenes mas reales y la estrecha relación entre películas de cine y los videojuegos. En la década de 2000, los videojuegos han pasado a generar mas dinero que el cine y la música juntas.

Teniendo en cuenta todos estos datos, se puede afirmar que es uno de los sectores actualmente en cabeza, con un gran numero de puestos de trabajo. Ademas esta industria trabaja para diferentes dispositivos, ya sean consolas, móvil, ordenadores, etc ... teniendo en cuenta todo esto la idea era buscar una herramienta que me permitiera crear aplicaciones multiplataforma para conseguir tener un acercamiento a las tecnologías necesarias para entrar en este sector.

Por este razonamiento se llego a elegir Unity3d un tipo de motor de videojuegos orientado a crear aplicaciones multiplataforma, ademas de ser un programa con grandes virtudes en este sector que se irán viendo en próximos temas.

1.2. Objetivo del proyecto

El objetivo de este proyecto es el de realizar un videojuego usando Blender y Unity3D. Para ello se realizara un estudio y explicación de ambos programas, sin pretender explicar todas las funcionalidades de estos, ya que son muy extensos, pero si que se explicaran los conceptos básicos que necesitaría un usuario que quisiera crear un videojuego desde sus comienzos. Ademas se dará una introducción a ambos programas, explicando información adicional de estos.

En esta memoria se usara de ejemplo, el videojuego plataformas creado para la ocasión para el aprendizaje propio del programa, utilizando la versión gratuita de este. En esta memoria se incluirán los siguientes pasos de la creación del videojuego:

- Crear escenarios y entornos
- Modelar y animar un personaje o objeto
- Hacer que dicho personaje sea controlable por el jugador
- Crear scripts sencillos, que pueden incluir, las físicas de los objetos, gestión de colisiones, inteligencia artificial de enemigos
- Crear diferentes escenas, aparte de menús para enlazarlas
- Crear un GUI donde se pueden ver nuestra situación actual, pantalla que estamos, vida actual, etc ...
- Crear las distribuciones según nos interesen, ya sea para pc(windows, linux, mac) o para webplayer.

1.3. Videojuego propuesto

La idea del videojuego, es que se tratara de un videojuego de plataformas en 2.5D por petición propia.

Para entrar en mas detalle, este tipo de videojuegos son un paso intermedio entre los juegos en 2D(2 dimensiones) y los juegos en 3D(3 dimensiones), quedándose en los movimientos en un plano en 2D pero teniendo escenario y objetos en 3D, o que por el punto de vista parece ser en 3D.

Algunos ejemplos serian:

Figura1 :Castelvania X The Dracula Chronicles(SonyPSP)

Figura2 :New Super Mario Bross (Wii)

Partiendo de esta base pasaríamos a nuestro juego:

- **Nombre del juego:**

  Time is ending

- **Descripción general:**

  El juego nos coloca en la piel de un personaje protagonista, el cual a llegado a una cueva encantada en la que tendrá que superar diversos obstáculos o enemigos, usando sus capacidades motrices (andar, saltar, golpear, etc...).

- **Mecánica:**

  El juego empezara en un menú principal donde puede empezar la partida, esta partida la empezara con un limite de vidas que puede perder si uno de los obstáculos o enemigos, lo mata, en el caso de que se quede sin vidas el usuario perderá y pasara a ver una fase de game over para volver a reiniciar si quiere, en el caso de que llegue al punto final de la pantalla, se le dará como campeón y se le felicitara.

1.4. Esquema del documento

En este apartado viene una pequeña explicación de el contenido de los diferentes capítulos que componen esta memoria.

- **Capitulo 1 Introducción** : En este capitulo están las motivaciones, objetivos, el videojuego propuesto y el esquema del documento

- **Capitulo 2 Estado del arte** : En este capitulo se hace una breve explicación de la historia de los videojuegos, se explica que es un motor de videojuegos y que tipos de motores hay actualmente.

- **Capitulo 3 Introducción a los programas utilizados** : Explicación de Blender y Unity3d, los dos programas utilizados, el porque han sido elegidos estos dos, y un pequeño acercamiento a ambos programas.

- **Capitulo 4 Videojuego "Time is ending"** : Muestra de los resultados finales de el videojuego, en este capitulo se pueden ver bocetos, estructura de objetos, escenas,scripts y screenshots del juego.

- **Capitulo 5 Proceso de desarrollo** : Explicación de como se ha ido creando el videojuego, que método de desarrollo y planificación se ha utilizado, y las iteraciones hechas durante el proceso.

- **Capitulo 6 Conclusiones** : Las conclusiones finales que he sacado de este trabajo de fin de grado.

- **Capitulo 7 Bibliografía** : Toda la bibliografía utilizada para el aprendizaje y utilización de dichos programas.

- **Anexos de la memoria** : Anexos que contienen todo lo que podremos encontrar en el archivo adjunto con el trabajo, en este caso el videojuego y como esta estructurada la carpeta que lo contiene.

CAPITULO 2:

ESTADO DEL ARTE

2.1. Historia de los videojuegos

La historia de los videojuegos tiene su origen en la década de los 1940 cuando tras la segunda guerra mundial, las potencias vencedoras construyeron las primeras supercomputadoras programables. Los primeros intentos no tardaron en aparecer, intentando crear juegos de ajedrez, y se fueron repitiendo durante las siguientes décadas.

Los primeros videojuegos modernos aparecieron a partir de la década de los 60, y desde este momento el mundo de los videojuegos no ha dejado de crecer y desarrollarse con el único limite de la creatividad de los desarrolladores y de la evolución de la tecnología.

- **Antecedentes**: Durante la Segunda Guerra Mundial el matemático británico Alan Turing había trabajado junto al experto en computación estadounidense Claude Shannon para descifrar los códigos secretos usados por los submarinos alemanes U-Boot. Las ideas de ambos científicos, que ayudaron a establecer las bases de la moderna teoría de la computación, señalaban la Inteligencia Artificial, como el campo más importante hacia el que había que dirigir todos los esfuerzos de investigación. En 1949 Shannon presentó un *paper* en una convención de Nueva York titulado *Programming a Computer for Playing Chess* donde presentaba muchas ideas y algoritmos que son utilizados todavía en los programas modernos de ajedrez.

  Otro juego que se había implementado tempranamente era el de las tres en raya(En México conocido como "Gato"), que Alexander Douglas había recreado en el EDSAC de la University of Cambridge en 1952 como parte de su tesis doctoral. OXO incorporaba gráficos muy similares a los actuales y aunque fue mostrado únicamente a unos pocos estudiantes de la Universidad, es considerado por algunos como el primer videojuego moderno de la historia.

William Higinbothamun ingeniero norteamericano que había participado en el Proyecto Manhattan, presentó un proyecto que cautivó a todos los visitantes de su laboratorio: un juego de tenis que había construido con la ayuda del ingeniero Robert Dvorak usando la pantalla de un osciloscopio y circuitería de transistores. El juego, que recreaba una partida de tenis presentando una visión lateral de la pista con una red en el medio y líneas que representaban las raquetas de los jugadores, se manejaba con sendos controladores que se habían construido a tal efecto.

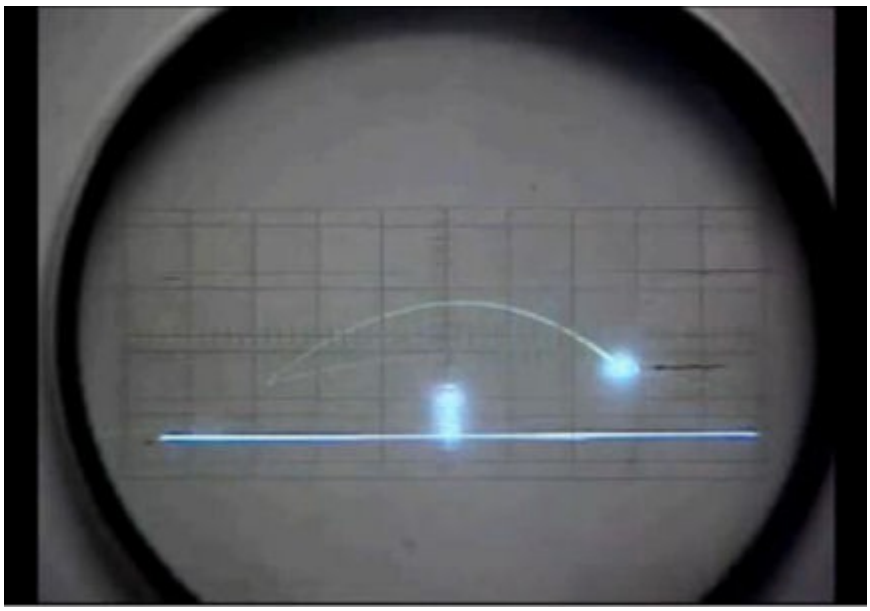


Figura 3 : Tennis for Two, uno de los padres de os videojuegos modernos

- **Decada de los 60**: La década de 1950 había sido una época de falsos comienzos para los videojuegos. Casi todos los que habían explorado la idea la habían abandonado inmediatamente convencidos de que era una enorme pérdida de tiempo. El ajedrez por computadora era un campo de investigación fructífero, pero permanecía dentro del ámbito académico, lejos del campo del entretenimiento. Sin embargo la idea de que los ordenadores sólo debían servir para fines serios tocaba a su fin.

  Tres de los miembros del club de estudiantes de el Masachussets Institute of Technology, se reunieron para decidir qué harían con él. Wayne Witaenem, Martin Graetz y Steve Rusell decidieron que harían un juego, y bajo el liderazgo de éste último desarrollaron Spacewar!, un duelo espacial para dos jugadores que vio la luz en 1962. El juego ocupaba 9k de memoria y causó sensación entre los miembros del MIT; numerosas copias del mismo fueron distribuidas a través de APRAnet.

Figura 4 : Spacewar! La primera idea de un videojuego que luego seria rediseñado multitud de veces

- **Decada de los 70**: A finales de la década de los 60 Bill Pits, un estudiante de la Universidad de Stanford fascinado por Spacewar! tuvo la idea de hacer una versión del juego que funcionase con monedas para su explotación en los salones recreativos. Desafortunadamente, el precio del hardware requerido para ejecutar el programa era mucho más elevado de lo que los propietarios de los salones, acostumbrados a pagar unos 1000 dólares por las máquinas electromecánicas de la época, podían permitirse. Cuando el nuevo PDP-111 apareció en el mercado al "económico" precio de 20 000 dólares, Pitts pensó que tenía ante sí la oportunidad real de construir su máquina, y llamó a Hugh Tuck, un amigo del High School para construir un prototipo. En 1971 ambos formaron Computer Recreations, Inc., con el propósito de construir una versión operada con monedas de *Spacewar!*; Pitts se hizo cargo de la programación y Tuck, ingeniero mecánico, construyó la cabina. Tras tres meses y medio de trabajo habían finalizado la máquina, pero decidieron cambiar el título del programa a Galaxy Game.

Figura 5 : Galaxy Game (1971)

Comenzó la fabricación de la recién bautizada Magnavox Odyssey*, la* primera videoconsola de la historia en una fábrica de Tennesse , y en abril de 1972 la firma presentó la nueva máquina a la prensa y a sus distribuidores. Al mismo tiempo se presentó el primer accesorio de la máquina, un rifle de plástico de buena apariencia, y diez juegos adicionales, todos ellos vendidos por separado. Los *flyers* de la época mostraban ya una consola de videojuegos exactamente tal y como la conocemos hoy, pero Magnavox cometió una serie de errores de marketing que jugaron en su contra: por un lado el anuncio de televisión daba la impresión de que la consola sólo se podría usar con un aparato de televisión de la misma marca, lo que no era cierto; por otro lado, la distribución se limitó a las franquicias de Magnavox, lo que limitaba considerablemente el número de clientes potenciales.

Figura 6 : Magnavox Odyssey (1972)

El 24 de mayo de 1972 Nolan Bushnell estaba entre el público asistente a una demostración de la Magnavox Odyssey que estaba teniendo lugar en Burlingame, California. Bushnell tuvo la oportunidad ese día de jugar al Ping-pong, uno de los juegos que incluía de serie la nueva consola, y tras este episodio contrató a Alan Alcorn, un ingeniero de Ampex a quien puso a trabajar en una versión árcade del juego que recibió el nombre de "*Pong*".

El inmenso éxito de *Pong* en 1972 reestructuró por completo el negocio del entretenimiento. De la mano de la compañía Atari los videojuegos eran más novedosos y confiables que los juegos electromecánicos típicos de la época, pues carecían de elementos mecánicos susceptibles de rotura y desgaste. La firma comenzó a comercializar una serie de productos a cual más innovador. Mientras sus competidoras no hacían más que lanzar nuevas copias de Pong, Atari comercializó nuevos éxitos comoSpace Race, Rebound, Gotcha, Quadrapong, Touch Me, Tank, Qwak!, o Gran Trak 10, cada uno de los cuales suponía en la práctica la inauguración de un nuevo género.

Mientras tanto, en Japón Tomohiro Nishikadoadopta la nueva tecnología de microprocesador, creó para Taito **Space Invaders**, un juego que originalmente consistía en disparar contra tanques y

aeroplanos, pero que, en parte por la presión de la compañía y en parte por el gran éxito que el film *Star Wars* estaba cosechando en la época, acabó adoptando su forma definitiva de batalla espacial. El juego obtuvo inmediatamente un éxito de dimensiones descomunales, fue convertido a todos los formatos importantes de la época y dio lugar a numerosas continuaciones e infinitos clones. No sólo inició un género que resultaría basilar en el desarrollo posterior de los videojuegos (el de los Shoot 'em upo "matamarcianos"), sino que situó definitivamente a la industria japonesa en el lugar que le correspondía e impulsó definitivamente la fiebre de los videojuegos a nivel mundial, iniciando la que en la literatura especializada se conoce como la **Edad dorada de los videojuegos.**

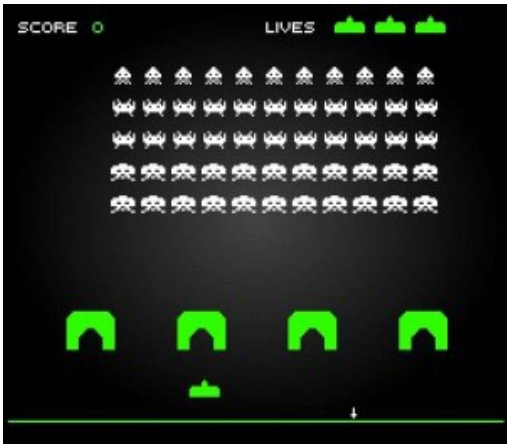


Figura 7 : Space Invaders(1978)

- **Decada de los 80** : Aqui se iniciaba la edad de oro de los videojuegos (1978 - 1983) . En el verano de 1982 la fiebre por los videojuegos aumentó considerablemente. Desde que Space invaders irrumpió en el mercado en 1978 los ingresos generados por la industria habían pasado de los 454 millones de dólares de ese año hasta los 5 313 millones de 1982, es decir, estaba incrementando sus beneficios un 5% mensualmente. El interés del gran público por los videojuegos parecía no tener fin y las máquinas recreativas se encontraban por doquier, desde los bares y restaurantes hasta los hoteles y supermercados.

  Los gigantes japoneses continuaron avanzando, en este caso, Hiroshi Yamauchi se encontraba entre los emprendedores que pensaban que el futuro estaba en la industria de los videojuegos, y así lo hizo saber a los directivos de su empresa, ordenando que abandonasen sus viejos proyectos para centrarse en la creación de videojuegos. Una tarde de 1980 Gumpei Yokoi, a la sazón Jefe de juguetes de Nintendo, observó a un hombre de negocios que jugueteaba aburrido con su calculadora electrónica, lo que le dio la idea de crear un videojuego portátil con tecnología LCD fácil de

fabricar y fácil de transportar. El resultado fue la serie Game & Watch, unas máquinas de bajo coste y tecnología LCD que hacían las veces de reloj y que venderían la friolera de 30 millones de unidades a lo largo de los siguientes 11 años. Nintendo lanzó al mercado el 15 de julio de 1983 la primera versión de su Famicom, un modelo muy innovador que disponía entre su catálogo de títulos varias versiones del clásico *Donkey Kong*, que dio lugar a algunas de las más famosas series de personajes de los videojuegos, y que estaba destinado a sentar las bases del mundo de las videoconsolas domésticas modernas. La consola catapultó a Nintendo al éxito de tal manera que la compañía no pudo abastecer la enorme demanda de videojuegos que existía para la nueva máquina, y la solución de Yamauchi, absolutamente innovadora para la época, fue establecer un sistema de licencias a terceras compañías mediante el cual se reservaba el derecho a publicar los títulos que considerase cumplían ciertos estándares de calidad al tiempo que cobraba a las compañías desarrolladoras por permitirles publicar sus títulos para la nueva plataforma. El éxito de la idea fue rotundo, y el catálogo de juegos de la máquina creció exponencialmente, incluyendo títulos tan emblemáticos como Super Mario Bros, Tennis, Dragon Quest, The legend of Zelda o Final Fantasy.

Figura 8 : Consola Game & Watch

- **Decada de los 90** : Maquinas de 16 bits y nuevas consolas, podrian definir esta decada. Junto a la consolidación definitiva de la industria y a cierto estancamiento en la creatividad de los creadores de videojuegos, debido en parte a la estabilización que habían sufrido los distintos generos.

  El mundo de las consolas estaba liderado por la Sega Mega Drive, una máquina de 16 bits que había sido lanzada en 1988 y que disponía de títulos tan populares como Sonic The Hedgehog. El liderazgo de Sega se mantuvo hasta 1991, año en el que Nintendo comercializa su Super Nintendo para reconquistar el mercado con títulos como Super Mario Kart(1992).

Figura 9 : Sega Megadrive

- La aparición de la Game Boy de Nintendo en 1989 supuso una pequeña revolución en su campo. Creada por Gunpei Yokoi, el mismo ingeniero que había diseñado las exitosas Game & Watch, la máquina de Nintento se posicionó enseguida como líder en ventas en el segmento de las consolas portatiles, gracias en parte a su reducido precio y en parte a su enorme catálogo.

Figura 10 : Nintendo Game Boy

Repentinamente la realidad virtual se convirtió en uno de los tópicos más recurrentes en las conversaciones acerca de la investigación informática, lo a su vez estimuló la aparición de documentales y programas de televisión qu e divulgaron la idea entre el gran público. Mientras, los desarrolladores de videojuegos se enfrentaban a los mismos retos que los investigadores del área de la realidad virtual.

Mientras tanto, Romero y sus colaboradores se encontraban en pleno desarrollo de Doom, un programa que, con su publicación en 1993 sacudió de arriba a bajo la industria de los videojuegos cambiando para siempre el curso posterior de la historia con sus muchas innovaciones. Doom no sólo destacaba por sus gráficos y la solidez de su engine, sino también por su inusitada violencia, nunca vista hasta la época. Se convirtió asimismo en el primer videojuego de la historia que permitía oficialmente el modding, esto es, la modificación de su diseño y sus niveles por parte de los jugadores, una idea que en realidad se remontaba a los juegos de pinball de la década de 1980. Doom popularizó enormemente el juego en linea, permitiendo a los jugadores el juego en red, bien a través de Internet, bien a través de conexiones por cable, y también supuso una revolución en el campo del marketing.

Figura 11: Captura de pantalla de Freedom la version libre de doom

- **De 2000 en adelante** : Lo que mas destaca desde esta epoca hasta ahora, es la nueva generacion de maquinas con mas potencia ya que la industria de las maquinas recreativas estaba en crisis, saliendo Sony como lider. Se crearon nuevos generos y franquicias multimillonarias como podrian ser los Sims o Gran Theft Auto.

  La creatividad que empujó a los programadores pioneros y amateurs de las décadas de 1960 y 1970 se encuentra también en las grandes superproducciones actuales, transformada y adaptada a los medios y tecnologías actuales. Lejos de haber alcanzado su madurez creativa, los videojuegos siguen siendo una nueva forma de arte que parece estar dando aún, 50 años después de su aparición, sus primeros pasos .

Figura 12 : Consola Playstation 3

2.2. Motor de videojuegos

Un motor de videojuegos es un término que hace referencia a una serie de rutinas de programación que permiten el diseño, la creación y la representación de un videojuego. Del mismo modo existen motores de juegos que operan tanto en consolas de videojuegos como en sistemas operativos.

La funcionalidad básica de un motor es proveer al videojuego de un motor de renderizado para los gráficos 2D y 3D, motor físico o detector de colisiones, sonidos, scripting, animación, inteligencia artificial, redes, streaming, administración de memoria y un escenario gráfico. El proceso de desarrollo de un videojuego puede variar notablemente por reutilizar o adaptar un mismo motor de videojuego para crear diferentes juegos. Algunos conceptos básicos para entender el funcionamiento del motor serian estos:

- **Assets** : Son los modelos, animaciones, sonidos, IA, físicas. Son los elementos que forman el juego en sí, el código hace funcionar los assets.

- **Render** : Es la parte del código que pone en pantalla los ambientes y objetos.

- **Three-point polygon** : Poligonos de tres puntas, triangulos, que son los más usados por las tarjetas aceleradoras 3D

- **Iluminación (lighting)** : Distintos APIs proveen diferentes tipos de iluminación

- **Sonido** : Creative Labs ahora ha proporcionado sus extensiones manejadores de sonido EAX para DirectX, y la nueva iniciativa de OpenAL (biblioteca audio abierta). OpenAL, como suena, es un API para los sistemas de los sonidos de la misma manera que OpenGL es un API.

- **Inteligencia Artificial**: La IA provee de estímulo al juego.

2.3. Tipos de motores

Los motores de videojuego vienen explicados de muchas formas diferentes y a muchos niveles de conocimiento de programacion. Para tener una idea de lo diferentes que pueden llegar a ser, se van a explicar tres tipos generales de motores :

- **Roll-your-own game engines**(nivel inferior): A pesar del coste hay muchas empresas ordinarias, que intentan sacar sus propios motores. Esto significa que utilizan interfaces de aplicaciones ya disponibles para el publico, como las API como XNA, DirectX, OpenGL, o las de Windows, Linux y SDL, para crear sus propios motores. Ademas se pueden utilizar otras bibliotecas, tanto comerciales y de codigo abierto, para ayudarles a lo largo del desarrollo.

  En general estos sistemas personales, dan a los programadores la mayor cantidad de flexibilidad, permitiendoles escoger y elegir los componentes que quieran y su integracion con la exactitud que ellos requieran. Pero tambien aumenta el tiempo para construirlo. Ademas los programadores frecuentemente tendran que construir toda la cadena de herramientas partiendo de cero, ya que rara vez se pueden trabajar con estas bibliotecas hacia otros proyectos. Esto hace que utilizar estos motores sea menos atractivo para los desarrolladores de juego, incluso para profesionales.

- **Mostly-ready game engines**(nivel intermedio): Son motores ya preparados directamente para ser usados, proporcionando muchas herramientas para la creación de juegos. Herramientas como una interfaz de usuario, de motor de físicas o otro tipo de opciones de renderizacion. Estos motores suelen requerir algo de nivel de programación para la creación del juego y generalmente dan mejor resultado que los nombrados anteriormente Roll-your-own, ya que requiere menos tiempo y menos esfuerzo.

  Alguno de los ejemplos de motores de este tipo podrían ser: ORGE, Unreal, id Tech o Torque, todos de diferente gama y precio.

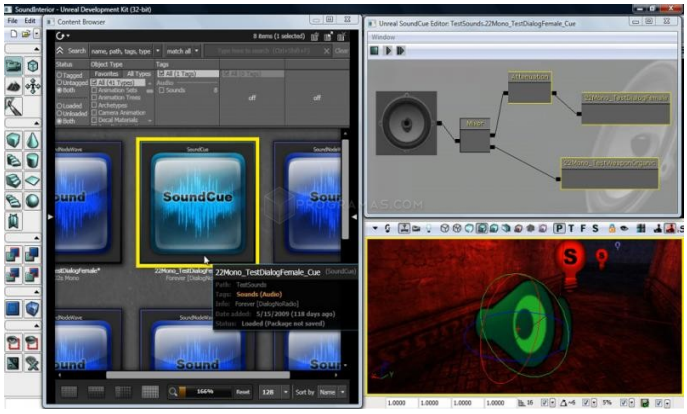


Figura 13 : Unreal engine

- **Point-and-click game engines**(nivel alto): Los motores de apuntar y seleccionar se están volviendo cada vez mas y mas extendidos en estos días. Estos incluyen una gran cantidad de herramientas que te permiten apuntar y seleccionar la manera de la que tu quieras diseñar tu juego. Algunos de los ejemplos de estos motores, son : GameMaker, **Unity3d**, Torque Game Builder. Intentan ser lo mas amistosos posible y están creados para necesitar la menor cantidad de código posible, esto no quiere decir que saber programar no vaya a ser de gran ayuda, pero no es tan necesario como en los dos anteriores motores.

  El problema que tienen estos motores, es que aveces se puede considerar que son limitados, a uno o varios tipos de géneros de juego, o uno o dos tipos de modos gráficos. Pese a estas restricciones, se pueden crear una gran cantidad de juegos creativos y encontrar maneras creativas de superar estas restricciones. La mejor cosa de estos motores es que te permiten trabajar rápido, y te permiten probar tus juegos rápidamente, sin tener una gran cantidad de trabajo realizado.

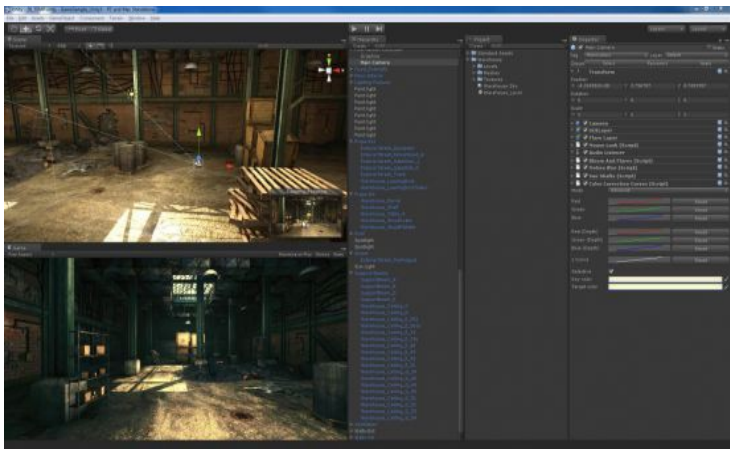


Figura 14 : Unity3d

CAPITULO 3: INTRODUCCIÓN A LOS PROGRAMAS UTILIZADOS

En este tema se explican los dos programas principales utilizados y una pequeña iniciación a ellos. El motivo de que se hayan elegido tanto Blender como Unity3d, es porque son dos programas totalmente compatibles, ya que unity3d permite la utilización de modelos blender de manera automatizada.

3.1. Que es Blender

Blender es un programa de modelado en 3D, apoyado por varias herramientas, es ademas multiplataforma (trabaja sobre windows XP, Vista 32 y 64 bits, Linux 32 y 64 bits, MacOS, solaris, etc.).Creado por la empresa Not a Number (NaN).

Está orientado a artistas y profesionales del diseño y multimedia, puede ser usado para crear, visualizaciones 3D estáticas o vídeos de alta calidad. También incorpora un motor de 3D en tiempo real el cual permite la creación de contenido tridimensional interactivo que puede ser reproducido de forma independiente.

Blender se desarrolla como Software Libre, con el código fuente disponible bajo la licencia GNU GPL, su descarga y su uso es completamente gratuito.

3.1.2. Estado en la actualidad

Aún siendo una herramienta relativamente nueva, ha gozado de la aceptación de muchos animadores independientes. En la industria de Generación de gráficos avanza como un proyecto prometedor, si bien las superproducciones no lo han usado para generar secuencias CGI. Existen proyectos actuales que han empezado a usarlo profesionalmente:

- El 18 de febrero de 2010 se estrenó el primer largometraje animado realizado íntegramente con software libre, usando a Blender como herramienta principal; se trata de Plumíferos, un proyecto que está impulsando el desarrollo de Blender aún más, sobre todo a nivel de animación y manejo de bibliotecas a gran escala.
- Películas tales como Spider-Man 2 que lo ha usado para hacer una previsualización de escenas (Screen-Board Test), han usado de manera incipiente las capacidades del popular programa GNU/GPL.
- Algunas propuestas más llevadas a la producción e integración con gráficos mediante Motion Tracktales como "Friday or another day". que es de los primeros esbozos de su uso a 35mm.
- Otros proyectos hechos en participación de diversos usuarios de Blender incluido Ton Rossendaal el cortometraje Elephant Dreams son experimentos de sus capacidades, extendidas gracias a la posibilidad de poder editar su código fuente, aportando de esta experiencia a los demás usuarios con innovaciones fundamentales: Un sistema de control de gestos (Morph system), un sistema de composición de textura y post producción (Composite), entre otros.

Alguno de los artistas mas importantes que usan actualmente Blender como principal herramienta:

- Jesus Manuel Zamora Sanchez (Bauni), (estupi modof, Mejor Animacion)

- Andreas Goralczyk (@ndy), ganador de dos Suzanne Blender Awards consecutivos. (2003 - Mejor Animación, 2004 - Mejor Imagen)
- Stefano Selleri (S68), (Suzanne Blender Awards 2003 - Mejor Imagen)
- Bassam Kurdali (slikdigit), (Suzanne Blender Awards 2004 - Mejor Animación)
- Sacha Goedegebure (Sago)] (Suzanne Blender Awards 2006 - Mejor Animación de Personajes)

3.1.3. Características

Las principales características del programa son:

- Software libre, gratuito y multiplataforma
- Potente y versátil
- Importa y exporta de múltiples formatos 3D
- Soporte gratuito vía blender3d.org
- Manual multilenguaje en linea
- Una comunidad mundial creciente.
- Un archivo ejecutable pequeño que permite una fácil distribución
- Te puedes olvidar de números de serie y activaciones
- Múltiples plugins también gratuitos que expanden las posibilidades del programa
- Si sabes programar puedes usar el código fuente para hacer modificaciones

3.1.4. Primeros pasos

Para empezar, el programa es gratuito y se puede descargar de la pagina web www.blender.org. Una vez descargado el primer contacto que tenemos con el, es intimidatorio ya que en el interfaz se ven muchos elementos diferentes que a priori se pueden escapar un poco de nuestro dominio.

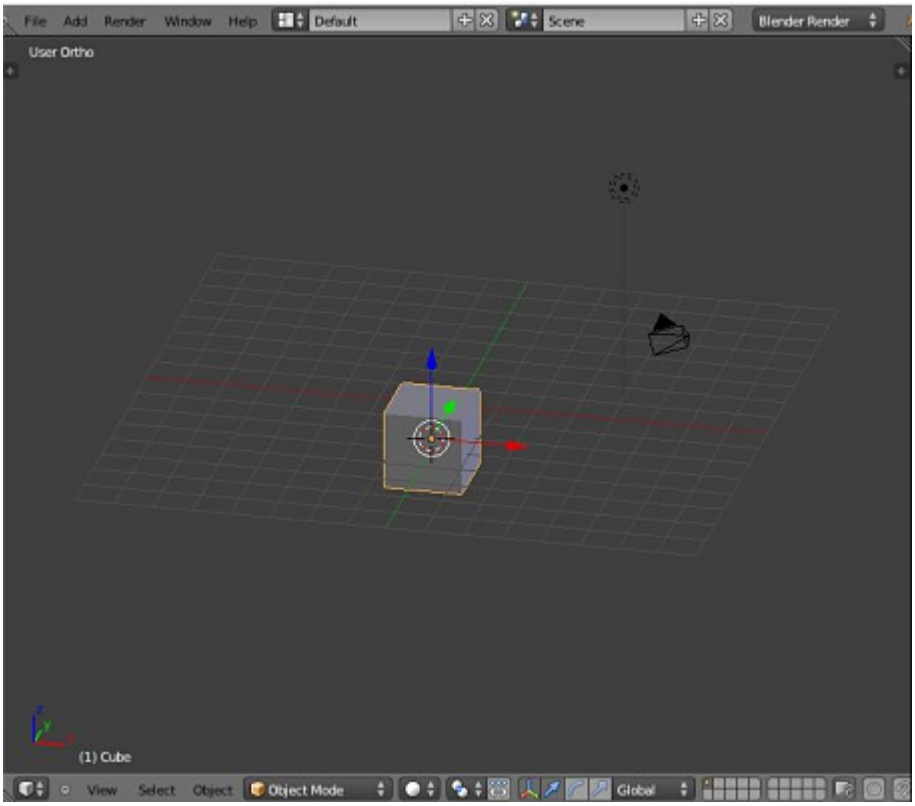


Figura 14 : Imagen inicial de la interfaz con un elemento cubo creado

En la figura anterior (fig.14), podemos observar en las pestañas, que son las que suelen haber en la gran mayoría de programas. Para empezar tenemos la pestaña file, en esta pestaña se pueden crear nueva imagen, cargar una creada anteriormente, grabar la actual, y luego importar y exportar elementos que podemos necesitar, ya sea para otras aplicaciones, en nuestro caso, el de unity3d, usaríamos el formato Autodesk FBX. La pestaña add, que nos permite añadir elementos a la escena, entre ellos, figuras geométricas, cámara o luces, elementos básicos en cualquier escenario 3D. La pestaña render que nos permite ver nuestra imagen renderizada(proceso de generar una imagen o vídeo mediante el calculo de iluminación GI partiendo de un modelo en 3D), para acabar window que nos permite partir o modificar lo que vemos y help que tiene diferentes tipos de ayuda.

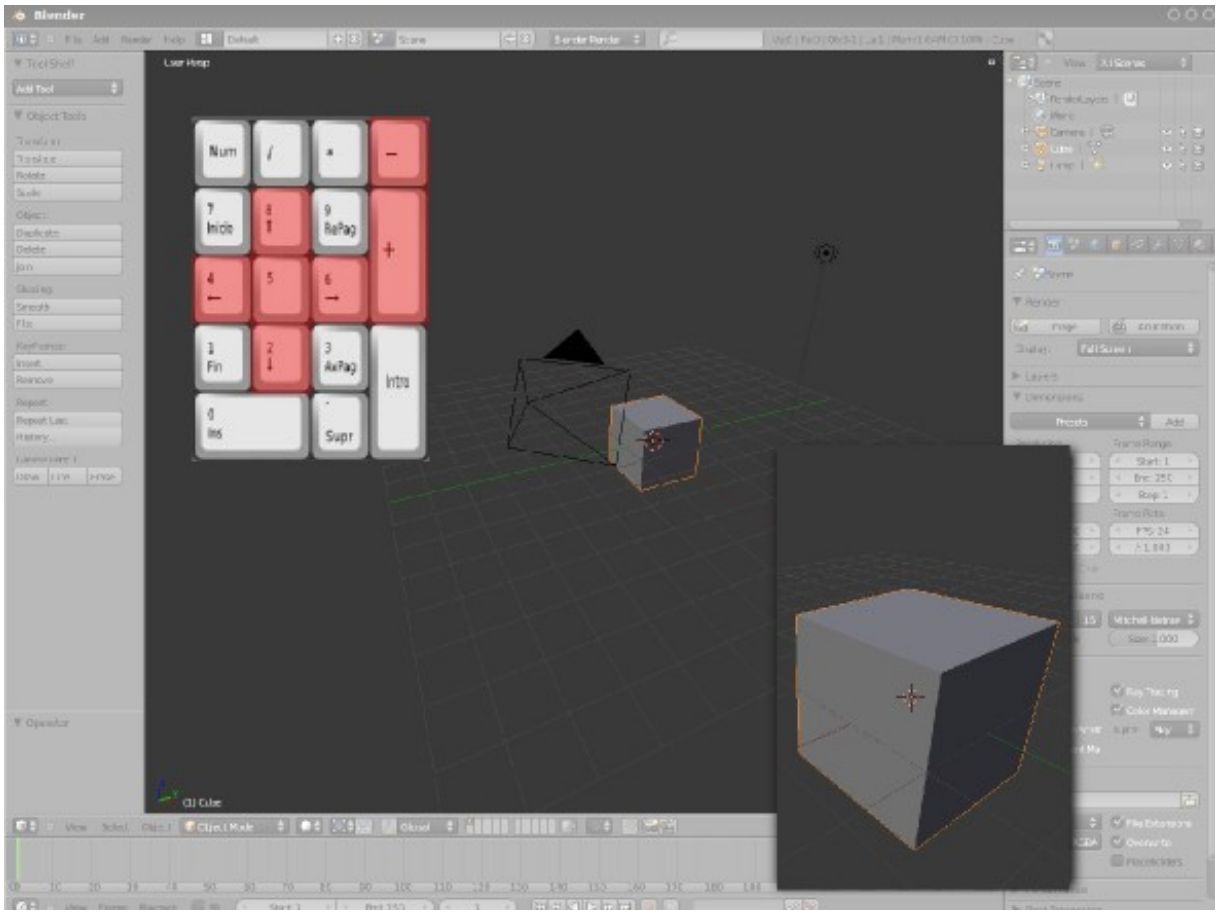


Figura 15 : Blender, Navegando

Al ser un escenario tridimensional, hay que saber moverse(navegar) por el, para ello tenemos para empezar con el teclado numérico, en este se tienen diferentes puntos de visión en los que se puede trabajar, en la imagen el user perspective(fig.15), jugar un poco con esto es importante para empezar a familiarizarnos con el programa. Luego mediante el ratón podemos realizar el efecto órbita(dejando pulsado la rueda y moviendo el ratón) o realizar zoom (moviendo la rueda para arriba o para abajo), luego tenemos el primer botón del ratón que sirve para seleccionar y aceptar las opciones y el segundo botón que sirve para realizar las acciones que tendrán que ser confirmadas con el primer botón del ratón.

Si nos fijamos en la imagen(fig.16), se pueden observar los elementos principales de una escena, por un lado tenemos la cámara, por otro el objeto 3d y para acabar un circulo con un punto en medio que simula un punto de iluminación para la escena. Estos elementos pueden ser modificados al gusto del que crea la imagen, jugar con estos elementos hará que nuestro render final sea totalmente diferente, por lo tanto tener en cuenta la posición que se toman estos elementos sera vital en la creación de nuestra figura 3d.

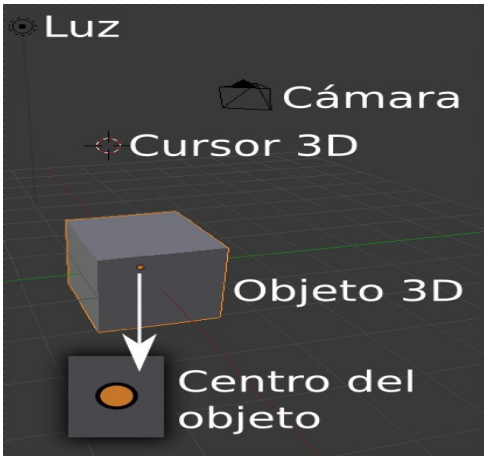


Figura 16 : Blender, elementos principales

Una vez tenemos nuestro cubo en la imagen, y teniendo en cuenta que no se puede explicar todas las opciones posibles a fondo, tenemos que saber que estamos en el object mode, este modo lo que hace es permitirte elegir los elementos de la escena por separado. Si queremos modificar estos elementos, tenemos que seleccionar el que queramos en este object mode usando el ratón, se quedara con un reborde naranja(fig 16), y una vez con ello, pasaríamos al edit mode, o modo para editar nuestra imagen(también se puede realizar mediante la tecla tab).

En este modo editar, podemos elegir los vértices, en este caso de nuestro cubo y estirarlo mediante el ratón hasta donde queramos(raton2->mover, raton1->confirmar), para tener mas precisión de esto, podemos pulsar X, Y o Z, del teclado para mover el vértice solo respecto ese eje. Como apunte, también se puede mover seleccionando en vez de los vértices, rectas o una cara entera(fig.17).Otro tipo de opciones que tenemos en este modo, seria entrar al modo wireframe, para ver la malla que constituye nuestro objeto(pulsando la Z), o poder seleccionar toda nuestra figura(pulsando la tecla A).

Aparte de poder mover nuestros vértices por separado, para crear diferentes tipos de elementos se usan otras técnicas, las mas usadas son :

- Cortar(usando control+r), esto nos permite en el caso del cubo partirlo y a partir de ahí ir creando nuevas secciones.
- Extrudir(tecla E), esto nos permite en toda la área seleccionada extrudirla creando un trozo mas de nuestra figura en la dirección seleccionada
- Rotar(tecla R), esto nos permite rotar nuestra zona seleccionada, con respecto al eje de esta.

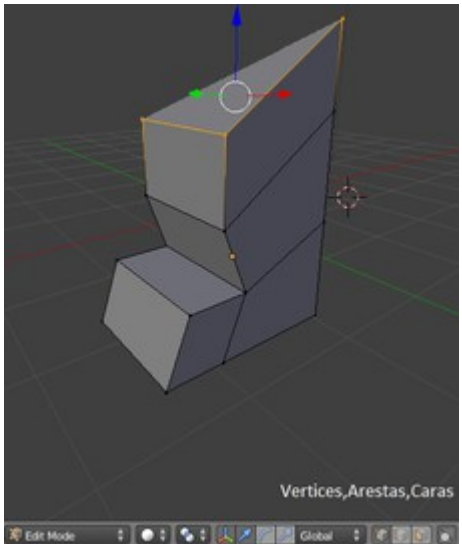


Figura 17 : Blender, figura usando tecnicas

En esta imagen(fig 17), podemos observar, lo que podría ser un pie de un robot, realizando unos sencillos pasos de cortar, extrudir y rotar. Obviamente el programa tiene en este modo editar muchas mas opciones, pero con esta pequeña iniciación y un poco de imaginación podríamos crear ya un buen numero de objetos 3D. Como ya se ha dicho inicialmente, al principio se hace un poco complicado controlar el programa con sus atajos de teclado, pero realmente cuando se llevan realizadas unas cuantas practicas es muy didáctico y fácil de asociar.

Otro de los pasos importantes de este modo editar, y el cual nos ahorrara para inexpertos muchos pasos, sera el de extraer un unwrap de nuestra imagen(teniéndola entera seleccionada pulsar la tecla U sobre ella), esto consiste en obtener una imagen que tendrá todas las caras de nuestra creación, y las podremos pintar a nuestro gusto en cualquier tipo de programa externo, para luego cargarla y ver el resultado(cuidado donde ponéis la iluminación para comprobarlo).

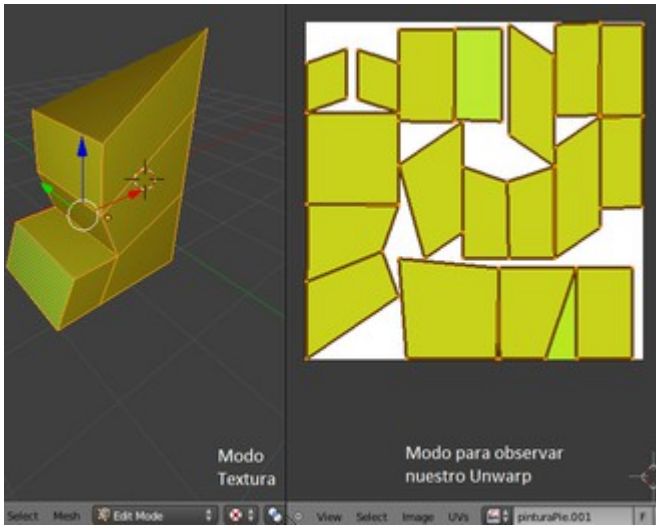


Figura 18 : Figura pintada

3.2.1. Que es Unity3d

Unity3D es un motor de videojuego multiplataforma creado por Unity Technologies.Unity está disponible como plataforma de desarrollo para Windows y OS X, y permite crear juegos para Windows, OS X, Linux, Xbox360, Playstation 3, Wii, Wii U, Ipad, iPhone y Android. Gracias al Plug-In Web de Unity, también se pueden desarrollar juegos de navegador, para Windows y Mac.

Requerimientos minimos para ejecutarlo :

- Windows: XP SP2 or later; Mac OS X "Snow Leopard" 10.6 or later.
- Graphics card with DirectX 9 level (shader model 2.0) capabilities. Any card made since 2004 should work.
- Using Occlusion Culling requires GPU with Occlusion Query support (some Intel GPUs do not support that).

Perfil:

- Plataformas : Windows, OS X, Linux, Xbox360, Playstation 3,Wii , Wii U, Ipad, iPhone y Android.
- Compañía : Unity Technologies.
- Sitio Oficial : Unity3d.com.
- Licencia : De dos tipos, Unity gratuita, con ciertas limitaciones para el uso de estudiantes que quieran empezar a usar el programa, y Unity-Pro de pago.

3.2.2. Características

- **Editor** : Entorno integrado de desarrollo con jerarquía, edición visual, vista previa del videojuego y inspector de propiedades

- **Publicación** : Soporte para publicación en múltiples plataformas

- **Assets** : Los archivos se importan automáticamente y se actualizan al ser modificados. Soporta 3ds Max, Maya, Blender, Zbrush, entre otros.

- **Shaders** : El sistema de shaders de unity, se combina a la perfección entre uso, flexibilidad y rendimiento. Todos los Shaders se integran con cualquier tipo de luz. También se pueden escribir los Shaders con el lenguaje ShaderLab con Cg y GLSL

- **Gráficos** : Optimizados para DirectX y OpenGL. Mallas de animación, sistemas de partículas y sistema avanzado de iluminación y sombras.

- **Terreno** : Se pueden crear paisajes 3D densos, que funcionaran en hardware de gama baja, y de reducido espacio en disco.

- **Red** : Se provee de funcionalidades multijugador, como chat o puntuaciones en linea, aparte de otras necesidades.

- **Audio y Vídeo** : Mezcla en tiempo real gráficos en 3D con streaming de audio y vídeo

- **Scripting** : Soporta tres lenguajes : JavaScript, C# y Boo. Ademas puede usar librerías .Net, con soporte a bases de datos, expresiones regulares, XML, acceso a ficheros y funciones de red.

- **Iluminación y Sombras** : Sistema de iluminación altamente optimizado con lightmaps y sombras en tiempo real.

- **Documentación** : Existe una gran cantidad de conocimiento y soporte que se puede utilizar. Ademas de una comunidad activa y abierta a los usuarios.

3.2.3. Interfaz de usuario

En esta parte se va a hablar del interfaz, por lo tanto con lo que nos vamos a encontrar a la hora de hacer nuestros proyectos.

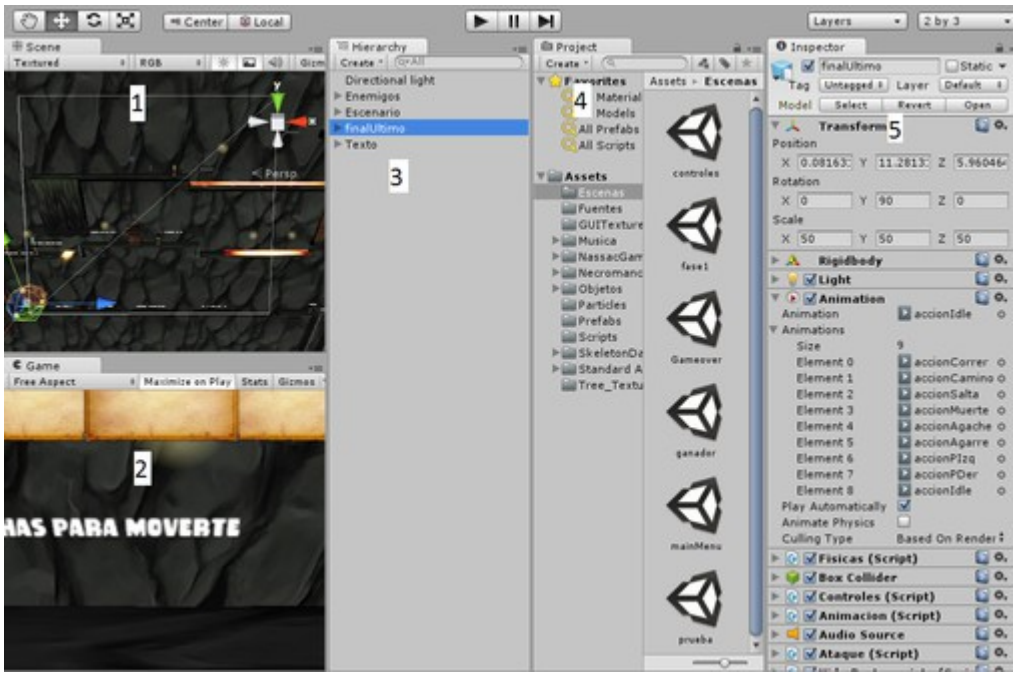


Figura 19 : Interfaz Unity3d

En la imagen superior(fig.19), se puede observar el interfaz de usuario, este viene separado en principalmente los 5 sectores numerados que se utilizaran para realizar nuestro proyecto. Estos son :

– **1 . Vista escena(Scene)**: Es la vista donde construimos visualmente nuestro juego. Para su mejor vista de cara a modificar objetos, se puede pulsar al espacio y pasara a verse en ventana completa.

Los juegos creados en Unity están divididos en "escenas" al igual que muchos motores AAA como Gamebryo y el Unreal Engine usan partes.La vista de escena es un entorno 3D para crear cada escena. Trabajar con la vista de escena, en la forma mas sencilla, sería arrastrar un objeto desde la vista de proyecto a la vista de escena que colocara el objeto en la escena; entonces podrás posicionarlo, escalarlo y rotarlo sin salir de la vista de escena.

Figura 20 : Cambio de perspectiva

Por defecto la vista de escena tiene un perspectiva 3D de la escena. Podemos cambiar esto por un numero de vistas Ortográficas: top down, side y front. En la parte derecha de la vista de escena veréis un "Gizmo"(fig.20) que parece una caja con conos que salen de ella.

Figura 21 : Botones de control

Los botones de control nos permiten realizar cambios en nuestra vista de escena. De izquierda a derecha:

- Hand Tool : Este control nos permite movernos al rededor de la vista
- Translate Tool : Nos permite mover cualquier objeto en ejes x, y, z.
- Rotate Tool : Nos permite rotar cualquier objeto de la escena.
- Scale Tool : Nos permite escalar cualquier objeto de la escena.

– **2 . Vista de juego(Game)** : Es la vista donde obtendremos una previsualizacion de nuestro juego. En cualquier momento se puede reproducir y jugarlo en esta vista.

Figura 22 : Controles de reproducción

La imagen sobre estas líneas muestra los controles de reproducción, que están localizados en la parte superior del editor. Puedes entrar en la previsualización del juego en cualquier momento pulsando el botón de reproducción(el primero por la izquierda), pausar usando el botón de pausa (central) o saltar adelante usando el botón derecho.

Puedes jugar desde la vista de juego o extenderla a pantalla completa. El panel también tiene un menú contextual en la esquina izquierda que aparece como "Free Aspect" por defecto, de esta lista podremos seleccionar un número de proporciones para el juego, lo que es ideal para probar nuestro juego en distintas pantallas y plataformas.

– **4 . Vista de proyecto(Project)** : Esta es la librería de assets, sera donde estará nuestro proyecto, puedes importar objetos, texturas, y crear otros objetos, como scripts o prefabs que se almacenaran aquí. Todos los assets creados o importados se quedaran aquí, para directamente poder insertarlos en tu juego. Por un mejor orden se estructura en carpetas.

En la parte superior de la vista de proyecto veras un botón "Create", que mostrara una lista desplegable con varias opciones de creación. Podremos crear carpetas, scripts, shaders, animaciones y otros tipos de objetos usando este panel. Hacer dos clic sobre un ítem en la librería nos permitirá renombrarlo.Puedes hacer clic y arrastre de carpetas y ítems para organizar la estructura. Puedes importar y exportar paquetes (colecciones de assets) simplemente haciendo clic derecho sobre la vista de proyecto y seleccionando la opción apropiada. Puedes importar assets como archivos de audio, texturas modelos etc. con hacer clic derecho y seleccionar "Import Asset".

– **3 . Vista de Jerarquía(Hierachy)** : Esta vista contiene todos los objetos actuales de la escenarios. La jerarquía también sirve como método rápido y fácil para seleccionar objetos en la escena. Si quieres por ejemplo, seleccionar un objeto de la escena puedes seleccionarlo desde la jerarquía en lugar de moverte por la escena, encontrarlo y seleccionarlo. Cuando un objeto es seleccionado en la jerarquía también lo es en la vista de escena, donde puedes moverlo, escalarlo, rotarlo, borrarlo o editarlo. El inspector también mostrara las propiedades del objeto seleccionado; de esta forma la jerarquía sirve como una herramienta útil para seleccionar rápidamente objetos y editar sus propiedades.

– **5 . Vista de inspector(Inspector)** : Esta vista sirve para una gran cantidad de cosas, ya que es la vista que nos proporciona la información mas detallada de los assets que tenemos, y nos permite modificar-los a nuestro gusto.

El orden de la ventanas no hace falta que sea como el mostrado en la imagen, ya que Unity nos permite modificar a nuestro gusto el interfaz, arrastrando una ventana encima de la otra para poder cambiar el orden de estas entre si.

Por ejemplo, si seleccionas una luz o cámara, el inspector te permitirá editar varias propiedades de la luz o de la cámara. Adicionalmente el inspector sirve como panel de herramientas para ciertos tipos de objetos. Por ejemplo, si seleccionas un terreno el inspector mostrara las opciones de terreno y también el editor con herramientas como esculpir, texturizar, etc.

3.2.4. Primeros pasos

En esta parte se van a explicar los pasos mínimos básicos para realizar nuestro juego, estos son:

- 1 . Crear nuevo proyecto
- 2 . Crear primera escena
- 3 . Añadir suelo a la escena
- 4 . Añadir cielo a la escena
- 5 . Añadir luces
- 6 . Añadir control en primera persona

- **1 . Crear nuevo proyecto** : Para crear un nuevo proyecto hay que irse a File → New Proyect, esto hará que se muestre el dialogo de creación de proyecto(fig.23).

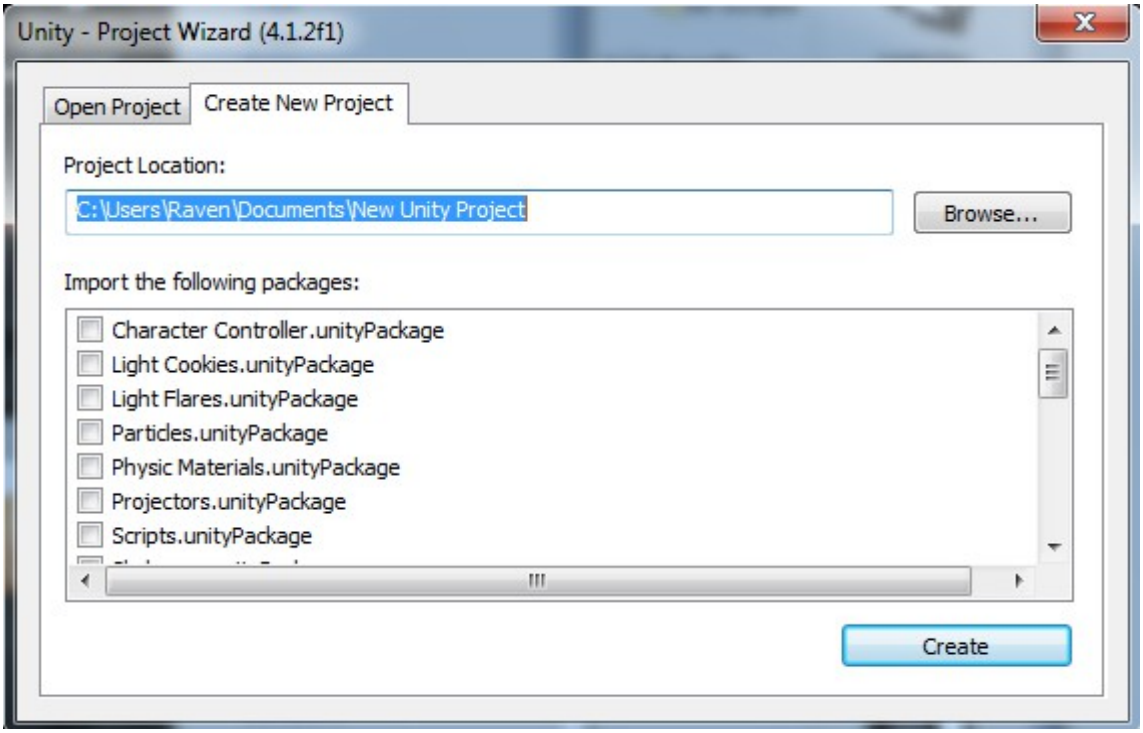


Figura 23 : Nuevo proyecto

- Lo primero que tendriamos que hacer es seleccionar la direccion donde queremos nuestro proyecto, usando el boton Browse.
- Lo siguiente seria elegir que paquetes por defecto o importados querrias agregar a tu proyecto.
- Cuando se realizan estos dos pasos, unity se reinicia y te crea la estructura de archivos necesaria. Obteniendo el interfaz visto anteriormente.

- **2 . Crear la primera escena** : Una vez creado el proyecto por unity, solo tendríamos que crear una escena nueva pulsando File → New Scene. Una vez creada la escena podríamos grabarla con File → Save Scene. Como nota para futuros proyectos, en el caso de necesitar diferentes escenas para nuestro juego, se agregan en File → Build settings, en esta ventana nos pone las escenas que tenemos actualmente (scenes on built) y podemos añadir nuevas usando Add Current.

- **3 . Añadir suelo a la escena** : Ya tenemos nuestro proyecto y la escena en blanco, ahora vamos a añadir un terreno para empezar a crear nuestro nivel.

  - Mientras creamos el terreno veremos algunas características y opciones básicas de los terrenos. Unity maneja los terrenos de la misma forma que muchos otros motores, como una malla plana que podemos texturizar y esculpir sin salir del editor. Para insertar un nuevo terreno vamos a "Terrain -> Create Terrain" desde el menú principal.

  - Lo que tenemos ahora no es particularmente bonito. Si no puedes ver el terreno, desactiva las luces en la vista de escena. También podrás apreciar que el terreno aparece en la jerarquía y un asset se ha añadido a la librería en la vista de proyecto.

  - Una vez con el suelo creado, podemos modificar sus características basicas en "Terrain -> setResolution", tales como altura, anchura...

  - Luego pasaremos al inspector en el podemos ver una barra de herramientas . En el tenemos tres partes. La primera Transform nos permite mover y escalar nuestro suelo, la segunda Terrain Script nos ofrece herramientas para para modificar nuestro terreno, como podria ser levantar o hundir el terreno, pintar texturas del terreno, colorcar arboles etc... y el tercero Terrain Collider, que tiene las propiedades de collision de nuestro terreno. En la imagen de la siguiente pagina(fig. 24) se puede observar.

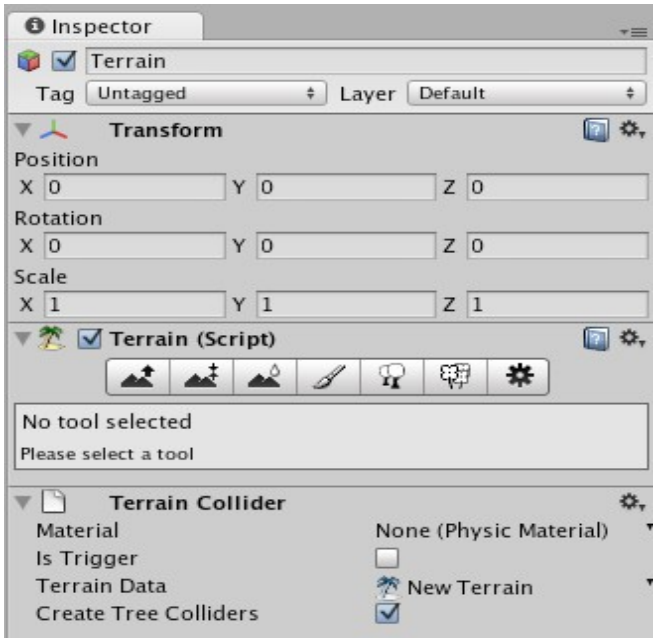


Figura 24 : Inspector de suelo

- **4 . Añadir cielo a la escena** : añadir el cielo a la escena es realmente sencillo, solo hay que colocar una textura, esta es llamada Skybox material, para añadirla a nuestro proyecto, hay que ir a las opciones "Edit -> Render Settings", cuando pulsamos, nos sale el inspector de la imagen inferior (fig 25), solo habria que añadir en la opcion Skybox Material, un cielo que nos gustaria para nuestro proyecto, y lo demas son opciones para modificarlo con mas niebla o menos, o otras opciones.

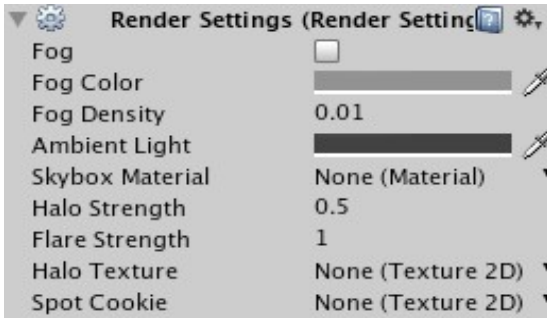


Figura 25 : Inspector para añadir cielo

- **5 . Añadir luces** : Como hemos explicado en temas anteriores, tenemos que añadir una luz(mínimo) para nuestra escena, para añadirla hay que ir al menú "GameObject → Create Other → Directional Light". No importa donde coloquemos la luz, ya que su efecto es sobre toda la escena(actúa como si fuera un sol artificial), lo único que realmente si importa, es la dirección, ya que según esta, afectara de una manera o otra para las sombras y diferentes colores de las texturas. Como siempre, desde el inspector se pueden modificar todas estas cosas y mas (fig 26)

  Para un mejor rendimiento y también la mejor calidad en iluminación se debe considerar hacer un bake de los lightmaps, de hecho esta debe ser una practica estándar en la producción de cualquier juego.

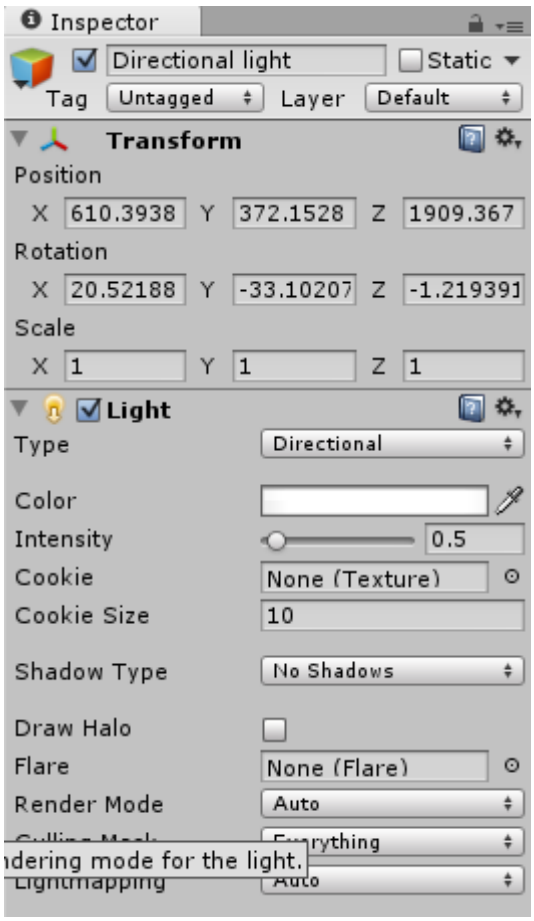


Figura 26 : Inspector Luz

– **6 . Añadir control en primera persona** : Para crear un elemento que el usuario pueda controlar en primera persona(también esta para tercera persona), viene un prefab en el unity3d para añadirlo y ya tener un elemento capsula que pueda realizar movimientos. Para realizar esto habría que ir a "Standart Assets → Prefabs". Dentro de esa carpeta existe un prefab llamado "First Person Controller". Ahora solo habría que poner la capsula encima del terreno, y ya lo tendríamos hecho.

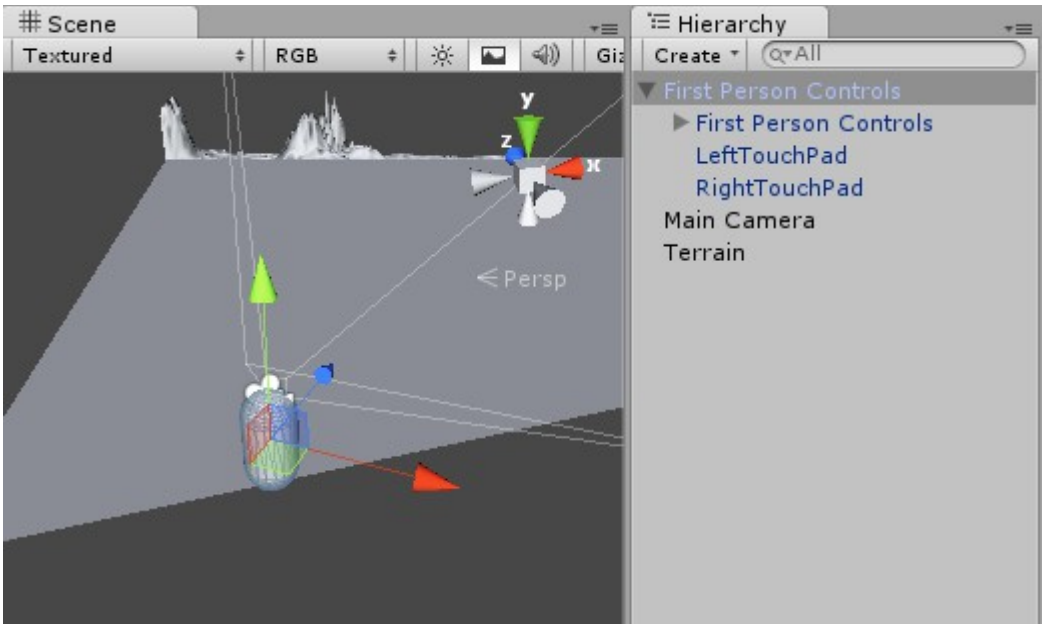


Figura 27 : Prefab primera persona

3.2.5. Preparando built del juego

Para crear la versión ejecutable de nuestro proyecto, tenemos que irnos a "File-> Build Settings". Encontrándonos en la imagen siguiente.

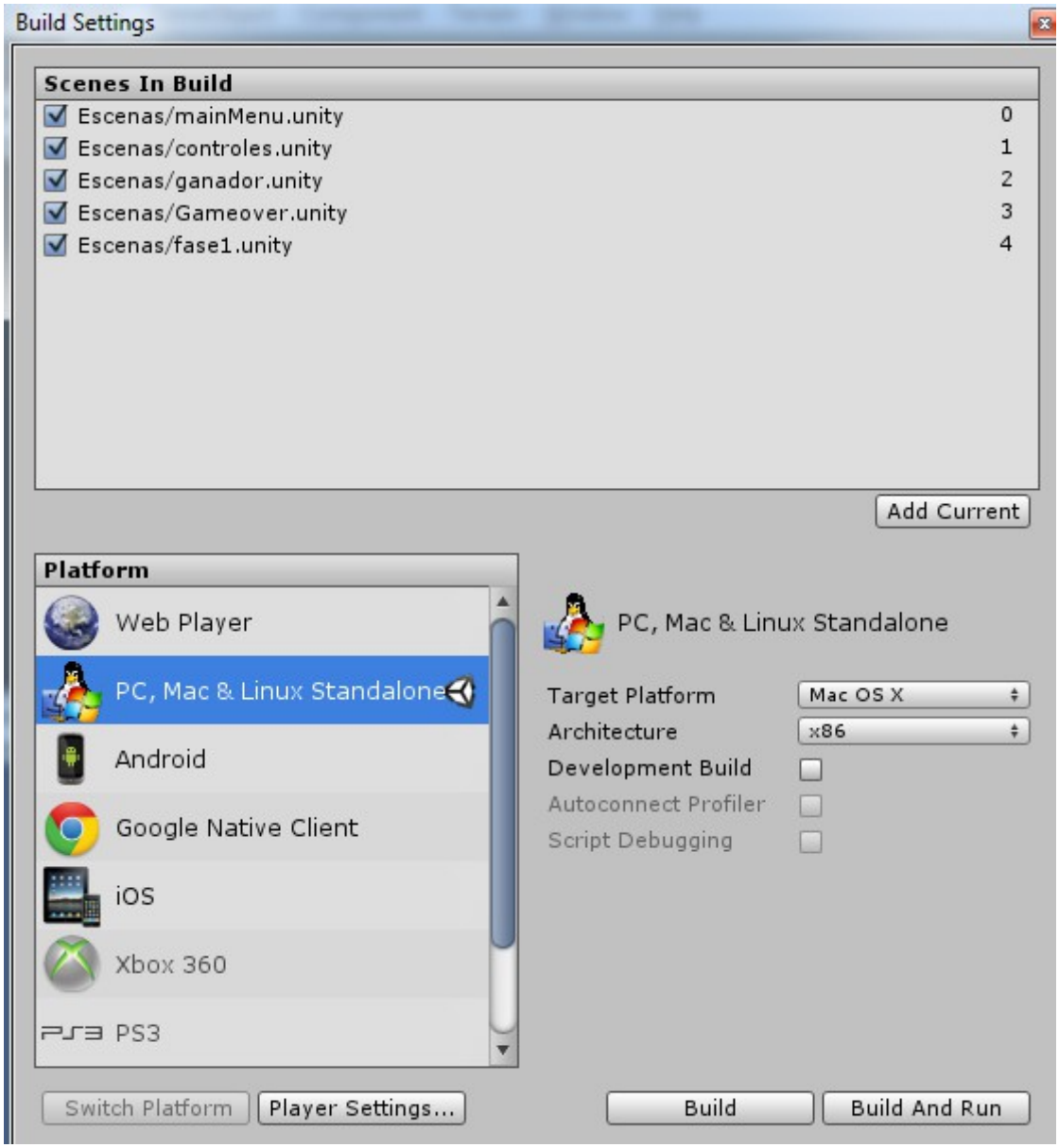


Figura 28 : Creando versión ejecutable

En la imagen anterior(fig.27), se puede observar en la parte superior, que están colocadas las escenas que tendrá nuestro proyecto, y luego se nos permite crear diferentes distribuciones(algunas limitadas a la versión de pago como podrían ser la versión PS3 o Xbox360).

Crear una versión para windows por ejemplo, es tan sencillo como colocar el sistema operativo que queremos, que tipo de arquitectura usara y pulsar la tecla Build o Build and Run(si queremos probarlo directamente). También se pueden modificar configuraciones iniciales, como resolución inicial, controles, etc en el botón player settings. En el caso que quisiéramos hacer un Webplayer se seguiría el mismo proceso.

CAPITULO 4: VIDEOJUEGO "TIME IS ENDING"

En este tema se explicaran el objetivo del proyecto, la estructura de archivos restantes, bocetos, escenas y imágenes finales conseguidos.

4.1. Objetivo

El objetivo es la creación de un videojuego 2.5D usando Unity3d como motor gráfico, y el programa Blender como diseño gráfico.

- El juego sera un juego de plataformas convencional

- El usuario podrá controlar un personaje principal, y tendrá que superar diversas trampas/obstáculos/enemigos para finalizar el juego

- En el juego se pide una versión demo, en la cual solo habría los menús principales, una pantalla jugable, y como se gana o se pierde

- Ha de intentar añadir el máximo de conocimientos posibles, y de opciones posibles en el programa pedido

- A nivel gráfico/artístico, se pide el mínimo aceptable, y ningún tipo de creación foto realista o de alto nivel artístico, ya que para los estudios que se intenta ofrecer este trabajo son estudios de informática

- Se tiene que ser capaz de superar todo tipo de problemas autogestionándose para buscar soluciones dichos problemas.

4.2. Bocetos

Uno de los elementos mas importantes, son los objetos que se encontraran en nuestro videojuego, como pueden ser personajes, figuras, plataformas, enemigos, etc …

Uno de los ejemplos que muestra uno de los objetos creados es el que se muestra a continuación.

En este caso se muestra el objeto mas importante de nuestro juego, se trata del personaje protagonista, para realizarlo se crearon y aplicaron bocetos en blanco y negro, y color para su realización en Blender.

Figura 33 : Bocetos de los personajes, blanco y negro , en color y en version 3d Blender ya utilizado dichos bocetos

Otros ejemplos de objetos que podemos encontrar en el videojuego, serian:

- Caja, Checkpoint, Skeleton, Lampara, ParedH(Horizontal), ParedV(Vertical), etc ...

4.3. Estructura y disposición de objetos

En esta sección se puede ver un ejemplo de la Jerarquía utilizada en nuestro juego, la Jerarquía en si, agrupa los objetos creados anteriormente, utilizados en el escenario que queremos crear, y en el que luego jugaremos.

Como se puede ver mis objetos han quedado ordenados en 5 subgrupos, estos son, el de iluminación, enemigos, escenario, finalultimo(personaje principal) y texto.

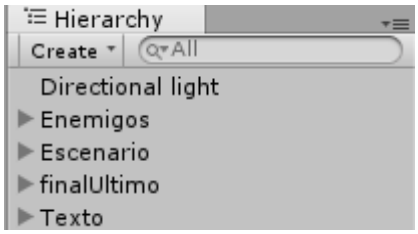


Figura 31 : Jerarquia basica de mi videojuego

- El **iluminación**, contiene una unica luz que ilumina todo el escenario.

- El **enemigos**, contiene todos los enemigos que vayan a aparecer en nuestra escena, estos se diferencian de los demas tipos de objetos por necesitar inteligencia artificial

- El **escenario**, contiene todos los elementos que suponen el escenario donde estamos, plataformas horizontales, verticales, cajas y mas elementos que estan en el escenario.

- El **finalUltimo**, contiene nuestro personaje protagonista, con todo lo que necesita y le pertecenece, como en este caso es la camara principal, y el esqueleto que le permite realizar animaciones.

- El **texto**, contiene diferentes tipos de texto que se muestran tambien en el escenario a modo de tutorial, indicandonos que realizaremos, cada texto es un "consejo".

En la imagen que tenemos a continuación podemos ver la jerarquia creada de manera expandida con todos los elementos visto anteriormente.

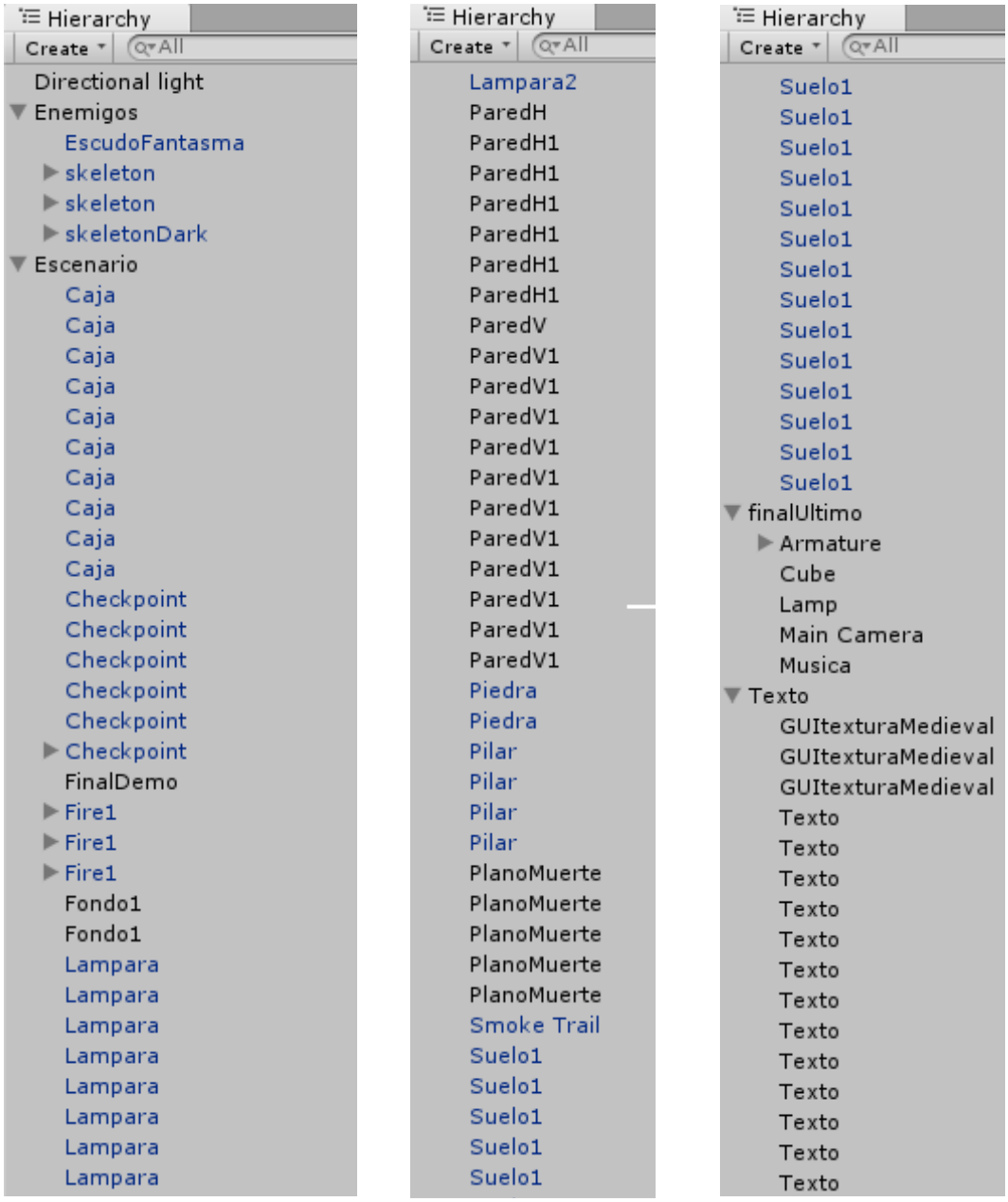


Figura 32 : Jerarquia extendida de mi videojuego

- De la imagen hay que apuntar que hay varios objetos con el mismo nombre, estos objetos sencillamente son copias de otros, por lo tanto realmente no hay tantos objetos diferentes, pero hay una gran cantidad porque son muchos copias de otros. Por ejemplo tenemos 10 veces el nombre caja en nuestra jerarquía de objetos, esto lo único que quiere decir es que tenemos 10 objetos del tipo caja, con las mismas características pero que pueden tener diferentes posiciones (x,y,z), o dimensiones, en nuestro escenario.

4.4. Escenas

Una vez tenemos una jerarquía de objetos añadida a nuestro proyecto, esto se le pasa a llamar escena,que nos servirá para poder cambiar de escena en escena según nos interese en el juego, en el caso anterior toda nuestra jerarquía de objetos supone la escena "fase1", que veremos a continuación, junto a las otras escenas que tenemos en el juego, son 5: mainMenu, controles, fase1, ganador y gameover.

- **mainMenu** : esta escena contiene todo el menú principal, es el que nos permite iniciar o reiniciar el juego.

- **Controles** : esta escena contiene la parte del menú donde podemos observar los controles de nuestro videojuego.

- **fase1** : esta escena contiene la pantalla de nuestra demo, en la cual podemos actuar con nuestro personaje para conseguir finalizar el nivel superando obstáculos y enemigos.

- **Ganador** : esta escena aparecerá en el momento que completemos nuestra pantalla, nos muestra una felicitación y nos deja volver al menú principal.

- **Gameover** : esta escena sera igual que la anterior, pero en este caso nos dará el mensaje de derrota.

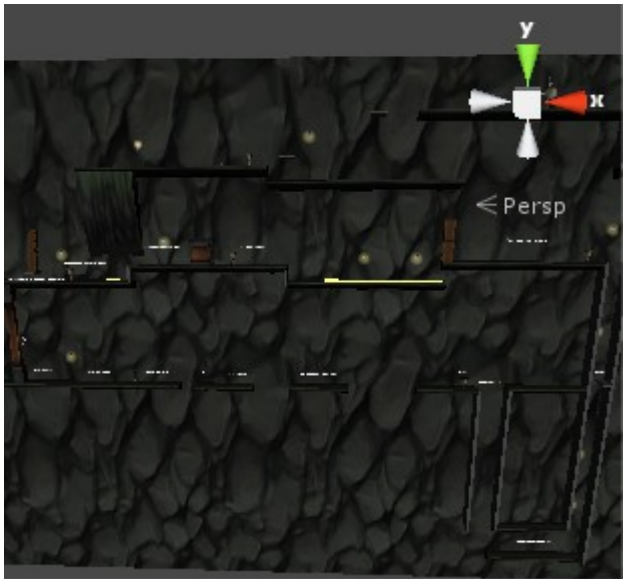
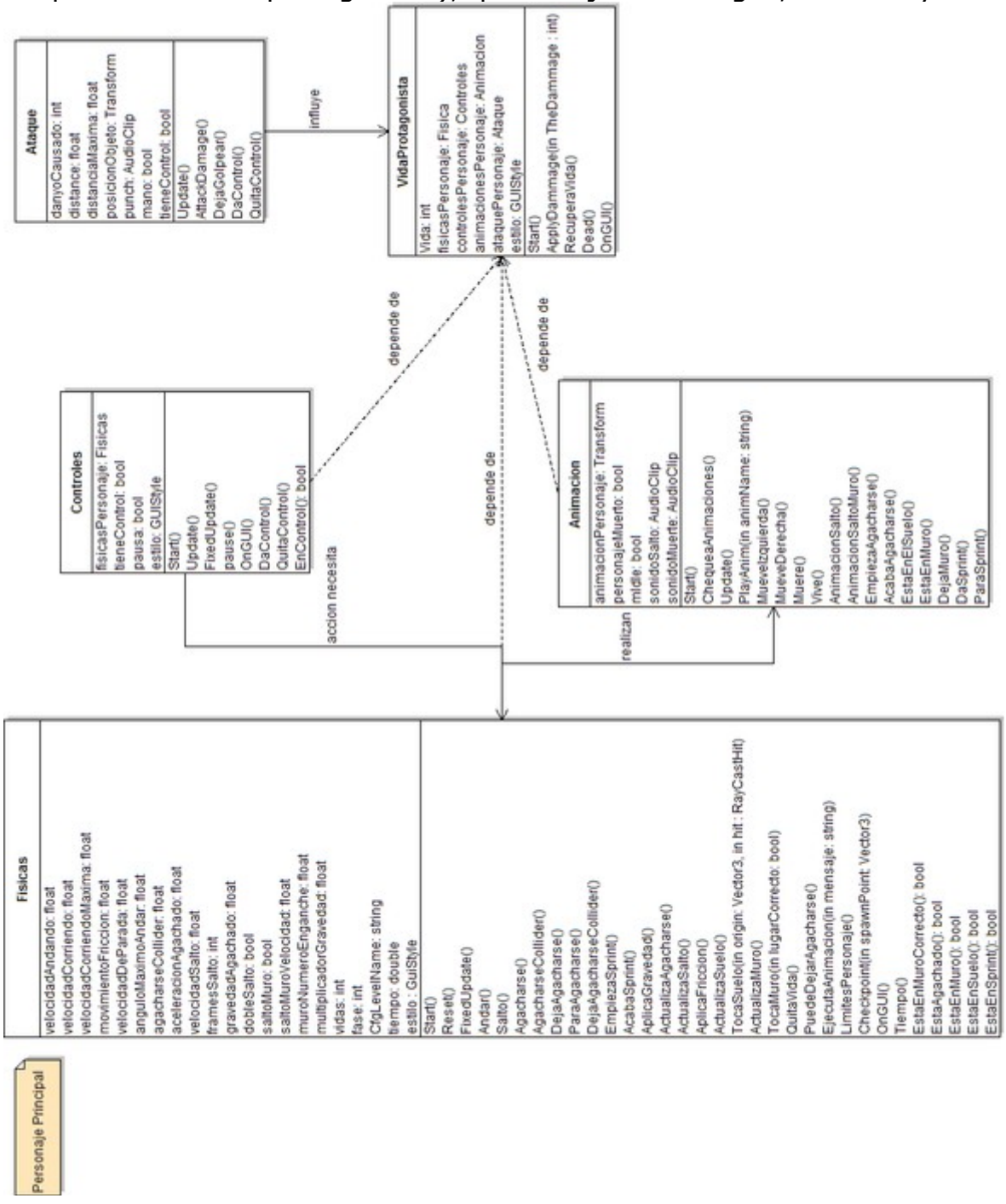


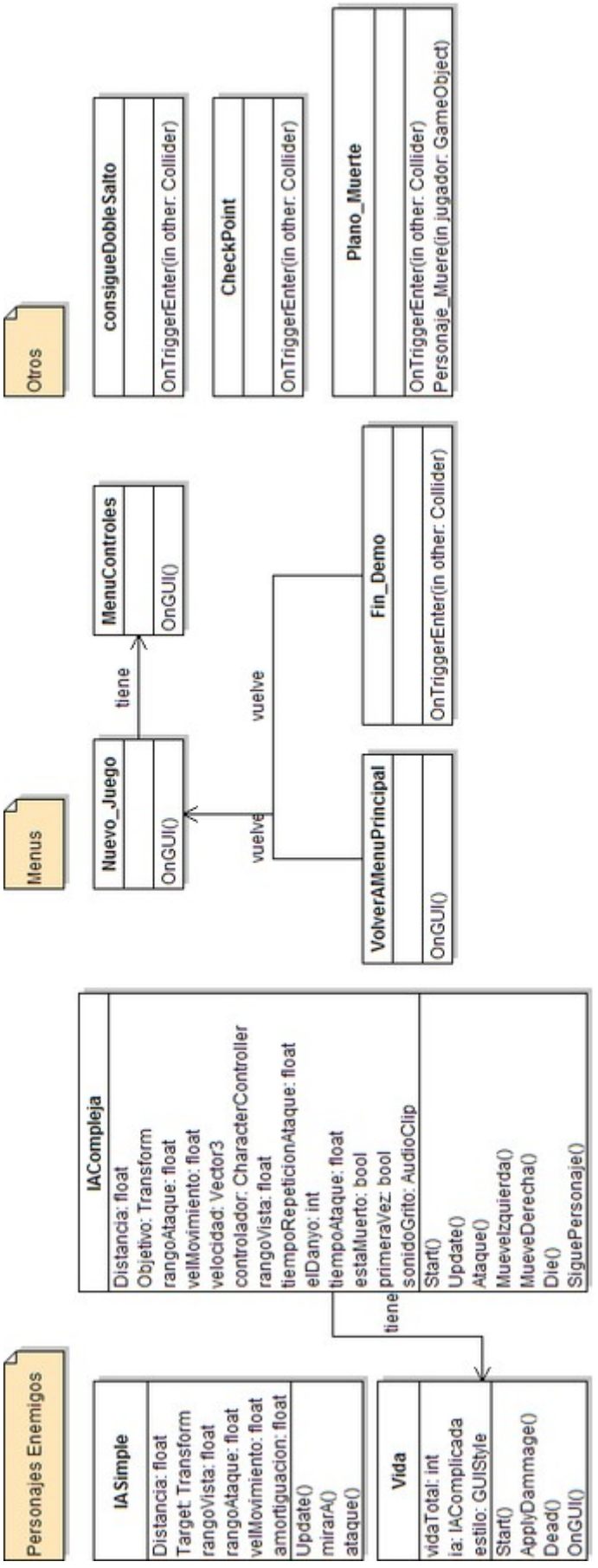
Figura 29 : Escenario fase1 en su vista de escena

4.5. Diagrama de Clases

Otra de las cosas a destacar del trabajo realizado, es la parte de programación en el lenguaje C#, este seria el diagrama de clases de nuestra aplicación, contiene cuatro grupos de scripts diferenciados. Esta parte incluye todo lo que hará que nuestros objetos puedan interactuar entre ellos, y que según en que situaciones pueda pasar una cosa o otra.

Estos grupos son: personaje principal(contiene todos los scripts atados al protagonista), personajes enemigos, menús y otros.




Otros
consigueDobleSalto
OnTriggerEnter(in other: Collider)
CheckPoint
OnTriggerEnter(in other: Collider)
Plano_Muerte
OnTriggerEnter(in other: Collider)
Personaje_Muere(in jugador: GameObject)
Menus
MenuControles
OnGUI()
tiene
Nuevo_Juego
OnGUI()
vuelve
vuelve
Fin_Demo
OnTriggerEnter(in other: Collider)
VolverAMenuPrincipal
OnGUI()
Personajes Enemigos
IACompleja
Distancia: float
Objetivo: Transform
rangoAtaque: float
velMovimiento: float
velocidad: Vector3
controlador: CharacterController
rangoVista: float
tiempoRepeticionAtaque: float
elDanyo: int
tiempoAtaque: float
estaMuerto: bool
primeraVez: bool
sonidoGrito: AudioClip
Start()
Update()
Ataque()
MueveIzquierda()
MueveDerecha()
Die()
SiguePersonaje()
tiene
IA Simple
Distancia: float
Target Transform
rangoVista: float
rangoAtaque: float
velMovimiento: float
amortiguacion: float
Update()
mirarA()
ataque()
Vida
vidaTotal: int
ia: IAComplicada
estilo: GUIStyle
Start()
ApplyDammage()
Dead()
OnGUI()

4.6. Scripts

Para realizar el juego se han tenido en cuenta los siguientes scripts, estos se asignan directamente al objeto que lo requiere de manera manual desde nuestro interface gráfico, y se ejecutan ininterrumpidamente usando la función Update(). Las animaciones son 15: Fisicas, Controles, Animacion, Ataque, VidaProtagonista, Vida, IASimple, IAComplicada, Nuevo_Juego, MenuControles, VolverAMenuPrincipal, Camara_Sigue_Personaje, Checkpoint, PlanoMuerte y ConsigueDobleSalto.

- **Fisicas**: este script permite controlar las fisicas de nuestro personaje, que nos otorgaran la capacidad de interactuar con el mundo ya creado.La parte mas importante de este script, es que se basa en controlar nuestra caja previamente asignada de collisiones desde el interfaz grafico. Esta caja que puede ser de diferentes formas(cubica, rectangular, esferica, ...), nos ayuda a saber el tamaño de contacto de nuestro objeto con el mundo.

  Sabiendo esto, se ha aplicado la tecnica recomendada por Unity3d para calcular las distancias, Raycast.

---

function **Raycast** (origin : Vector3, direction : Vector3, distance : float = Mathf.Infinity, layerMask : int = kDefaultRaycastLayers) : boolean)

Esta funcion crea un rayo que va contra todos los otros colliders de la escena y nos devuelve si hemos colisionado o no con otro objeto con un collider asignado.
Esta funcion se ha usado en el codigo principalmente para localizar en que situacion estamos, un ejemplo claro, es lanzando un rayo que parte de la parte baja de el collider de mi personaje hacia abajo, para saber si hay, o no, suelo debajo suyo.

Figura 29: Modo debug, las lineas verde y rosa simulan el raycast y el collider cubico que contiene nuestro personaje

- **Controles**: este script permite controlar la interacción de nuestros periféricos (ratón, teclado, …), con nuestro juego. Y es el que se encarga de traducir una tecla pulsada, para que nuestro personaje realice una acción. Ademas este script también se ocupa de pausar nuestro juego.

  Para obtener la tecla pulsada Unity3d usa el comando.

---

function **GetKeyUp** (name : String) : boolean

Ademas Unity3d nos ofrece diferentes opciones, aparte del GetKeyUp, que es la función que obtiene la tecla pulsada una vez se suelta, también están otras funciones como GetKeyDown que se ocupa de tratarlo nada mas pulsarlo, o GetKey que la obtiene dejándola pulsada de manera repetitiva.

También se ha dicho que se trata la pausa en este script, para realizar-lo en unity, se puede controlar el tiempo directamente con un atributo creado especialmente para ello.

---

static var **timeScale** : float

Modificando este parámetro Time.timeScale = 0, podemos parar el tiempo de ejecución, y dejarlo pausado, para volverlo a dejar a velocidad normal solo habría que modificar el parámetro a 1, cabe destacar que se pueden realizar movimientos de slowmotion poniendo valores intermedios entre 0 y 1.

- **Animacion**: Este script es el que se ocupa de comprobar las animaciones cargadas previamente en nuestro objeto de manera automática mediante un archivo .fbx, ver que están todas las necesitadas por nuestro personaje, y luego ejecutarlas cuando sea necesario, esta clase sera llamada por las otras según necesite una animación o otra, englobando todas las animaciones de nuestro personaje, ademas de asignarle los sonidos que ejecutaran algunas de estas acciones.

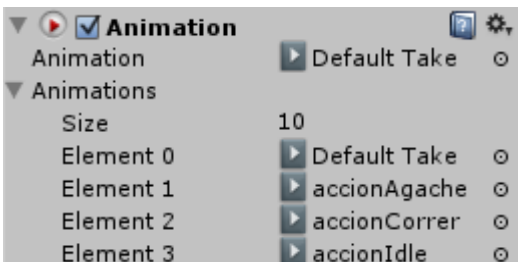


Figura 30 : Ejemplo de animaciones cargadas

La variable animation nos permite tratar directamente con estas las animaciones cargadas en nuestro objeto, permitiéndonos ejecutarlas, cambiar su velocidad, o realizar efectos de animar para alante-atras estilo yo-yo, ademas de otras posibilidades.

Un ejemplo de su uso en mi código seria:

```
animacionPersonaje.animation.Play("accionSalta");
```

Este código ejecutaría la animación "accionSalta".

Para la parte de los sonidos, hay creado un tipo de objeto contenedor de sonidos para ello, este es AudioClip.

Una vez creada nuestra variable con este tipo de objeto, podemos ejecutarlo usando PlayOneShot(sonido);, con esto se ejecutara una vez, para mas veces o repeticiones también hay mas opciones añadidas.

Para usar la parte de sonido, hay que añadir a nuestro objeto que vaya a cargar estos sonidos, un audioSource, al igual que añadimos anteriormente las cajas de collisiones, que cubrirá lo mismo que estas pero en este caso con todo lo relacionado con el sonido.

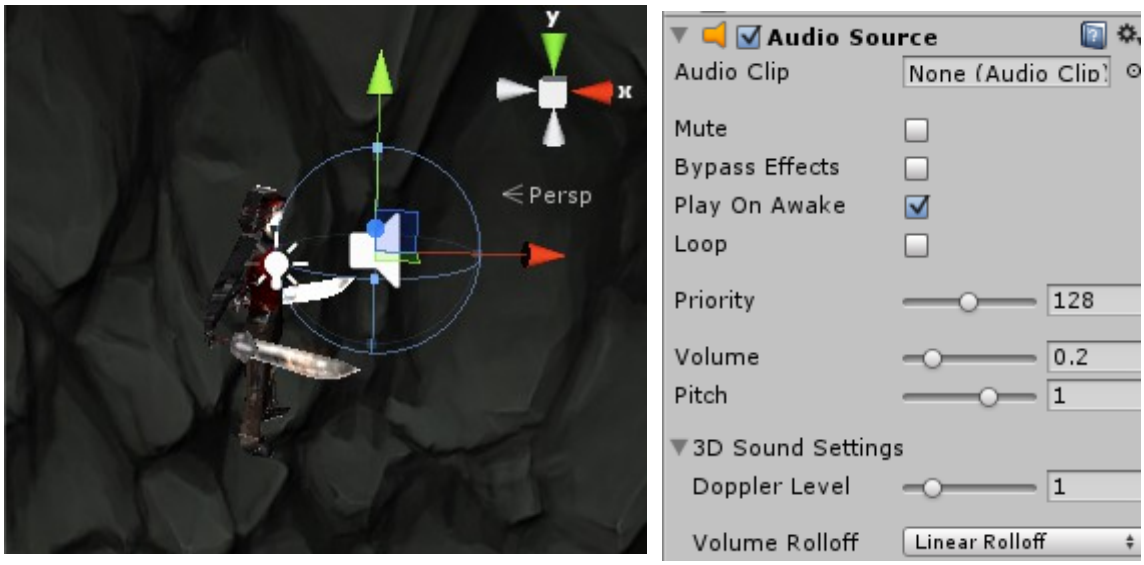


Figura 31 : AudioSource asignado a nuestro personaje

- **Ataque**: Este script es el que nos controla todo lo relacionado con que pueda atacar nuestro personaje, para saber si tenemos un objeto golpeable delante, se realiza igual que anteriormente en las físicas, usando raycasting, en el caso de colisionar enviara un mensaje que contendrá el daño realizado, este mensaje se lanza sin necesitar una respuesta.

```
hit.transform.SendMessage("ApplyDammage",daño,SendMessageOptions.DontRequireReceiver);
```

  En este caso, que es de mi código, lanza el mensaje al objeto que ha previamente colisionado con el raycast. Este ya elegirá si puede recibirlo o no, y lo que implica en el.

- **VidaProtagonista**: Este script es el que controla la vida actual de nuestro personaje, con una simple variable vida máxima.

  También trata el que pueda recibir los ataques enviados de el estilo mostrado anteriormente "SendMessage" anterior.

  En el caso que nuestro personaje pierda toda la vida, este script también se ocupa de que se "Muera" y todo lo que esto supone.
  Para realizar una muerte mas realista, se controla el tiempo con un tipo de método dinámico, que nos permite dar una pausa interna usando el comando **yield**, para dar tiempo a que la animación de muerte acabe.

```
IEnumerator Dead(){
        yield return new WaitForSeconds(1.0f);
}
```

- **Vida**: Realiza exactamente lo mismo que el anterior, pero para los enemigos. Con la diferencia que cuando el enemigo muere, sencillamente se destruye el objeto y se saca de escena para ahorrar memoria.

```
Destroy (gameObject);
```

  Funciona directamente con la IAComplicada, ya que la IASimple no requiere de daño, estos scripts se muestran a continuación.

- **IASimple**: Este script es el que almacena la IA (inteligencia artificial) de un enemigo, este enemigo lo único que hará, es perseguirnos a partir de una distancia mínima y intentar empujarnos.

  Para realizar-lo, en vez de usar raycasting se puede también para ahorrar cálculos, utilizar en estos casos mas sencillos la función de Distance, esta función nos permite eligiendo el objeto que tiene el script, localizar a que distancia esta de un objeto objetivo elegido.

```
Vector3.Distance(Target.position, transform.position);
```

En el caso de que se llegue a la distancia que el objetivo te pueda ver, se ejecuta otro comando, que permite al objeto mirar a una dirección.

```
Quaternion.LookRotation(Target.position - transform.position);
```

En el caso que llegara a la distancia mínima que el rival, ya puede empezar a moverse, se trasladan con la opción Translate.

```
transform.Translate(Vector3.forward*velMovimiento*Time);
```

Ademas para comprobar que funcionaba correctamente se uso cambios de color en el material, con los colores de un semáforo.

```
renderer.material.color = Color.yellow;
```

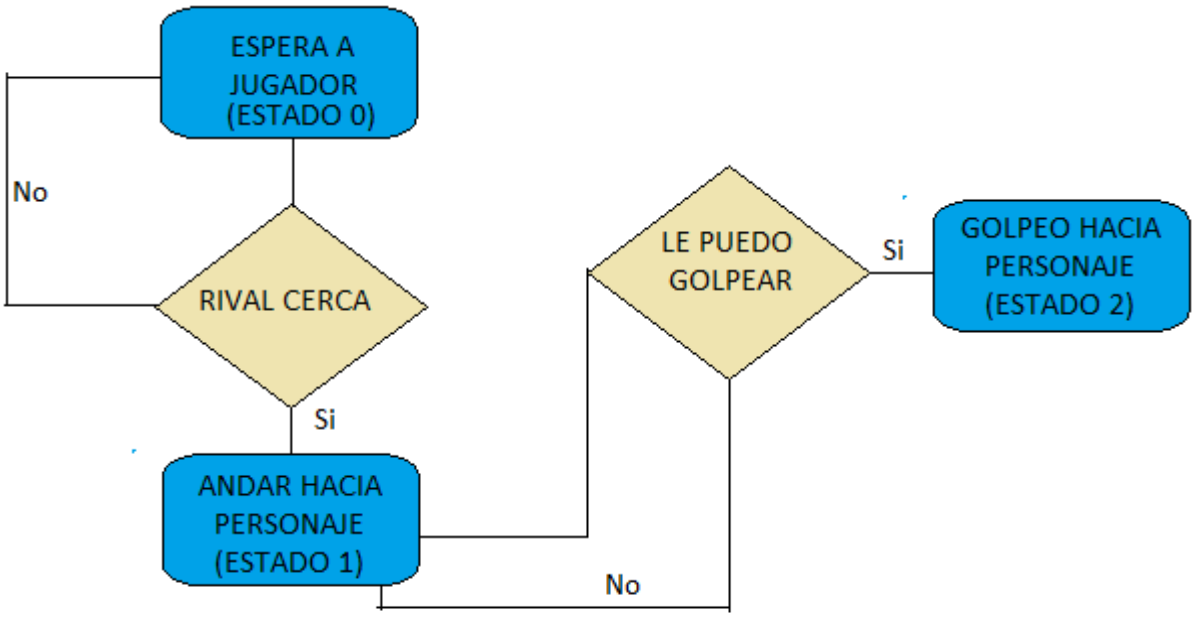
Figura 32 : Prueba/Debug con cambio de material según distancia

- **IAComplicada**: Este script trata como el anterior de dar IA a nuestros enemigos, en este caso la IA también le permite atacar al enemigo, en vez de solo perseguir a nuestro usuario.

  El script es como el anterior, pero en este caso el enemigo nos perseguirá por el suelo, por ello se moverá en el eje de las X persiguiéndonos según estemos a un lado suyo o al otro calculado con la anterior Vector3.Distance.

  Para que el enemigo nos ataque usara el SendMessage a partir de una distancia mínima con respecto a nosotros, que si todo funciona bien lo recibirá nuestro script vidaProtagonista.

  Maquina de estados del enemigo:



  La maquina de estados de la IASimple, seria la misma que esta contando solo ESTADO 0 y ESTADO 1.

- **Nuevo_Juego**, **MenuControles**, **VolverAMenuPrincipal**: los tres son scripts relacionados directamente con menús, estos son muy similares y se basan en lo mismo, en el uso del método OnGUI.

  Este método es el que nos permite crear menús de manera muy fácil. Como por ejemplo labels o botones.

function **Label** (position : Rect, text : String) : void
function **Button** (position : Rect, text : String) : boolean

- **Camara_SiguePersonaje**: Este script, lo que hace es transformar nuestra cámara, en una cámara de un videojuego 2D, haciendo que nos siga al personaje principal desde uno de sus laterales.

  Se realiza obteniendo el **Transform** (esta variable nos permite obtener la posición del objeto al que esta asignada en el espacio), y mediante esto modificar las tres coordenadas (x,y,z), a cierta distancia para que la cámara quede como buscamos.

- **Checkpoint**: Este script, nos realiza la creación de un checkpoint, un punto de control en el cual si previamente morimos, podremos reaparecer, para no tener que iniciar la pantalla desde el principio.

  Este script se basa principalmente, en tocar un objeto en pantalla(el checkpoint), y en ese momento, ese punto pasara a ser nuestro punto de inicio ya asignado previamente en las físicas.

  Para obtener cuando nuestro personaje a colisionado, o pasado por encima de nuestro objeto, unity3d utiliza la función.

void **OnTriggerEnter**(Collider other)

  Esta función como su nombre indica, nos accede cuando un Collider, que no sea al que el script esta asignado directamente, en este caso pasaríamos a actualizar con el actual transform nuestra posición inicial.

- **Plano_Muerte**: Este script, nos delimita los margenes del mapa, y nos permite asignarlo a un objeto que si nuestro usuario lo toca, morirá directamente.

  Funciona igual que el anterior con un OnTriggerEnter, y ademas realiza la acción de muerte, con el anteriormente explicado IEnumerator (método dinámico de StartCroutine).

- **ConsigueDobleSalto**: Este script, seria un script de un ítem que podríamos coger en nuestra pantalla, un power-up, que haría que nuestro personaje obtuviera la capacidad de dar doble saltos. Realmente el script lo único que hace es activar una función desactivada ya previamente creada en las físicas.

  También funciona como los dos anteriores, con un OnTriggerEnter, pero en este caso el objeto al ser tocado se destruye con el destroy(gameObject).

4.7. Screenshots


FASE 1
VIDAS RESTANTES 3
TIEMPO DE JUEGO 6.44
VIDA ACTUAL 100 DE 100


FASE 1
VIDAS RESTANTES 3
TIEMPO DE JUEGO 15.04
VIDA ACTUAL 100 DE 100
PASTE CALOR? REFRESCATE UN POCO
S SIN PENSARTELO TE ARREPENTIRAS

CAPITULO 5: PROCESO DE DESARROLLO

5.1. Modelo iterativo e incremental

Para llevar a cabo el desarrollo del juego "Time is ending", se van a mostrar las etapas que se han seguido para el desarrollo del juego.

En un desarrollo iterativo e incremental el proyecto se planifica en diversos bloques temporales(en el caso de Scrum de un mes natural o hasta de dos semanas, si así se necesita) llamados iteraciones. Las iteraciones se pueden entender como miniproyectos: en todas las iteraciones se repite un proceso de trabajo similar (de ahí el nombre "iterativo") para proporcionar un resultado completo sobre producto final, de manera que el cliente pueda obtener los beneficios del proyecto de forma incremental. Para ello, cada requisito se debe completar en una única iteración: el equipo debe realizar todas las tareas necesarias para completarlo (incluuyendo pruebas y documentación) y que esté preparado para ser entregado al cliente con el mínimo esfuerzo necesario. De esta manera no se deja para el final del proyecto ninguna actividad arriesgada relacionada con la entrega de requisitos.

En cada iteración el equipo evoluciona el producto(hace una entrega incremental) a partir de los resultados completados en las iteraciones anteriores, añadiendo nuevos objetivos/requisitos o mejorando los que ya fueron completados. Un aspecto fundamental para guiar el desarrollo iterativo e incremental es la priorización de los objetivos / requesitos en función del valor que aportan al cliente.

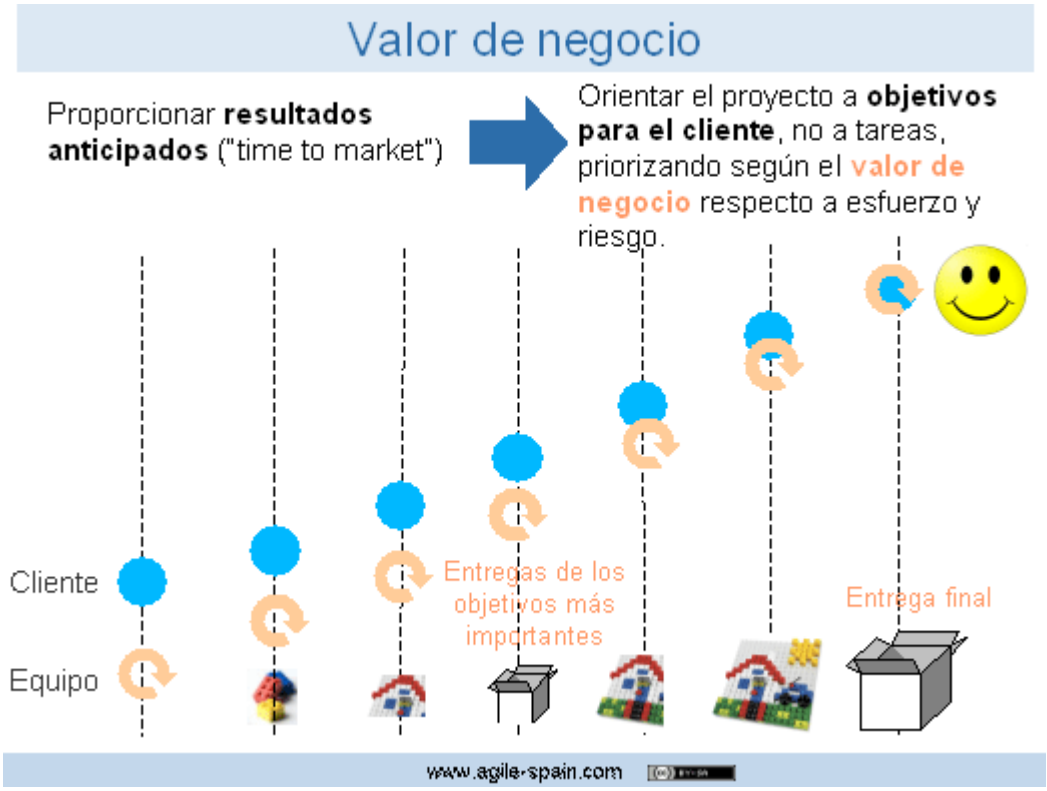


Figura 33 : Modelo iterativo y incremental

Beneficios :

- Se puede gestionar las expectivas del cliente (requisitos desarrollados, velocidad de desarrollo, calidad) de manera regular.

- El cliente puede comenzar el proyecto con requsitos de alto nivel, quizás no del todo completos, de manera que se vayan refinando en sucesivas iteraciones.

- El cliente puede obtener resultados importantes y usables ya desde las primeras iteraciones.

- El cliente como máximo puede perder los recursos dedicados a una iteración, no los de todo el proyecto.

- Dado que cada iteración debe dar como resultado requisitos terminados, se minimiza el número de errores que se producen en el desarrollo y se aumentar la calidad.

Restricciones :

- La disponibilidad del cliente debe ser alta durante todo el proyecto dado que participa de manera continua

- Cada iteración ha de aportar un valor al cliente, entregar unos resultados cerrados que sean susceptibles de ser utilizados por él.

- Es necesario disponer de técnicas y herramientas que permiten hacer cambios fácilmente en el producto

Recomendaciones :

- Utilizar iteraciones cortas de 2 a 4 semanas incrementa la productividad del proyecto, dado que el equipo trabaja de forma más eficiente cuando tiene objetivos a corto plazo.

- Utilizar iteraciones regulares, de manera que todas sean un timebox de la misma duración

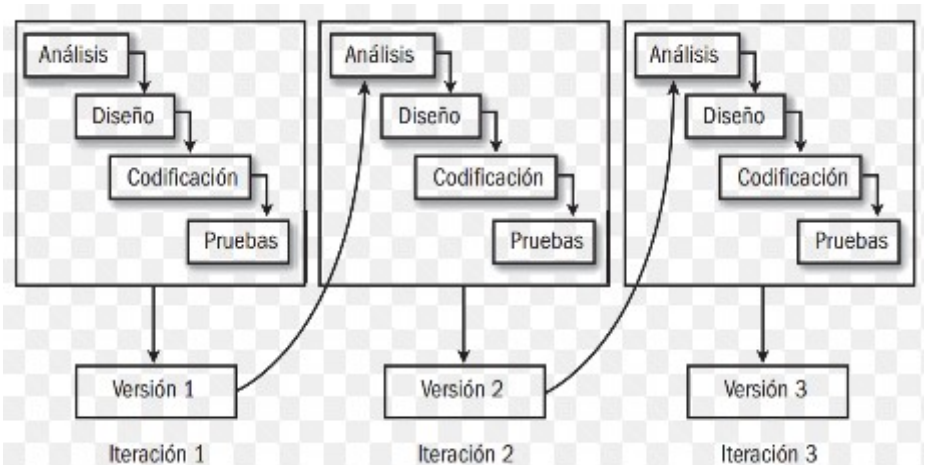


Figura 34 : Modelo iterativo e incremental

5.2. Planificación

El proyecto se ha llevado a cabo en cuatro grandes fases :

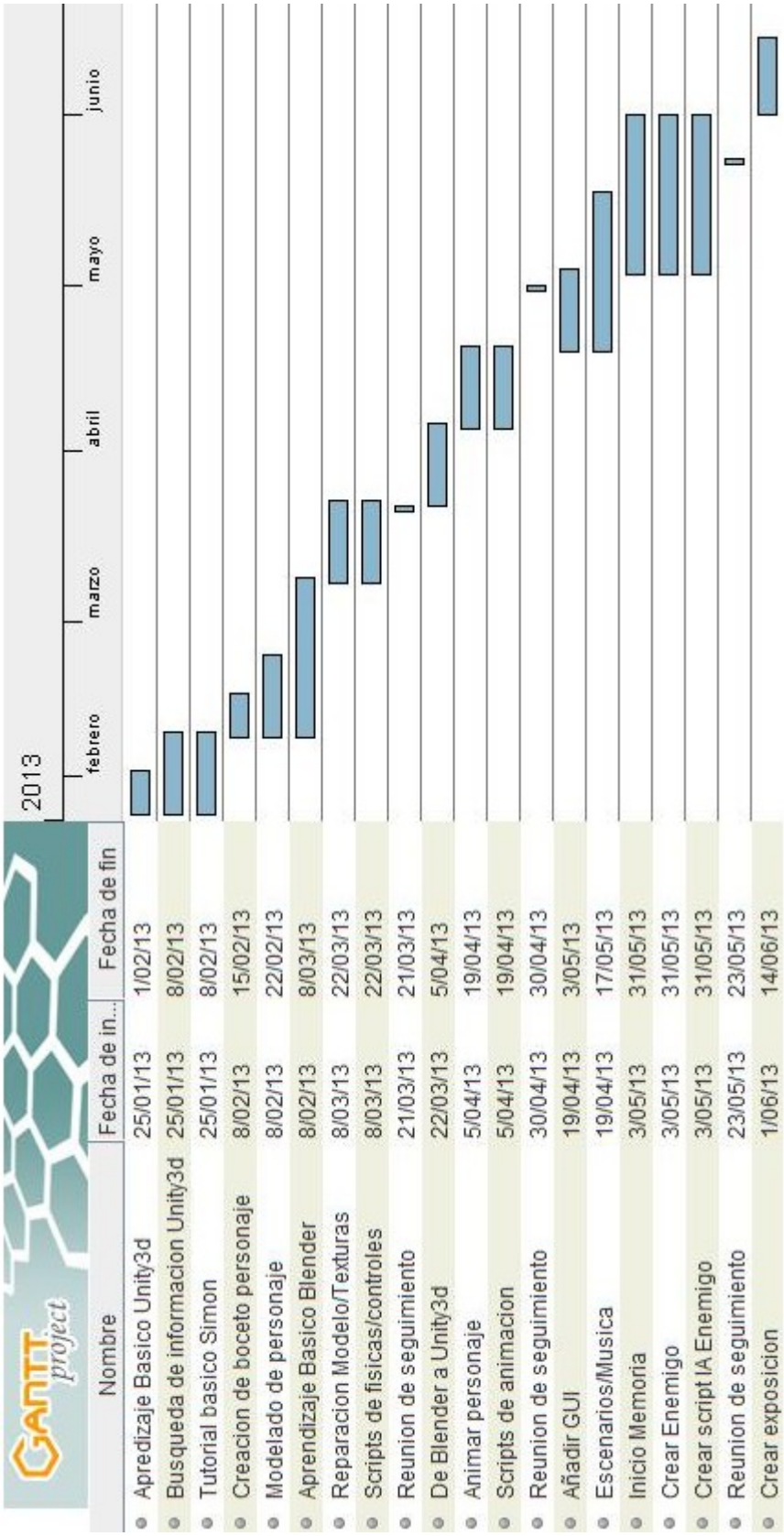
- **Fase inicial de estudio previo**: en esta fase se ha dedicado todo el tiempo posible a aprender sobre que se debería aprender y como se debería aprender para poder llegar al objetivo al final del plazo establecido. En esta fase se planifico el alcance a priori de lo que se quería hacer, teniendo en cuenta que pudiera haber limitaciones de tiempo o conocimientos. Y como seria el videojuego que se quería crear.

- **Fase de estudio teórico/practico de Blender**: en esta fase se dedico a aprender el funcionamiento del programa de diseño gráfico Blender, realizando ejemplos y manipulando el programa para conseguir objetivos. Una vez hecho esto, se propuso que se haría en blender de el proyecto, llegando a decidir que por temas de tiempo se haría unicamente el personaje principal. Para realizar este personaje, se realizaron bocetos en blanco y negro y a color, y se emplearon como plantilla en blender para realizar el modelo 3d.

- **Fase de estudio teórico/practico de Unity3d**: en esta fase se dedico a aprender el funcionamiento del programa de motor gráfico Unity, para ello se realizaron ejemplos previos y se empezó a crear un base plana con un objeto personaje, con el que poder empezar a testear scripting y asimilar el uso de los objetos y interfaz que nos propone unity.

- **Fase de finalización**: en esta fase se dedico a juntar toda la información recolectada y sumarla a un documento para crear la memoria, ademas de acabar los últimos retoques del videojuego para que pudiera dar a luz, para la fecha de la entrega.

5.3. Diagrama de Gantt



- En el diagrama aparecen las diferentes fases en las que se ha dividido el proyecto y las tareas correspondientes a cada fase junto con el tiempo estimado o real para llevar a cabo la tarea.

5.4. Explicación iteraciones realizadas

En esta parte se explicaran a grandes rasgos los puntos mas importantes de las entregas realizadas cada dos semanas, en ellas se puede ver el proceso seguido para la realización del juego.

- **Primera iteracion** : Se partia desde cero y tenia que empezar de una idea inicial, en este caso el objetivo era crear un mundo con una plataforma, que seria el suelo y un objeto que pudiera moverse por este suelo.

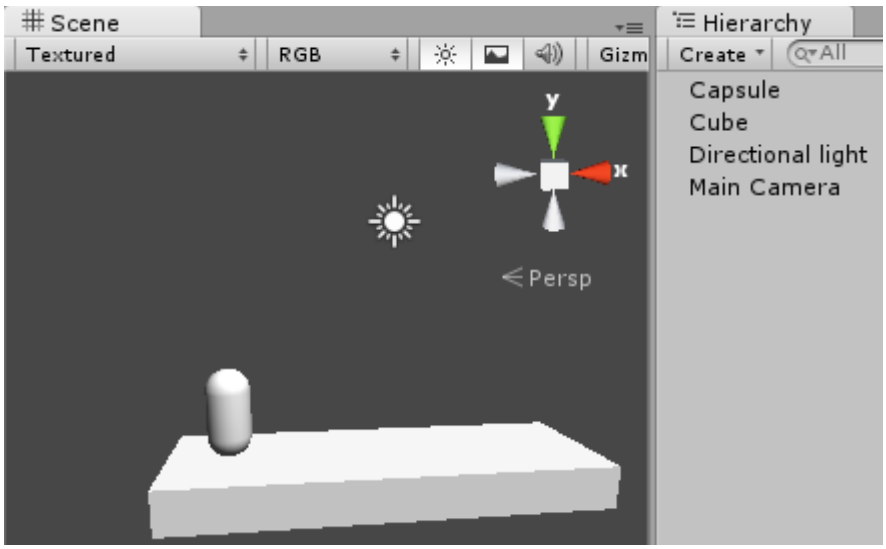


Figura 35 : Primera iteracion Unity3d

Aunque parece inicialmente un paso sencillo, pasa a ser el mas importante a la hora de realizar todo lo demas.

Como se ve en la imagen, lo mas importante de esta es que tiene que tener los objetos basicos y necesarios en cualquier escena, estos son objetos, camara y iluminación.

Como nota en este paso, es colocar todos los objetos en la partiendo de la posición x = 0 , y = 0 , z = 0 que nos servira de referencia, no hay que olvidar que se trabaja en 3D por lo tanto el orden en el espacio pasa a tener una importancia mayuscula.

– **Segunda iteracion** : En esta segunda iteracion, se empezo a trabajar en nuestro personaje principal, para ello se empezaria a usar el programa de diseño grafico Blender.

Para ello se crearon los bocetos anteriormente mostrados, y partiendo de ellos y de las tecnicas explicadas anteriormente en el tema 3 (Blender), se llego a realizar nuestro personaje en 3d, mediante creación/modificación de vertices.

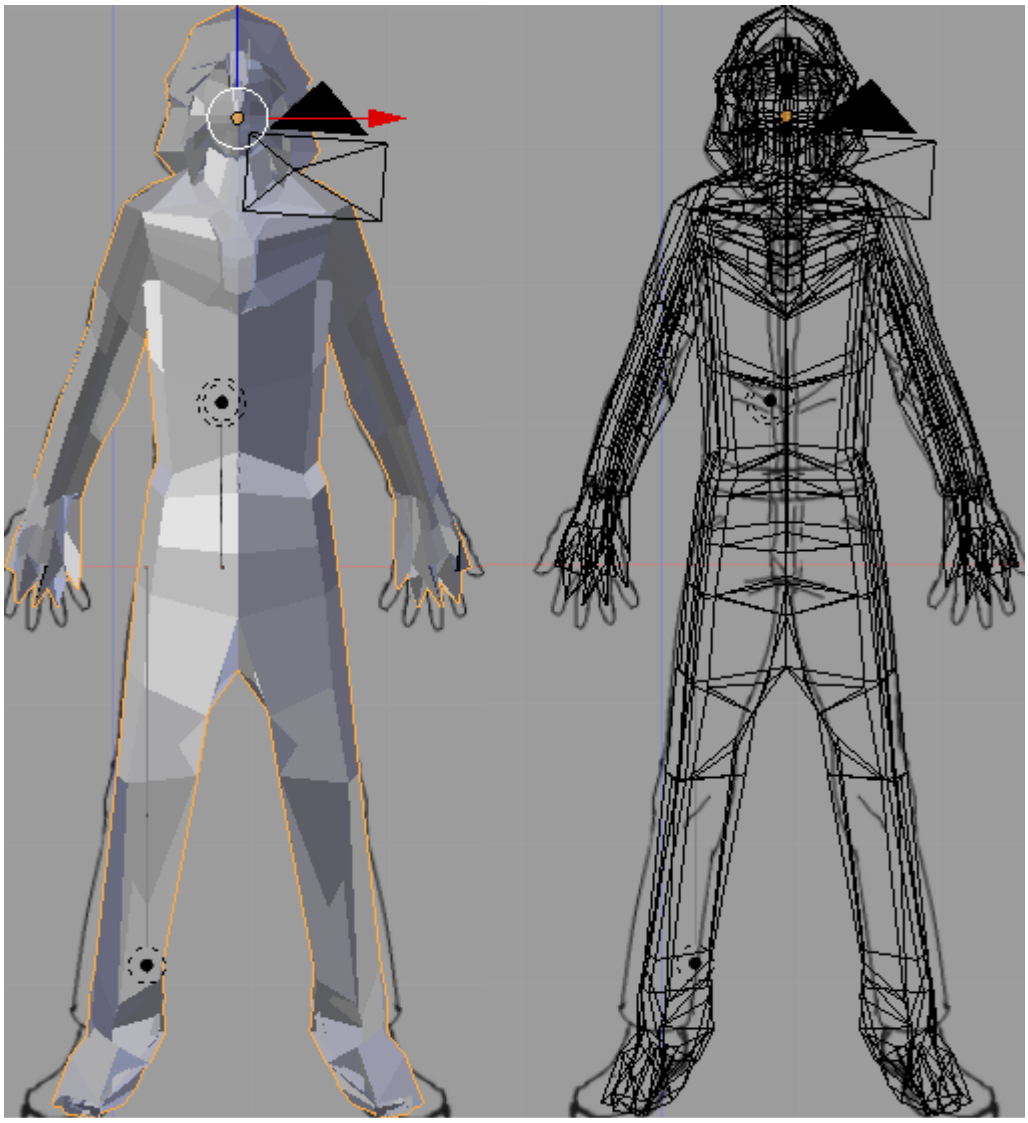


Figura 36 : Vista objeto y vista wireframe de nuestro personaje con en el fondo la imagen de referencia respetada

Como detalles importantes en este proceso, se respeto parcialmente el boceto creado previamente del personaje, y se realizo usando la tecnica de mirror, que nos permite a raiz de crear un semicirculo, tenerlo entero, en este caso sirvio para conseguir tener un personaje totalmente simetrico. Se descartaron otras modificaciones importantes como subdivision surface(mayor calidad de modelo), por la sencillez del juego.

Una vez realizado nuestro modelo, se pasa a realizar la parte de

texturizado. En esta parte tenia varias maneras de realizarlo, con Unwrap, Smart UV project, etc... en este caso elegi hacerlo usando una project from view, que nos permite cojer directamente las caras de nuestro modelo y colocarlos en la imagen que ya teniamos previamente creada en color. El resultado seria este:

Figura 37 : Modelo objeto y project view con la textura cargada

Como se puede observar solo se coje la mitad de nuestro boceto, no hay que olvidar como dicho anteriormente que se ha realizado el personaje usando la tecnica de mirror(espejo).

Como ultimo paso se inserto en nuestro escenario en el unity3d. De manera realmente sencilla, solo colocando nuestro modelo en nuestra carpeta de Assets el programa ya te lo importa automaticamente, gracias a su gran compatibilidad.

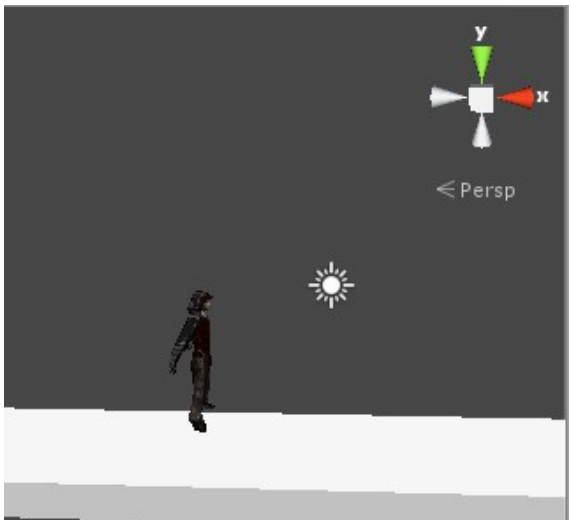


Figura 38 : Personaje blender importando en unity3d

– **Tercera iteracion** : En esta tercera iteracion se acabo de acoplar el personaje al juego, para ello se empezaron a crear los scripts necesarios para el funcionamiento correcto de este, sobre la plataforma creada.

Se añadio a nuestro personaje una box collider(caja de collisiones), que nos permite saber respetando los limites de nuestro personaje, donde va a collisionar, y rigidbody que nos permite controlar otro tipo de variables, como por ejemplo la gravedad, o hacia que diferentes direcciones puede desplazarse, en este caso el eje de la Z seria bloqueado por ser un juego con movimientos en 2d.

Se crearon unas fisicas (como actuaria nuestro personaje con respecto el escenario), relacionadas con los controles (las teclas que usariamos para moverlo). Para destacar de este paso se uso la tecnica de raycasting para saber si colisionamos o no. Esta tecnica consiste en tirar unos rayos invisibles desde nuestro objeto en ciertas direcciones, con esto podemos controlar dichas collisiones, y si se esta tocando o no otro objeto.

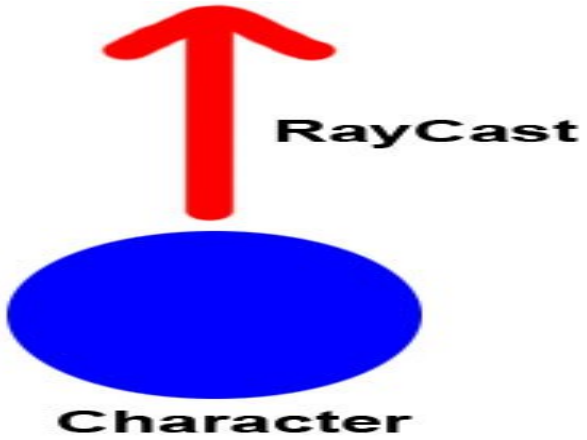


Figura 39 : Raycasting

Tambien se creo un script para controlar la camara que siguiera a nuestro personaje, para dar la sensacion de juego 2.5D .

Para acabar esta entrega se acabaron creando un plano invisible que delimitaria por abajo nuestro escenario, y que nos haria "morir" en el caso de tocarlo, y aparte unos checkpoint(puntos de control), que nos permitirian, al morir volver a nacer en el ultimo checkpoint tocado.

Figura 40 : Checkpoint

– **Cuarta iteracion** : En esta cuarta entrega se implementaron las animaciones de personaje, creandolas en blender y importandolas a unity3d para posteriomente ser llamadas desde los scripts segun se necesite.

Para empezar a animar nuestro personaje, primero tenemos que crear un esqueleto, en este caso se creo un esqueleto humano respetando todas las articulaciones de nuestro personaje.

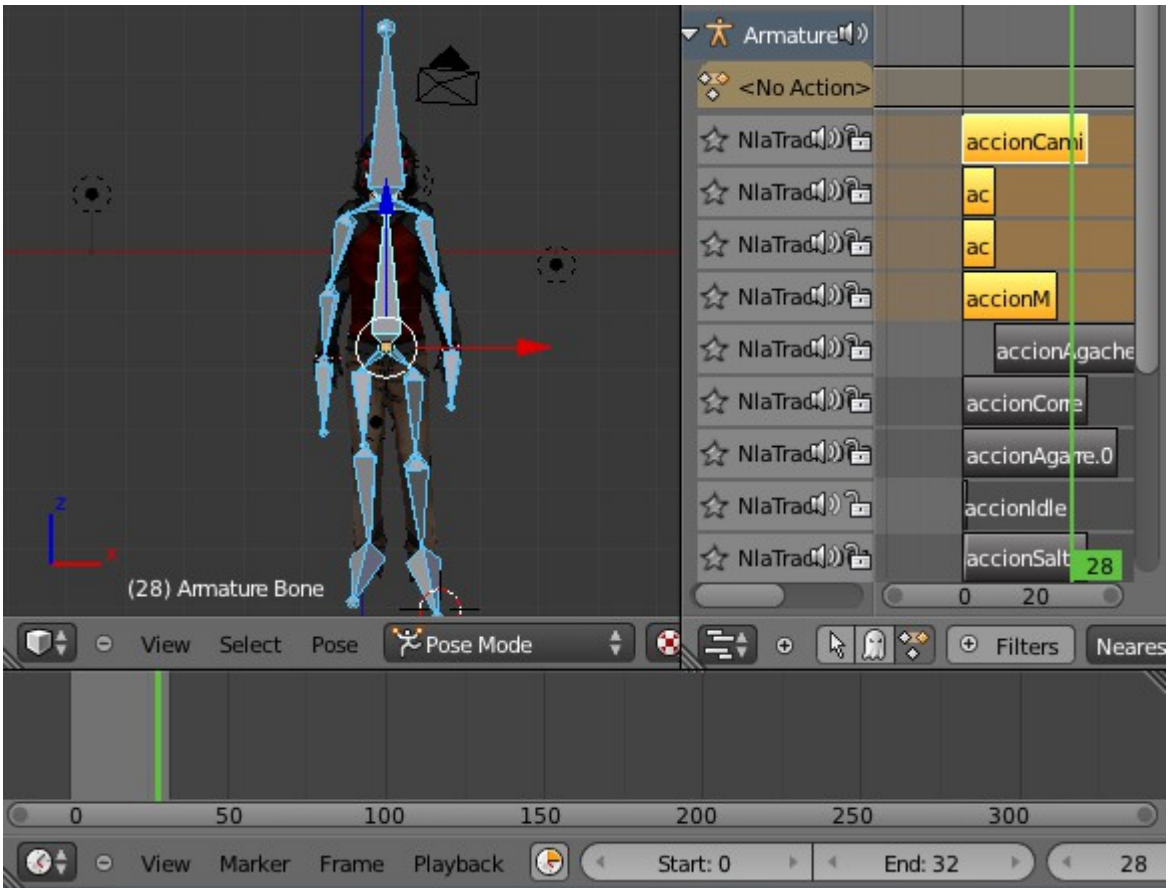


Figura 41 : Blender armadura y animaciones

Una vez creado, se exporto el archivo .fbx (autodesk) que unity3d carga automaticamente. Una vez con esto tenemos que cargar en nuestro modelo dentro de unity3d las animaciones.

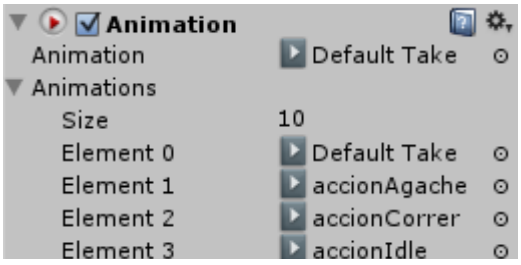


Figura 42 : Animaciones en unity3d

Una vez cargadas ya pueden ser llamadas desde los scripts, en este caso uno creado especialmente llamado animaciones.

– **Quinta iteracion** : En esta quinta iteracion se fue a implementar todo lo relacionado con el escenario, para ello se realizo una idea inicial de como seria la pantalla de demostración en la que se actuaria, y se empezo a diseñar usando plataformas varias, para crear tanto el suelo como las paredes.

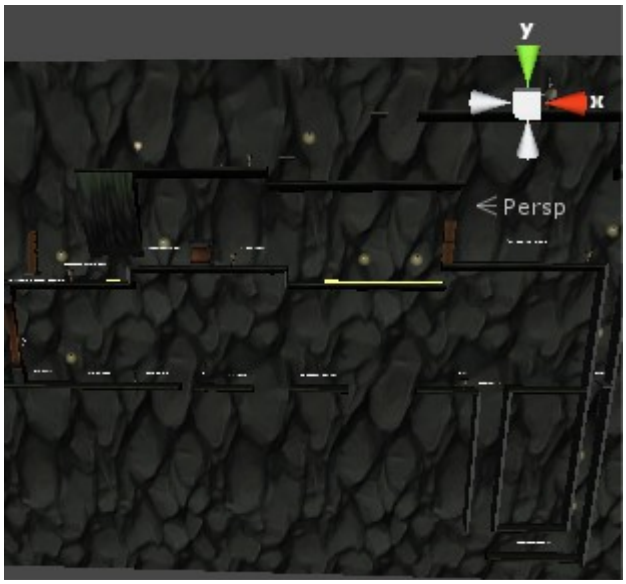


Figura 43 : Escenario de nuestra pantalla demo

– **Sexta iteracion** : En esta sexta iteracion se fue a implementar todo lo relacionado con los enemigos y su IA(inteligencia artificial), en este caso se decidio que fueran diferentes tipo esqueletos, y un tipo de escudo fantasmal.

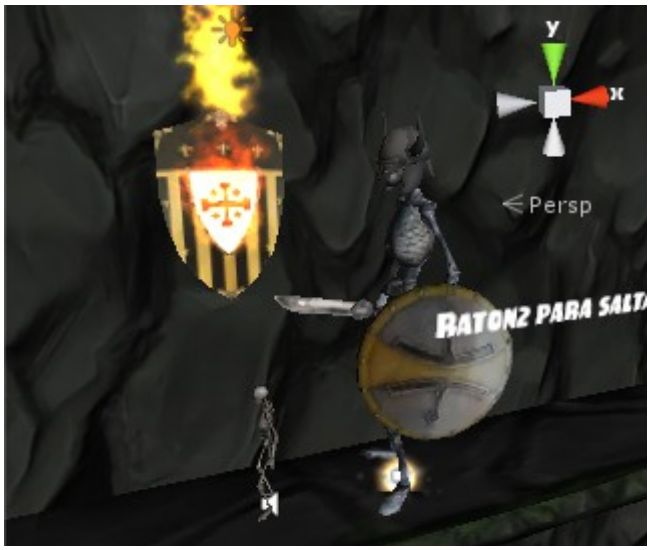


Figura 44 : Enemigos creados

De el script de IA, destacar el uso de Vector3.Distance, que nos permite calcular la distancia respecto un objeto ya señalado(en este caso el protagonista), de el objeto que queramos. Usando esto podemos hacer mover a nuestro personaje y darle la accion de golpear en el momento apropiado para dar sensacion realista.

– **Septima iteracion** : En esta septima iteracion se trato todo lo relacionado con el ambiente, estos son: musicas, efectos de sonidos, efectos de particulas, menus y GUI del juego.

Figura 45 : textura GUI ingame

Como detalle en unity3d se nos proporciona un metodo para tratar con todo tipos de gui, este metodo se trata del public void OnGUI() , en el podemos modificar todo lo que queramos relacionado con todo lo que necesitamos que se vea en pantalla independientemente de otros escenarios.

Figura 46 : Parte de uno de los menus principales

– **Octava iteracion** : En esta octava iteracion se trato todo lo relacionado con arreglar bugs, y añadir todos los assets posibles para completar el juego, otorgandole un inicio y un final.

Figura 47 : Añadidas espadas con efecto flamigero a nuestro personaje

Figura 48 : Escenas para cerrar nuestra demo

5.5. Valoración del trabajo realizado

El proyecto se a realizado en el periodo de 5 meses, desde el 25 de enero, hasta el 14 de junio, unos 140 días, con una media de 3-4 horas diarias de dedicación entre, estudio e implementación, lo que da un total de 420-560 horas de dedicación.

Las entregas finales estaban programadas a realizarse en un plazo de 2 semanas, siguiendo el sistema iterativo e incremental, pero debido a ciertos problemas algunos de las entregas, han tenido que ser vueltas a tratar o no se han podido entregar a tiempo. Los principales problemas han sido:

- Muchos problemas en la interacción para el paso de Blender a Unity3d de los modelos necesitados.

- Problemas con documentación a nivel código C#, que complicaban el desarrollo de los scripts.

- Desconocimiento de algunos conceptos básicos, que se tuvieron que ir aprendiendo progresivamente y no entraban en una planificación inicial.

- Experiencia nula en la creación de videojuegos y en relación con los motores relacionados a estos.

- Algunos pequeños recortes en la idea inicial del juego, para acabarlo de manera total para su entrega, pese a ello el resultado es mas que satisfactorio para lo que me había propuesto de inicio.

CAPITULO 6: CONCLUSIONES

Las conclusiones que se pueden sacar de este trabajo son mas que satisfactorias para mis intereses.

- **1** . Se ha conseguido realizar el tipo de juego que se quería, dando un acabado mas que suficiente teniendo en cuenta que lo ha realizado una persona. No para llegar a nivel comercial, pero si para demostrar que se han adquirido los conocimientos suficientes, para que con mas tiempo, o personas haber podido realizar un juego que podría servir a nivel comercial.

- **2** . Se ha conseguido un acercamiento a tecnologías de desarrollo como pueden ser Unity, o otras de diseño gráfico como podría ser Blender. Analizando y poniendo a prueba estos programas de última generación.

- **3** . Se ha conseguido diseñar en código c#, todo lo necesario, dejando un código muy ordenado y comprensible para cualquier persona que quiera verlo, pudiendo servirle como tutorial para realizar su propio juego 2.5D de plataformas.

- **4** . Actualmente con los nuevos tipos de motores de point and click(unity3d), es muy fácil programar un videojuego, y no se necesita un alto conocimiento de programación, haciéndolo mucho mas sencillo de lo que podría parecer anteriormente.

- **5** . Unity3d es un sistema multiplataforma que permite exportar los juegos en muchos formatos, desde windows hasta navegadores web o consolas.

- **6** . Una metodología de desarrollo de videojuegos no es más que un proceso de desarrollo de software orientado a la creación de juegos. Esto significa que se pueden utilizar todas las herramientas y prácticas de Ingeniería de Software para crear videojuegos consistentes y escalables. Y reutilizables para próximos videojuegos.

CAPITULO 7: BIBLIOGRAFÍA

- 1. Pagina oficial de unity3d : http://unity3d.com/
- 2. Pagina oficial de unity3d España : http://spanish.unity3d.com/
- 3. Tutoriales oficiales : http://unity3d.com/learn/tutorials/modules
- 4. Manual oficial : http://docs.unity3d.com/Documentation/Components/index.html
- 5. Documentación Scripting oficial : http://docs.unity3d.com/Documentation/ScriptReference/index.html
- 6. Pagina de vídeos con cantidad de tutoriales: http://www.youtube.com
- 7. Tutoriales unity en español : http://trinit.es/unity/tutoriales/
- 8. Pagina oficial de blender : http://www.blender.org/
- 9. Pagina con modelos blender : http://www.blender-models.com/
- 10. Tienda oficial unity3d :http://www.unity3d.com/asset-store/
- 11. Pagina para descargar sonidos : http://www.freesound.org/
- 12. Guía Blender : http://joaclintistgud.files.wordpress.com/2009/11/guia_blender_25.pdf

ANEXOS EN LA MEMORIA

Como anexo se encuentra el archivo con el videojuego, en la carpeta llamada "Mi juego", abriendo esta carpeta se pueden encontrar diferentes carpetas estas son:

- **Assets** : Assets creados para crear mi juego, en esta carpeta esta todo lo utilizado y creado para la realización del juego

- **Ejecutable** : En esta carpeta están los ejecutables creados para Windows, Mac y Webplayer.

- **Library** : La librería principal de unity3d donde están los compiladores básicos y librerías de unity3d.

- **Project Settings** : Configuraciones del proyecto actual.

- **Temp** : Archivos temporales.

- **Otras cosas** : Bocetos y otro tipo de archivos.

El detalle de la carpeta mas importante Assets :

- **Escenas** : Contiene las escenas creadas para el juego.
- **Fuentes** : Contiene los tipos diferentes de letras empleados.
- **GUITexture** : Contiene las texturas básicas del juego.
- **Música** : Contiene las músicas y sonidos del juego.
- **Objetos** : Contiene todos los modelos 3D utilizados.
- **Scripts** : Contiene todos los scripts creados.
- **Standard Assets** : Contiene los Assets creados por unity3d básicos para realizar cualquier aplicación con el programa.