Launchageddon: Gameplay

Memoria del Proyecto de Final de Carrera
de Ingeniería Informática

Realizado por Marcos Sueiro Eglicerio

y dirigido por Enric Martí Gòdia

Bellaterra 17 de Septiembre de 2012

El abajo firmante, Enric Martí Gòdia

Profesor de la Escuela de Ingeniería de la UAB,

CERTIFICA:

Que el trabajo al que corresponde esta memoria ha sido realizado bajo su dirección por **Marcos Sueiro Eglicerio**

Y para que conste firma la presente.

Firmado: Enric Martí Gòdia

Bellaterra, 17 de Septiembre de 2012

Agradecimientos

Quiero agradecer a mis padres Agustín y Pilar, mi novia Celia y mi hermano Eric su apoyo durante el duro desarrollo del proyecto.

A mis compañeros de equipo de proyecto: Enric Martinez Ibarra y Jordi Cartes Rosell, por toda su paciencia y ayuda ofrecida.

A mi director de proyecto Enric Martí Gòdia por la confianza depositada en el equipo y en el proyecto, así como por la libertad de decisiones que nos ha dejado tomar.

A Marta Preguezuelos por el diseño del logo de Launchageddon, texturas e iconos.

También agradecer a artistas como John Parr, Tim Feehan, Depeche Mode, 38 Special, ZZ Top y Yes por la música que me ha amenizado el desarrollo del proyecto.

Resumen

El siguiente documento corresponde a la memoria del proyecto de final de carrera de Ingeniería Informática, que contiene el diseño y la implementación del videojuego Launchageddon. Éste corresponde a un proyecto desarrollado entre tres personas.

En el presente encontrará la parte perteneciente a diseño e implementación del juego, física, cámaras, lógica de juego, interfaz de usuarios, gestión de archivos, localización del juego y efectos gráficos y de sonido.

El resultado es un juego dónde se pueden jugar varios niveles y experimentar con diferentes mecánicas. Éste ha sido validado por toda una serie de personas que lo han probado.

Resum

El següent document correspòn a la memòria del projecte de final de carrera d'Enginyería Informàtica, que conté el disseny i implementació del videojoc Launchageddon. Aquest correspon a un projecte desenvolupat entre tres persones.

En el present trobarà la part que correspon a disseny i implementació del joc, física, càmeres, lógica de joc, interfície d'usuaris, gestió d'arxius, localització del joc i efectes gràfics i de sò.

El resultat es un joc on es poden jugar varis nivells i experimentar amb diferents mecàniques. Aquest ha estat validat per tota una serie de persones que l'han provat.

Abstract

The following document deals with the report of the final Information Technologies degree which contains the design and implementation of Launchageddon game. This belongs to a project developed by three people.

In the present you will find the part that relates to the design and implementation of the game, physics, cameras, game logic, user interface, file management, the game's localization and the graphics and sound effects.

The result is a game where you can play different levels with different mechanics. This has been validated by a number of people who tried it.

Índice

1. Introducción

Un videojuego es un tipo de aplicación interactiva en la que sus objetivos pueden ser varios: narrar una historia, enseñar acerca de algún tema, ser un punto de reunión entre varios amigos que juegan a la vez, en definitiva, un tipo de aplicación destinada a hacer pasar un buen rato a quien juega y a divertirle.

Los videojuegos datan justo después de la segunda guerra mundial, allá por la década de 1940, empezaron a aparecer prototipos de juegos de tablero como por ejemplo el ajedrez realizados con placas electrónicas, leds y pulsadores. Pero hasta la década de 1960 con la figura de Ralph Baer [Ral] no se crearon los primeros videojuegos. Ralph patentó la que fue la primera consola de juegos, Brown Box, además de los primeros bocetos acerca de conceptos de diseño de videojuegos para un jugador y para dos.

A lo largo de los años se fue conformando lo que era un videojuego pero el gran boom llegó en la década de los 70-80, compañías como Atari [Ata], Nintendo [Nin] y Sega [Seg] fueron las que llevaron a cada uno de los hogares sus diseños en consolas y juegos. Dentro de los videojuegos existen diversos géneros según la temática de juego que ofrecen y según cómo se juegan. Podemos encontrar juegos de acción, de deportes, de puzles y lógica, de aventuras, cooperativos, competitivos, de lucha, etc.

Hoy en día la importancia del mundo de los videojuegos es innegable. En cuanto a volumen de negocio mueve más dinero que la música y el cine juntos. El mercado también se ha diseminado bastante en cuanto a jugadores y en cuanto a plataformas de juego. La edad de juego ya no se puede acotar, el rango de edades de personas jugadores es bastante amplio y podemos encontrar plataformas para ejecutar juegos en consolas, en pc, en la nube (navegadores web) en móviles, en recreativas, etc.

El desarrollo de un videojuego siempre pasa por una serie de fases: concepto/diseño de prototipos, producción/implementación, y test, cada una de ellas tiene tareas bien diferenciadas.

Desde bien pequeño me empecé a interesar por los videojuegos debido a familiares que tenían en casa una consola. Pero el gran momento llegó a los 8 años cuando me

regalaron mi primera consola, esta fue una Sega Megadrive, una consola de la denominada generación de los 16 bits según a las especificaciones del procesador que utilizaba.

Desde entonces no paré de jugar y disfrutar de este gran hobby de una forma bastante activa. Durante las siguientes generaciones de consolas y ordenadores seguí en ello desde plataformas como el PC, PlayStation hasta la actual Xbox 360.

Conforme fui haciéndome mayor empecé a estudiar ingeniería informática, cursé asignaturas de gráficos y poco a poco me fui interesando más por el tema, hasta que hice la asignatura de gráficos 2.

En esta asignatura realizamos un proyecto de un videojuego junto con unos cuantos compañeros más. Fruto de todo ese trabajo y del buen funcionamiento del grupo propusimos realizar como proyecto de final de carrera algo de este tipo. Todas estas experiencias son las que me han llevado a la realización de un videojuego y a conformar un equipo con Enric Martinez Ibarra y Jordi Cartes Rosell para desarrollar un juego al que hemos llamado Launchageddon. Creo que es bastante interesante el hecho de tener la oportunidad de investigar y estudiar sobre cómo se realiza un videojuego.

Seguidamente detallaremos los objetivos que se pretenden cumplir con el proyecto. Seguidamente el concepto de juego de Launchageddon que hemos propuesto. Este será el documento de referencia principal para el equipo que nos servirá de referencia a lo largo de todo el desarollo. De él extraeremos los módulos que formarán el proyecto y sus relaciones, cada uno de ellos conformará los objetivos individuales de cada miembro del equipo.

1.1. Objetivos del proyecto

El planteamiento inicial es hacer un proyecto de gran envergadura y con partes bien diferenciadas para que se pueda realizar en tres bloques bien distintos. Cada uno de ellos será realizado por una persona. Posteriormente el trabajo realizado se sincronizará y se tendrá que preparar para que funcione en conjunto. Los objetivos a cumplir por el proyecto son los siguientes:

1) Desarrollar un juego en 3D.

2) Diseño del concepto de juego para Launchaggedon en el que nos basaremos para la implementación.

3) Dividir las tareas equitativamente en tres módulos, que en su conjunto formen el videojuego propuesto en el concepto de juego.

4) Llevar a cabo la integración de los tres módulos y posteriormente verificar su correcto funcionamiento.

5) Solucionar los problemas que surjan durante la integración.

1.2. Launchageddon: concepto de juego

Este apartado explica a grandes rasgos en qué consiste el proyecto de crear un videojuego y cómo nace la idea.

También detalla sobre qué objetivos hemos trabajado para elaborar el concepto de juego y en que nos hemos inspirado. Se incluye el estudio del arte realizado para poner sobre la mesa los requerimientos necesarios en tecnología para el desarrollo y la metodología de trabajo definida para llevar a cabo todas y cada una de las tareas. Al final veremos un diagrama de módulos del proyecto en conjunto. Este nos valdrá para identificar cada una de las partes que componen el proyecto y sus interacciones, tanto internas entre ellos como externas con la tecnología, interfaces, usuario, etc.

Antes de empezar a trabajar en el proyecto en sí, se ha realizado una valoración acerca de las distintas opciones, teniendo en cuenta las posibilidades del equipo, factores que dependen básicamente del tiempo y recursos disponibles.

Una forma de definir el concepto de un juego es buscar los rasgos característicos que mejor lo describen. Durante la valoración se hizo un listado de tales palabras clave, las cuales definen los objetivos generales que nos hemos marcado para el juego. Estas palabras son: divertido, moderno, rápido, sencillo, competitivo, estratégico, portable y 3D.

1.2.1. ¿Porqué y con qué fin?

- **Divertido**: Como la propia palabra indica un videojuego no deja de ser una experiencia que ha de divertir, en concreto una experiencia interactiva, su razón de ser es un producto que divierta a la persona que está jugando.

- **Moderno:** Es muy difícil crear un concepto que sea completamente original hoy en día, no obstante siempre puede haber algún elemento fuera de lo común, que haga la aplicación más interesante para los usuarios. Este es uno de los motivos principales por los que se incluye Kinect en el juego, ya que es una tecnología relativamente moderna.

- **Portable:** El hecho de querer una aplicación que sea descrita con todas las características anteriores tiene muchos motivos y uno de ellos es que cuanto más sencilla y actual sea la aplicación más fácil será de llevarla a plataformas móviles, si a la vez nos hacemos valer de una tecnología que lo permita. Hoy en día esta característica está muy valorada, debido a que la gran mayoría de empresas realizan videojuegos multiplataforma con la finalidad de poder llegar a una cantidad mayor y diversa de público.

- **Rápido**: El juego debería ser rápido de jugar, una aplicación que se pueda usar en un ámbito casual que no requiera demasiada concentración. De este modo resulta más sencillo que el usuario decida usar la aplicación.

- **Sencillo**: Consideramos que un juego enfocado a plataformas móviles aparte de ser rápido tiene que ser sencillo. De este modo al usuario le costará menos comprender la dinámica de juego y la posibilidad de que un mayor número de personas lo prueben aumentará. Aparte de estos motivos el hecho de las limitaciones de recursos también influyen, puesto que no podemos optar a un proyecto excesivamente complejo, preferimos reducir este factor a un nivel factible para nosotros y hacer un producto de la máxima calidad posible.

- **Competitivo:** No sólo interesa en un juego que un usuario lo pruebe, sino que además vuelva a usarlo. Hacer un juego competitivo es una buena manera de mantener activos a los usuarios, aparte de que resulta más divertido y motivador para estos.

- **Estratégico**: Para nuestra propia motivación y la del propio usuario, uno de los objetivos principales es crear un "reto" con unos objetivos claros. Es uno de los principios de los juegos y una de nuestros propios desafíos. No debemos limitarnos a crear una aplicación gráfica visual. El objetivo es que la aplicación tenga una funcionalidad bien definida.

- **3D:** Una de las características que creemos más importantes fue el hecho de hacer un juego en tres dimensiones. Gracias a ello y a la integración con Kinect proporcionaríamos una experiencia de juego más inmersiva.

En las siguientes líneas se describe la mecánica del juego, incluye objetivos, jugabilidad y las fuentes de inspiración.

1.2.2. Mecánica de juego

Nuestra principal fuente de inspiración ha sido el juego llamado Angry Birds de Rovio Mobile [Rov], es un juego realizado en 2D que básicamente trata de lanzar unos pájaros para destruir una serie de estructuras y enemigos.

Launchageddon es un juego que consiste en derribar unos objetos situados dentro de unas determinadas estructuras mediante una variada gama de disparos en un entorno en tres dimensiones.

Las **estructuras** consisten en diferentes tipos de piezas o bloques amontonados entre sí formando una estructura global, como si se tratara de derribar castillos compuestos por las clásicas piezas de juguete hechas con madera (figura 1.1).

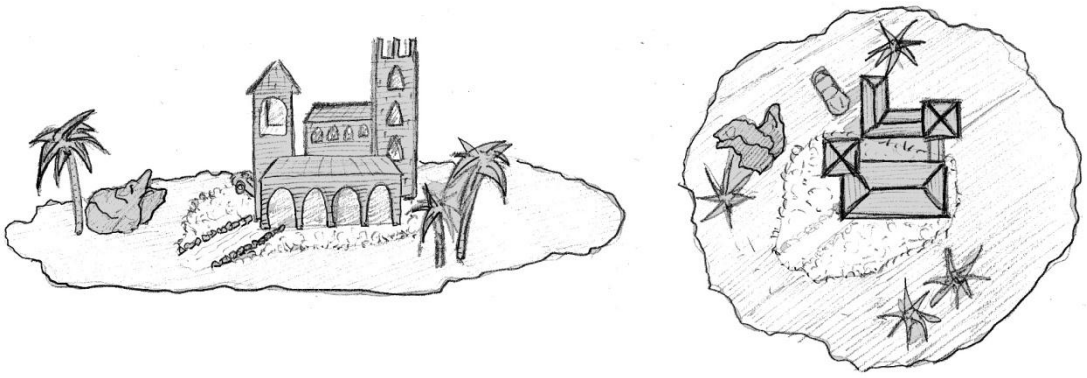

Figura 1.1: Concepto de estructuras del escenario, a la izquierda plano frontal, derecha plano alzado

Dentro de estas estructuras se encuentran una serie de objetos especiales que el jugador tiene golpear para conseguir el mayor número posible de puntos.

Los disparos se efectúan con un personaje que se lanza desde una posición determinada hacia la dirección que el jugador elija.

La jugabilidad se complementa con dos variables distintas: la primera de ellas es el tipo de ropa o traje que lleva el jugador, la cual modificará el tipo de lanzamiento que efectuará.

Se podrá escoger entre distintos tipos de traje, estos son los siguientes:

- **Traje normal:** es el lanzamiento más básico, un tiro parabólico.
- **Traje de paracaidista:** permite mientras se vuela seleccionar la trayectoria.
- **Traje de hombre bala:** permite lanzarse a una velocidad muy grande contra el escenario.
- **Traje de antigravedad:** permite ignorar todas las leyes de gravedad en vuelo y provoca un rebote al chocar.

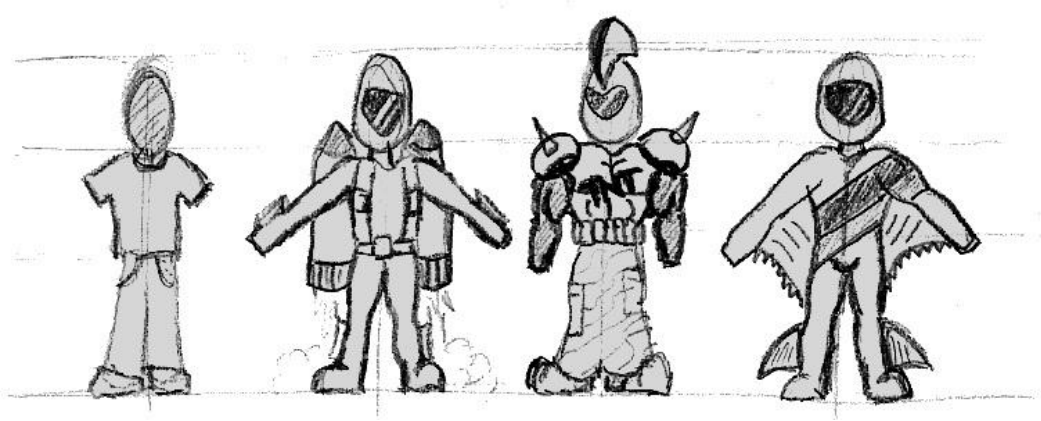

Figura 1.2: Concepto inicial de diseño de trajes

Por otro lado tenemos el poder de escoger entre una gama de cuatro cascos distintos para el personaje, estos otorgan distintos poderes una vez nuestro personaje impacta contra el escenario, estos son:

- **Casco normal:** es un casco que no otorga ninguna característica especial.
- **Casco de hacha:** permite que al chocar contra el escenario se rompan los objetos en trozos.
- **Casco explosivo:** al impactar el casco coloca una bomba en el escenario.
- **Casco científico:** permite ver a través de las estructuras para buscar objetos de importancia.

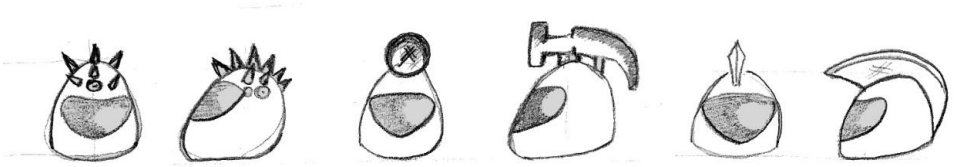

Figura 1.3: Concepto inicial de diseño de cascos

Una de las cualidades de la jugabilidad y que además otorga un punto de estrategia es que los tipos de lanzamiento pueden combinarse con los tipos de cascos, de modo que el jugador puede probar distintas combinaciones para conseguir sus objetivos.

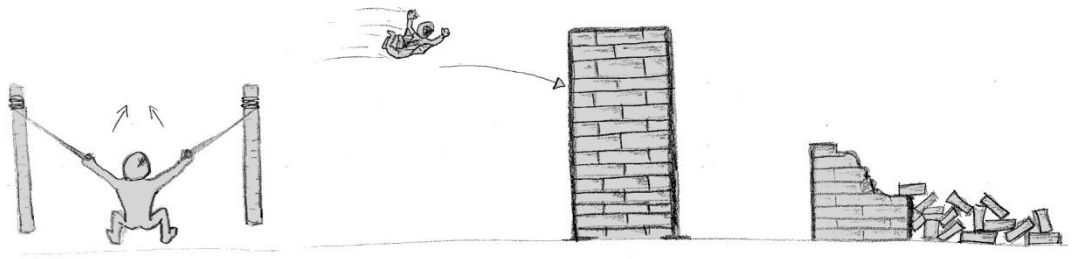

Figura 1.4: Concepto de fases de juego

Dicha estrategia también consiste en que el jugador piense bien qué disparos va a utilizar, puesto que estos estarán limitados de una forma distinta para cada fase. Cada fase también tendrá un número de puntos a batir para superarlo, que se tendrán que conseguir con un número de rondas distintas para cada fase. A lo largo del juego pasaremos por una serie de fases y niveles, ellas se pasarán de forma satisfactoria obteniendo puntos de destrucción, que se conseguirán provocando destrucción en el escenario y destruyendo toda una serie de objetos específicos que darán más puntos. Toda esta serie de objetivos serán definidos por la persona responsable del diseño de juego del equipo.

1.2.3. Elección entorno de desarrollo

Previamente al desarrollo del proyecto hemos realizado un estudio con la idea de llegar a un consenso sobre qué entorno y herramientas utilizaríamos, teniendo en cuenta las características definidas en el concepto de juego.

El estudio se ha basado en la búsqueda de motores gráficos [Anexo I] que pudieran servirnos para llevar acabo el proyecto, también se ha valorado la implementación de uno propio.

Para realizar el estudio hemos valorado tres posibilidades, dos son motores gráficos existentes (Unity3D y Unreal Engine), mientras que la última de ellas es la de realizar un motor propio.

Entre las cosas buenas que aporta el uso de un motor gráfico externo es que podemos encontrar un notable ahorro en tiempo en la codificación del proyecto, ya que dispondremos de una serie de herramientas ya realizadas. Otro factor importante será la integración con las demás partes del proyecto, dependiendo de la heterogeneidad de tecnologías que se tengan que implementar.

El hecho de utilizar un motor gráfico externo nos va a facilitar mucho la tarea de codificación del proyecto y tiempo de desarrollo, además de una facilidad mayor a la hora de integrar todas las partes. La tarea entonces será de saber qué cosas se pueden hacer, aprender a hacerlas y qué cosas no se pueden hacer debido a las limitaciones del motor escogido.

Nos hemos centrado en el estudio de dos motores gráficos uno es Unity y otro es Unreal Engine. El motivo de ello es debido a que son muy utilizados hoy en día y bastante bien documentados en libros y por la red.

a) Unity 3.5

Figura 1.5: Logotipo de Unity3D

Unity [Uni] (figura 1.5) es una herramienta especializada en el desarrollo de videojuegos. Entre sus características principales destaca la potencia de simulación de física, iluminación a tiempo real, importación de archivos de otros programas como Blender, 3D Studio Max. Pero sin duda uno de los puntos fuertes de Unity reside en la capacidad de llevar fácilmente la aplicación desarrollada a distintas plataformas, entre las cuales podemos encontrar: Navegador web, iOS, Android, Pc, Mac, Wii, Playstation 3 y Xbox360.

La versión actual de Unity es compatible con tres lenguajes de scripting: JavaScript, C# y Boo. Además posee un editor gráfico que permite crear escenarios de juego, colocar objetos de una forma cómoda para facilitar el proceso de desarrollo.

El factor multiplataforma de Unity es un claro punto muy positivo para la elección del motor gráfico, ya que permite realizar un desarrollo pensando en una plataforma y en el futuro, invirtiendo un poquito de tiempo a adaptar la misma aplicación a otra plataforma.

Un aspecto a tener en cuenta también reside en la enorme comunidad de desarrolladores independientes que realizan aplicaciones en Unity. Eso provoca que por la red se puedan encontrar referencias, ayudas, ejemplos y consejos para iniciarse en elmundo de Unity.

Unity tiene dos tipos de licencia: gratuita y profesional. La profesional cuesta unos 1500 dólares, y depende de las plataformas a las que quieres llevar la aplicación. En principio con la gratuita nos basta para la realización del proyecto ya que soporta compilación y generación de ejecutables para PC, MAC, navegador web y android.

b) Unreal Engine 3

Figura 1.6: Logotipo de Unreal Engine

Unreal Engine [Unr] (figura 1.6) es también un motor gráfico diseñado especialmente para el desarrollo de videojuegos. Es un motor desarrollado por la empresa Epic Games, muy utilizado en todo el mundo. Epic, aparte del desarrollo de tecnología propietaria, desarrolla videojuegos. Dos de las franquicias más conocidas que han diseñado son “Gears of War” y “Unreal Tournament”.

Igual que en Unity, Unreal es capaz de portar la aplicación que realizamos a diversas plataformas en un click, entre otras podemos encontrar PlayStation 3, iOs, Mac, PC y Xbox360.

Unreal Engine utiliza un lenguaje propio de desarrollo llamado UnrealScript, bastante parecido a un lenguaje de alto nivel orientado a objeto. También posee de un editor integrado bastante completo, aunque tiene escasas herramientas de importación de modelos, escenarios, etc, de otros programas.

Las licencias en Unreal son bastante más caras que en Unity. El precio de publicación es de 2500 dólares, y los royalties posteriores por ventas ascienden al 25%.

c) Uso de un motor gráfico propio

La otra alternativa es la creación de un motor gráfico propio. Creemos que esta opción requiere una carga de trabajo excesiva para cumplir los objetivos del proyecto debido a la heterogeneidad de las tareas que realizaremos.

Un aspecto positivo de crear una tecnología es que puedes tener el control total sobre ella, sin depender de terceros y saber a todo momento que estas implementando y en que estas focalizando tus esfuerzos.

Es una opción que nos obligaría a invertir demasiado tiempo de desarrollo y difícilmente nos podríamos centrar en el desarrollo del juego hasta que el motor fuera funcional.

Después de ver todos los puntos positivos y negativos del uso de un motor gráfico externo o propio, nos decantamos por el uso de uno externo, ahorrando bastante tiempo en desarrollo de tecnología cosa que nos permitirá centrarnos en el desarrollo del juego en sí, con el coste de tener que invertir bastante tiempo en el aprendizaje y la asimilación del motor gráfico externo que utilicemos.

Después de unas cuantas tardes de estudio y pruebas, y gracias a los manuales que hemos utilizado para conocer más acerca de Unity y Unreal Engine, hemos llegado a toda una serie de consideraciones para seleccionar la tecnología que utilizaremos.

Para ello hemos elaborado una tabla donde marcamos con un **+** los puntos donde que motor es mejor al otro, los puntos que hemos valorado son los siguientes:

- **Potencia gráfica:** Una vez vistas las demostraciones y proyectos que circulan por internet nos podemos hacernos una idea de cual es la potencia gráfica capaz de soportar cada uno. Eso significa el número de modelos que es capaz de gestionar en pantalla de forma fluida, su nivel de detalle, el número de partículas y la calidad de la iluminación que genera.

- **Editor:** Una vez instalados ambos motores gráficos y realizando una serie de tutoriales hemos podido valorar el nivel de edición de juego que soporta cada uno de los motores, temas como edición de niveles, edición de animaciones de personajes, edición de videos, sonido, etc.

- **Integración con programas externos:** Se ha valorado la capacidad del motor gráfico en permitir integración con librerías externas e importación de archivos generados en otro programa.

- **Documentación:** Realizando búsqueda por internet hemos valorado el soporte que tiene cada uno de los motores gráficos, la comunidad que genera contenido y las webs dedicadas a ofrecer ayuda y tutoriales.

- **Lenguaje de scripting:** Consultando en la información de cada motor gráfico hemos encontrado el lenguaje mediante el cual se programa en ellos.

- **Precio licencia profesional y precio licencia de publicación:** En la información oficial de cada uno de los motores gráficos se puede encontrar acerca del precio de usar el motor, y del precio de publicar las aplicaciones que realices.

La opción que más se adecua a nuestros requerimientos es claramente Unity. Los factores de mayor peso han sido la documentación existente. Para Unity hemos encontrado mucha más información, ejemplos y comunidad que desarrolla; otro motivo es que los lenguajes de scripting que utiliza Unity son más estándar que UnrealScript (el utilizado por Unreal Engine). Por tanto será un valor añadido que podremos aprovechar en un futuro, y

para finalizar el tema de las licencias: si algún día pensamos llevar nuestro producto a la venta tiene un precio mucho más accesible que Unreal Engine.

Características:	**Unity**	**Unreal Engine**
Potencia Gráfica		**+**
Editor		**+**
Integración con programas externos	**+**	
Documentación	**+**	
Lenguaje Scripting	**+**	
Precio licencia Pro	**1500 $**	**2500 $**
Precio licencia de publicación	**Gratis**	**Pro + 99$ + 25% ganancias**

Tabla 1.1: Comparativa entre Unity y Unreal Engine

Una vez escogida la tecnología que usaremos, para empezar a trabajar en el proyecto se ha empezado a estudiar toda una serie de manuales [Tut] y la API de Unity [Ure], que nos detallan todas las opciones de las que dispondremos.

Para la realización del proyecto se tomaron algunas decisiones más acerca de la tecnología que usaríamos, entre ellas podemos destacar: el proyecto será programado en un lenguaje de alto nivel orientado a objeto, C#. Su estructura básica estará formada por clases que interactuarán entre ellas utilizando patrones y mensajes. Para la programación se utilizará el entorno *MonoDevelop* con el compilador *Mono*, es un editor que está perfectamente integrado con Unity, este nos permitirá compilar, depurar y ejecutar rápidamente sin que tarde demasiado en sincronizar los cambios con Unity.

1.2.4. Diagrama de módulos

Una vez puesto en común el documento de concepto de juego se ha procedido a elaborar el diagrama de módulos de Launchageddon (figura 1.7). La división del trabajo se ha realizado de forma que quedan tres módulos que creemos bien diferenciados - con una carga de trabajo bien balanceada - donde cada uno será desarrollado por un integrante del equipo y que se relacionarán entre ellos.

Como podemos ver en la figura 1.7 los módulos son:

a) Gameplay.
b) Editor.
c) Tecnología.

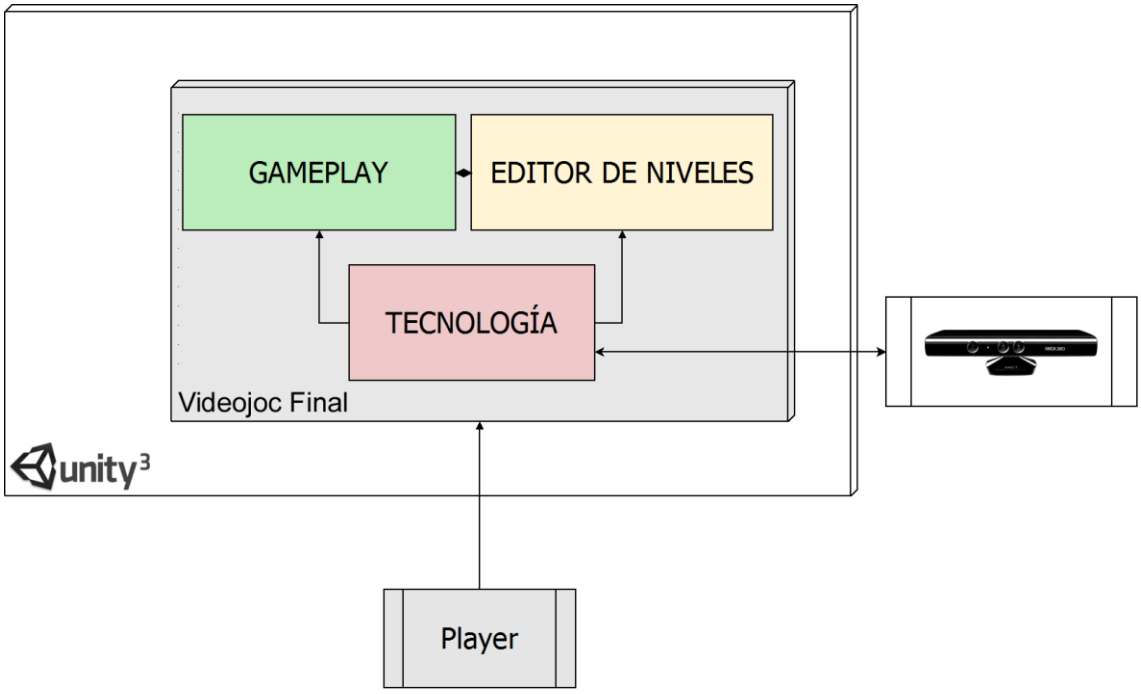

Figura 1.7: Diagrama de módulos del proyecto

a) Gameplay es el módulo encargado de gestionar toda la lógica de juego, las físicas, la interfaz de usuario en juego, la interfaz de menú principal y los niveles. Además será el encargado de detallar e implementar el diseño de juego, sistemas necesarios para gestionar

las puntuaciones del juego, acceso a archivos de configuración, de información sobre niveles y de localización del juego a otros idiomas.

b) Editor de niveles se encarga de proporcionar las herramientas necesarias al jugador para diseñar sus propios escenarios personalizados y exportarlos al módulo de Gameplay para poder disponer de estos en el juego.

c) Tecnología establece comunicación con los otros dos módulos y les proporciona las herramientas necesarias para poder trabajar con Kinect como interficie de usuario. Este modulo requiere una comunicación externa con el dispositivo Kinect para analizar los datos físicos del jugador necesarios para los controles.

Para llevar a cabo la integración entre módulos tendremos que definir cómo se comunican entre ellos y por qué.

Comunicaciones entre módulos

Seguidamente encontraremos un resumen de las integraciones más importantes que se llevarán a cabo:

- **GAMEPLAY:** El módulo encargado del juego tendrá que ser adaptado para recibir órdenes desde el modulo de **TECNOLOGÍA**, que enviará los datos en un formato estipulado por ambas partes, de manera que la parte de **GAMEPLAY** actúe en consecuencia y realice las acciones pertinentes dentro del juego. También se comunicará con **EDITOR** para detallar la forma en la que los niveles tendrán que ser cargados en el juego para que funcionen. Ambos llegaran a un acuerdo para la estructura de este tipo de datos y su funcionamiento.

- **EDITOR:** Este módulo establece comunicación con los otros dos. Por un lado recibe soporte del módulo **TECNOLOGIA** para poder utilizar el control por Kinect. De otra forma se comunica con el módulo de **GAMEPLAY** para compartir un conjunto de datos para poder acceder a todos los escenarios personalizados. En el caso de **EDITOR** para modificarlos y en el caso de **GAMEPLAY** para que se hagan servir en el juego.

- **TECNOLOGIA**: Para que los otros dos módulos puedan utilizar los controles que se ofrecen, se han llevado a cabo reuniones para realizar el análisis de requerimientos, donde se ha estipulado el formato de los datos a enviar.

1.2.5. Integración del trabajo realizado

Cada uno de los módulos es un trabajo individual. No obstante consideramos interesante poner factores en común de modo que los trabajos interaccionen entre si como si formaran parte de un proyecto global.

Lo consideramos un valor añadido, puesto que se suma la dificultad de coordinar 3 trabajos individuales de modo que cada uno no se aleje de su línea pero a la vez comprenda las necesidades de los otros dos.

Para poder integrar el trabajo realizado con las demás partes hemos definido una metodología de trabajo. Ella se ha cumplido mediante reuniones de carácter semanal o bisemanal.

En la figura 1.8 mostramos las tareas que se han realizado durante el tiempo que han durado las reuniones y el tiempo de dedicación a cada una de ellas por reunión. Las tareas que se han realizado en cada una de las reuniones han sido las siguientes:

- **Brainstorm:** durante esta fase se ha realizado una lluvia de ideas pensando en cosas a añadir al proyecto, funcionalidades, aspectos de diseño, etc.

- **Mostrar trabajo individual:** cada uno de los integrantes del grupo ha mostrado la faena que ha realizado de forma individual entre reunión y reunión.

- **Feedback/Estado del proyecto:** una vez vistas las cosas implementadas se ha procedido a realizar una valoración del estado del proyecto en conjunto y de cada una de las funcionalidades que se han desarrollado individualmente.

- **Sincronización:** una vez validadas las funcionalidades individuales de cada uno de los proyectos se ha procedido a integrar los cambios entre ellos, se ha adaptado la faena realizada para que funcionara con los demás módulos.

Normalmente el grupo se ha reunido una tarde por semana.

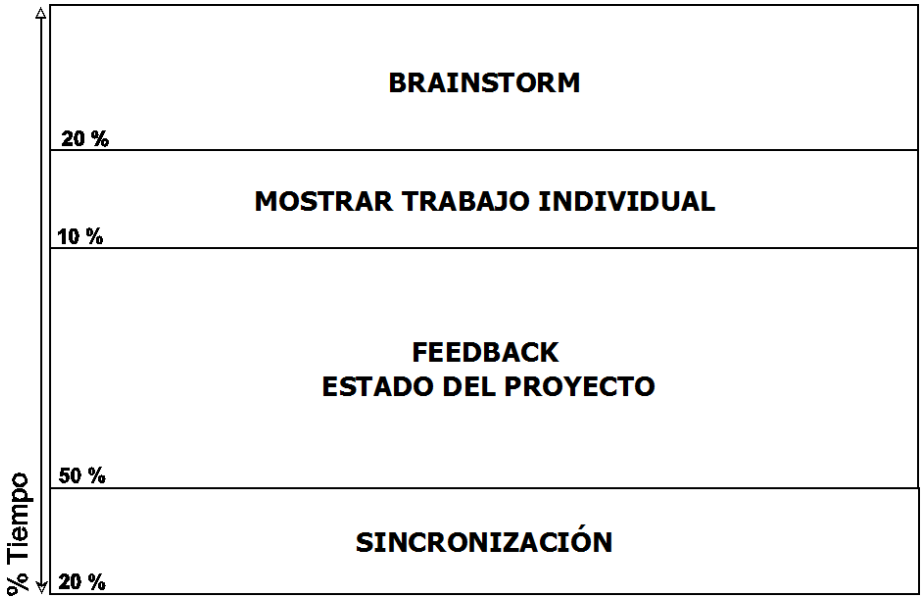

Figura 1.8: Porcentaje de tiempos orientativo para las reuniones de grupo

Como se puede observar, las reuniones nos ha servido tanto para comunicarnos entre los miembros de grupo como para sincronizar trabajo, tomar decisiones acerca del desarrollo, valorar la modificación e inclusión de características nuevas a la aplicación, etc. Fruto de estas reuniones se han ido realizando actas de equipo.

1.3. Objetivos individuales

A continuación encontraremos los objetivos individuales propuestos y desarrollados en la memoria, ellos hacen referencia un módulo de los vistos en el diagrama de modulos (1.7), en este caso **Gameplay**, ellos son los siguientes:

1) Investigación de técnicas y herramientas modernas que se utilizan para el desarrollo de videojuegos, así como su aprendizaje e implementación.

2) Diseño e implementación de todas las partes necesarias para el módulo **gameplay** de Launchaggedon.

3) Estudio de la física que necesitaremos y su implementación.

4) Aprender a utilizar la tecnología de detección de colisiones que emplea Unity, y realizar el estudio e implementación de resolución de colisiones.

5) Estudio e implementación de cámaras que se utilizaran para el juego, en concreto en tercera persona.

6) Diseñar el concepto de juego final y la mecánica de éste, sistemas de puntuaciones y rankings.

7) Diseñar las diferentes interfaces de usuario del juego.

8) Implementar la gestión y el acceso a datos y su codificación.

9) Diseñar un sistema para permitir la traducción fácil del juego a otros idiomas.

La memoria está dividida en capítulos y anexos, en cada uno de ellos encontraremos:

- En el capítulo 2 **Desarrrollo**: detallaremos el diseño de juego, así como el método seguido para implementar los módulos que cumplen cada uno de los objetivos propuestos.

- En el capítulo 3 **Validación:** servirá para validar por separado cada uno de los módulos detallados en **Desarrollo**.

- En el capítulo 4 **Conclusiones y mejoras**: se realizará una valoración de los objetivos que se han alcanzado con el proyecto como los que no.

- En el anexo I información acerca de: motores gráficos.

- En los anexos II, III, IV, V explicaciones sobre temas referentes a física: cinemática, dinámica, resolución de choques y diagramas de clases para programación.

- En el anexo VI el diagrama de clases para programación del sistema de gestión de archivos.

- En el anexo VII el manual de usuario de la aplicación.

- En el anexo VIII la encuesta realizada a los usuarios que han probado el juego.

2. Desarrollo

La parte de desarrollo se ha dividido en tres partes: en primer lugar se encuentran los requerimientos funcionales del programa, la planificación, viabilidad y la definición del concepto de juego y características. Luego la metodología de trabajo a seguir para su realización y para finalizar el diagrama de módulos de Launchageddon así como la explicación de cada uno de ellos, esta parte también incluirá diseño de sistemas para el juego. Para finalizar se incluye una pequeña explicación acerca de los efectos gráficos y de sonido incluidos en Launchageddon.

2.1. Diseño de juego

El siguiente apartado corresponde a las cosas que se han tenido en cuenta para la realización del juego: restricciones para aplicaciones en tiempo real – en requerimientos funcionales -, planificación del proyecto, viabilidad, objetivos de juego y sus principales características.

2.1.1. Requerimientos funcionales

El videojuego que vamos a realizar es un tipo de aplicación que tiene una restricción que afecta a la forma en la que se ha de estructurar y pensar su desarrollo.

Esta restricción es que es una aplicación que funciona a tiempo real. Dependiendo del tipo de videojuego esta restricción no se tendrá, ya que su tiempo de ejecución será basado en turnos o un sistema similar. En nuestro caso y para toda aplicación de esta naturaleza deberá ser capaz de ser ejecutada al máximo nivel de fluidez, para que todos y cada uno de sus elementos vayan sincronizados y sean capaces de hacernos creer que lo que estamos viendo es real.

Para ello se ha utilizado una estructura de diseño conocida y que funciona bastante bien para aplicaciones gráficas de este tipo. Es la mostrada en la figura 2.1.

Podemos observar como la estructura se compone de dos partes bien diferenciadas, una se ejecuta una sola vez como son inicialización, carga y finalización, mientras que

procesado de eventos, update y render se realizan en un bucle que dura hasta que finaliza la aplicación.

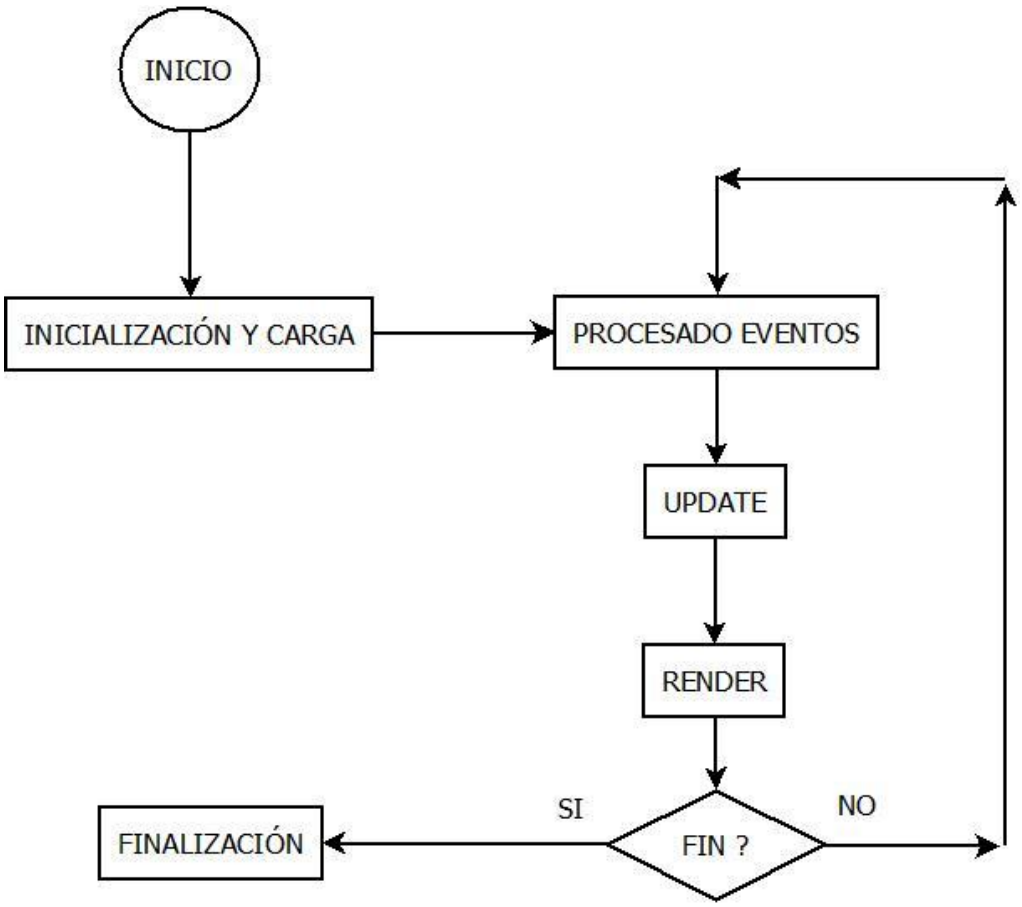

Figura 2.1: Estructura de una aplicación en tiempo real

No es muy difícil deducir, a raíz de la estructura que hemos podido observar, que para que la aplicación funcione a tiempo real hay que intentar minimizar la carga de trabajo que se realiza en las fases que se ejecutan más. Eso es, en las que están contenidas en el bucle principal mencionado (procesado de eventos, update y render).

Seguidamente hemos realizado un listado de las tareas que se deberían hacer en cada una de las fases que componen la estructura, y las cosas a evitar con el fin de intentar minimizar el tiempo que tarda en ejecutarse el bucle principal de la aplicación. Son las siguientes:

- **Inicialización y carga:** Esta parte es la única de todas que se considera fuera del bucle principal de la aplicación. Es la encargada de realizar las tareas que se consideran más costosas, entre ellas podemos encontrar carga en memoria de estructuras de datos, objetos, texturas, sonido y geometría de objetos, configuración de propiedades de la aplicación, lectura de archivos de disco, etc.

  En esta fase también se realiza la inicialización de las estructuras de datos necesarias, todo con la idea de intentar evitar que en el bucle principal se tengan que crear, inicializar y destruir objetos, operaciones que suelen tener un coste elevado en cuanto a computación.

- **Procesado de eventos:** En la parte de procesado de eventos se gestionan todos los métodos de entrada para la aplicación, eso es teclado, ratón y/u otras interfaces cmo Kinect, etc. También se ejecutan las rutinas asociadas a estos eventos.

  Una buena idea para esta parte suele ser evitar que se ejecuten las rutinas asociadas a los eventos más de una vez. Para ello se ponen limitaciones por contador de tiempo a la hora de detectar los eventos, ya que por ejemplo, si pulsamos un botón de teclado, a la aplicación le pueden llegar varias veces la pulsación de esa tecla y por tanto la ejecución de la rutina. Para evitar este problema se suelen poner restricciones que invalidan la lectura de eventos, ya realizados durante determinados intervalos de tiempo (a un tiempo 0.5 segundos por ejemplo), de manera que por mucho que se pulse no se ejecutará la rutina asociada.

- **Update:** Durante la fase de Update (actualización) se realizan todos los cálculos de física y movimientos de objetos en la aplicación, se tienen en cuenta tales cálculos como actualización de la posición del jugador, posición de enemigos, proyectiles, animaciones de personajes, gestión de sonidos, gestión de estados de juego, detección y resolución de colisiones, etc.

  Todas las actualizaciones de movimientos que se realizan son en base al tiempo transcurrido entre el estado de juego anterior (ejecución del bucle principal anterior) y el actual. Así se consigue que éstos se actualicen independientemente de la máquina donde se ejecuta el programa, evitando que en ordenadores más rápidos se ejecute la aplicación más rápida o al revés.

Los movimientos que se realizan han de basarse siempre en el movimiento anterior, evitando de todas las maneras posibles la ejecución de bucles de gran tamaño que generen la secuencia de estos durante la misma iteración de bucle principal.

Si en algún momento requerimos de la creación de objetos o destrucción en esta parte lo que se suele hacer es evaluar de alguna forma lo que se va a utilizar, con el fin de realizar la reserva de memoria al sistema en bloques grandes, así ahorrando llamadas al sistema.

- **Render:** Durante esta fase se realiza el render (pintado) en pantalla de todos los objetos que intervienen en el juego. Es una parte que depende bastante de la tecnología empleada, y se tienen en cuenta factores como por ejemplo qué tipo de algoritmo de pintado utiliza, qué técnicas de ocultación de objetos se utiliza, si se renderiza todo lo que hay o sólo lo que ve la cámara, qué caras de los objetos se renderizan, etc.

  A nivel del desarrollador del videojuego se pueden tener cosas en cuenta como por ejemplo: qué primitiva de pintado es más optima para la tarjeta gráfica y así modelar los objetos utilizando triángulos, cuadrados, etc., utilizar diferentes modelos de detalle para los objetos para renderizar los que se ven más cerca a un nivel de detalle mayor (más caras en el objeto) y los demás a uno menor, aprovechar llamadas de render para renderizar más de un objeto a la vez, etc.

- **Finalización:** Una vez terminada la ejecución de la aplicación es necesario liberar toda la memoria utilizada, y así evitar los conocidos *memory leaks,* fragmentos de memoria que se han quedado huérfanos en el sistema reservados por nuestra aplicación y que no pueden volver a ser utilizados.

  Uno de los consejos a seguir a la hora de liberar la memoria que ha utilizado un programa es hacerlo de forma inversa a cómo se ha pedido esta, liberando memoria de objetos más específicos a objetos más genéricos, ir escalando de una forma gradual en el árbol de dependencias entre clases que tiene nuestro programa. De esa forma garantizaremos el funcionamiento correcto de esta fase. Vale la pena decir que para ello podemos buscar herramientas que facilitan la tarea, como librerías o complementos para nuestro entorno que nos avisan de la memoria que hemos dejado sin liberar.

Durante la realización de un proyecto es útil tomar medidas acerca de cómo evolucionan algunas de las características que dan información sobre una aplicación en tiempo real. A medida que va creciendo y se van añadiendo nuevas funcionalidades, recursos, etc., es útil tener datos de referencia con los que comparar para decidir si añadir o quitar cosas. Todo esto se realiza siempre pensando en la plataforma donde se va a llevar la aplicación, ya que todo depende de las restricciones de cada una de ellas.

Podemos encontrar medidas de todo tipo. Por ejemplo: cómo se escala el programa basándonos en datos como consumo de memoria de la aplicación, accesos a memoria, fallos de lectura, etc. Pero una de las más utilizadas es la denominada FPS (frames-per-second). Ella nos dice el tiempo que se tarda entre cada ejecución del bucle principal, lo que se tarda entre frame y frame.

Como referencia de FPS para aplicaciones en tiempo real nos podemos basar en o bien llegar a los 30 FPS (eso son 30 ejecuciones del bucle principal por segundo: 0.333 milisegundos por iteración) o 60 FPS (60 ejecuciones del bucle principal por segundo: 0.166 milisegundos por iteración).

Un valor de FPS situado entre estos dos dará una buena sensación de aplicación en tiempo real, mientras que una que esté situada por debajo se verá como “ralentizada”.

2.1.2. Planificación

Para la realización del juego se ha propuesto una planificación que se intentará seguir en la medida de lo posible. El período de diseño e implementación del proyecto corresponde a los meses entre Marzo y Septiembre de 2012, la planificación es la siguiente:

- Marzo:
    - Empezar a aprender a utilizar la herramienta Unity mediante tutoriales, aprender a programar en C#.
    - Definir la metodología de trabajo individual y de grupo.
    - Definir y escribir el documento de diseño de Launchageddon.

- Abril:
    - Seguir con el aprendizaje de las herramientas a utilizar.
    - Empezar a definir los módulos que se necesitan para el proyecto y sus comunicaciones.
- Mayo:
    - Desarrollo del módulo de física y su validación.
- Junio:
    - Desarrollo de los módulos de lógica de juego, cámaras y puntuaciones.
    - Definición del documento final de diseño de juego, sus objetivos y características principales.
    - Integración con el equipo de las partes desarrolladas hasta la fecha.
- Julio:
    - Finalización del desarrollo de lógica de juego, cámaras y puntuaciones y validación.
    - Empezar el desarrollo de la interfaz de usuario, realización de animaciones para las interfaces, transiciones entre los diferentes estados de juego.
    - Integración con el equipo de las partes desarrolladas hasta la fecha.
- Agosto:
    - Finalización de los módulos de interfaz de usuario y sus animaciones, transiciones de juego.
    - Integración con el equipo de las partes desarrolladas hasta la fecha.
    - Diseño de niveles para el juego.
    - Inclusión de efectos gráficos y de sonido.
- Septiembre:
    - Finalización del juego.
    - Finalización de la memoria.

2.1.3. Viabilidad

Después del estudio del arte previo y el diseño del concepto de Launchageddon se puede llegar a afirmar que el proyecto es viable.

A nivel de presupuesto no será necesaria la inversión en ningún tipo de software ni de hardware, ya que todas las herramientas de desarrollo que se utilizarán tendrán licencia gratuita para su uso.

A nivel de viabilidad técnica y dada la planificación el proyecto será técnicamente posible.

A nivel de viabilidad legal no existirán problemas de este tipo. No se utilizarán datos de carácter personal de ningún tipo y las licencias de programas a utilizar estarán compradas previamente o serán gratuitas.

2.1.4. Objetivo y características

Todo videojuego ha de tener unos objetivos compuestos por una serie de tareas e hitos a conseguir. En primer lugar podemos realizar una separación bien básica según el tipo de estos. Podemos encontrar videojuegos que se basan en cumplir objetivos en solitario; si hablamos de juegos para un jugador y objetivos a cumplir en cooperativo; para juegos que se desarrollan entre varios jugadores a la vez.

Los objetivos de nuestro videojuego Launchageddon son unos objetivos enfocados para jugar en solitario. Estos consisten en conseguir puntos mediante la destrucción de objetos clave y estructuras de un escenario, dicha destrucción se realiza lanzando a nuestro jugador hacía el escenario.

A continuación encontramos una imagen que muestra el logo de Launchageddon, y ejemplos sobre los tipos de objetos clave que encontraremos y los tipos de estructuras (figura 2.2).

Figura 2.2: Logo del juego (arriba), objetos clave (centro), ejemplo de estructuras (abajo)

A pesar de ser un juego en solitario se ha incluido soporte para ranking de puntuaciones. Eso añade un factor de competitividad importante al videojuego, cada una de sus pantallas posee un ranking de los mejores jugadores que han jugado a ella, de tal manera el juego se convierte en un elemento de competición entre amigos. Una vez has pasado un nivel puedes volver a jugar a él para intentar batir la puntuación que ha conseguido otra persona.

El juego está organizado por diferentes niveles que se pueden seleccionar para jugar. Una vez entramos en un nivel tendremos una serie de rondas para completar nuestro objetivo. A lo largo de cada ronda podremos realizar una serie de acciones como por ejemplo, observar el escenario en primera persona para planificar una estrategia, planificar la

estrategia, lanzarnos hacia el objetivo, ejecutar acciones en el aire mientras se vuela, etc. Además todas y cada una de ellas ligadas con una pequeña animación.

Para completar los objetivos, además podremos hacer uso de toda una serie de trajes y cascos que nuestro jugador podrá vestir y combinar para producir diferentes efectos a la hora del lanzamiento y a la hora del choque. Seguidamente encontraremos la definición de los trajes y cascos que se han utilizado, y la forma en la que funciona la puntuación del juego.

a) Trajes

Seguidamente encontramos todos los trajes que se han diseñado e implementado para complementar la jugabilidad. En total son cuatro y cada uno de ellas afecta a la forma en la que te desplazas por el escenario. En este caso para la forma en la que vuelas por él.

Hay trajes que te permiten interactuar con el jugador mientras se vuela, hay otros que una vez activados todo el proceso de juego es automático (ver figura 2.3).

Los trajes son los siguientes:

- **Traje normal:** Este traje es el más básico de los cuatro. Su funcionamiento se basa en un tiro parabólico en tres dimensiones. Es un traje que una vez se inicia el lanzamiento actúa automáticamente siguiendo la trayectoria definida por el ángulo que hemos seleccionado hasta que se produce el choque contra el escenario.

- **Traje de paracaidista:** El traje del paracaidista es un traje que tiene la característica de permitir modificar mientras se vuela la trayectoria que sigue el jugador.
  Su funcionamiento está dividido en dos fases. En su primera fase sigue la trayectoria definida por un tiro parabólico como el traje anterior, con la diferencia que cuando interactuemos con algún botón de acción del juego se iniciará su segunda fase. En la segunda fase el tiempo de juego se ralentizará permitiéndonos utilizar las teclas de dirección para desplazar nuestro jugador perpendicularmente a la trayectoria parabólica que sigue. Esta forma de interactuar con él nos facilitará el corregir y/o modificar nuestro movimiento para intentar acceder a zonas a las que no podíamos,

aprovechando el margen de tiempo que nos da la ralentización de la acción y el movimiento perpendicular rápido.

- **Traje de hombre bala:** Al seleccionar el traje bala podremos hacer uso de una velocidad inigualable para impactar contra nuestros objetivos. Es útil cuando queramos penetrar en muros para llegar al interior de edificios.
  Su funcionamiento también está dividido en dos fases. La primera actúa como los dos trajes anteriores, se selecciona la trayectoria para efectuar un lanzamiento parabólico, al llegar al punto que queramos pulsamos un botón de acción para que empiece la segunda fase. En la segunda fase se parará el tiempo mientras nuestro personaje está suspendido en el aire y podremos escoger una nueva trayectoria de lanzamiento, esta vez utilizando un movimiento rectilíneo uniformemente acelerado. De esta forma podremos seleccionar con la máxima precisión nuestro objetivo y sumar una velocidad mayor cuando se llegue al impacto.

- **Traje de antigravedad:** Para acabar, tenemos el traje de antigravedad, que nos permitirá ignorar todas las leyes de gravedad que afectan al mundo, así como provocar que nuestro personaje rebote en todos los choques que se produzcan.

  La trayectoria de movimiento que realiza es un movimiento rectilíneo uniforme. Al empezar se selecciona la dirección en la que se quiere lanzar el personaje, éste sigue la trayectoria hasta que choca con un objeto. Al chocar con el objeto la resolución del choque se basa en física de sólidos rígidos y contempla dos pasos. En un primer paso se supone que el objeto contra el que chocamos es estático con el fin de calcular la velocidad de rebote del personaje. Posteriormente se calcula la velocidad de respuesta del objeto contra el que se choca como si éste no fuera estático. Entonces se le aplica al mismo la velocidad lineal y la velocidad angular de respuesta.

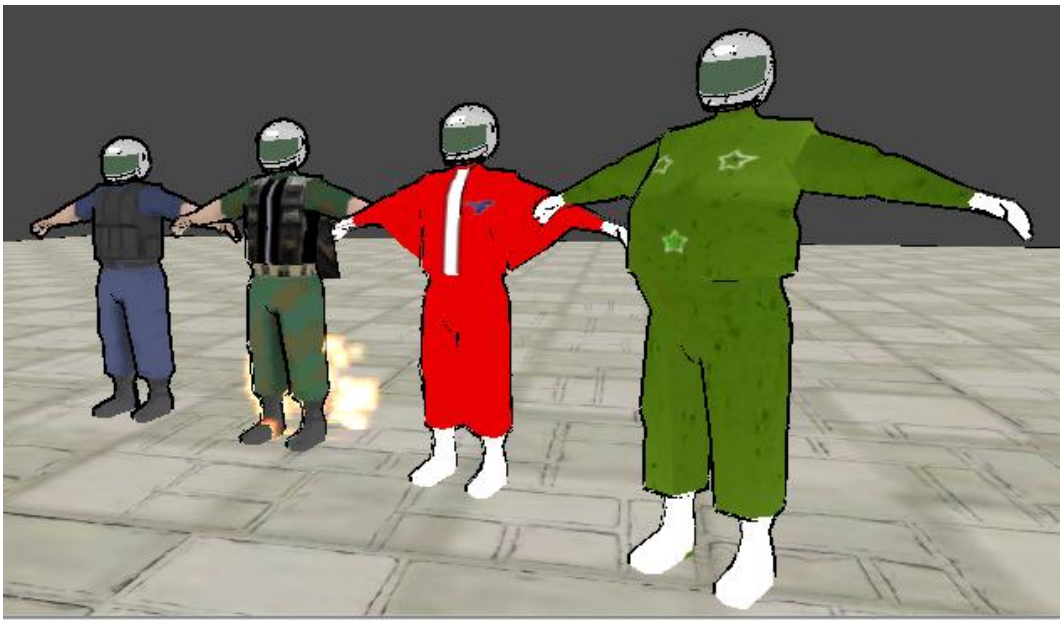

Figura 2.3: Diseño de trajes, de izquierda a derecha: normal, hombre bala, paracaidista y antigravedad

b) Cascos

Una vez explicados todos los trajes que podremos utilizar seguiremos con los cascos que podrá vestir nuestro jugador. En total se han implementado cuatro tipos de cascos diferentes. La principal característica de ellos es que producen efectos una vez nuestro personaje choca contra las estructuras objetivo del escenario, aunque también podemos encontrar alguno que afecta a la forma en la que vemos el escenario.

Los cascos son los siguientes:

- **Casco normal:** Éste casco no produce ningún beneficio, prácticamente se utiliza para evitar que nuestro personaje sufra daños al chocar contra el escenario.

- **Casco de hacha:** Al vestir el casco de hacha podremos partir las estructuras contra las que choquemos en dos, su utilidad es la de poder penetrar en los edificios rompiendo las paredes.

- **Casco explosivo:** Si vestimos el casco explosivo e impactamos contra el escenario liberaremos instantáneamente una bomba que tendrá un amplio radio de efecto, nos servirá para producir grandes daños en las estructuras.

- **Casco científico:** Al utilizar el casco científico podremos ver a través de los edificios para buscar los objetos clave de destrucción que nos darán el máximo de puntos, los edificios se volverán transparentes.

En la figura 2.4 se puede observar el diseño final para cada uno de ellos.

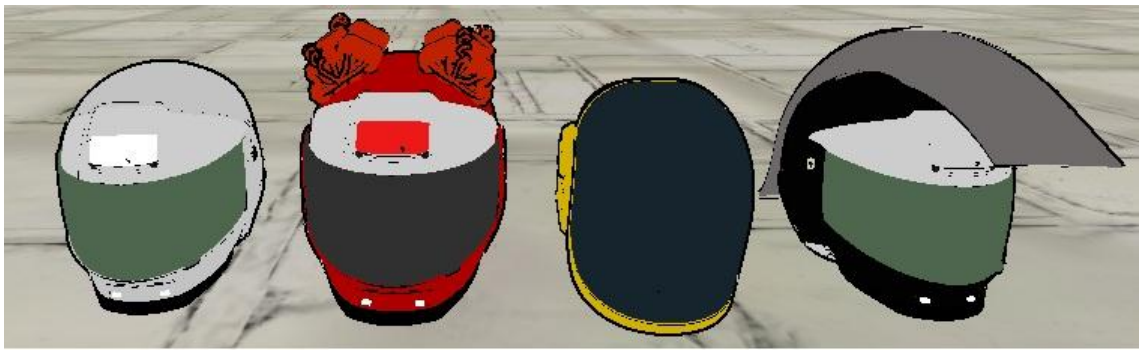

Figura 2.4: Diseño final de los cascos, de izquierda a derecha: normal, explosivo, científico y hacha

c) Combinaciones especiales

Uno de los temas que se ha contemplado al pensar en la jugabilidad es que cada uno de los trajes y cascos dieran características únicas, que se pudiera discernir de una forma fácil entre ellos para que cada uno de los efectos que producen fueran fáciles de recordar e hicieran la experiencia más dinámica.

No obstante se pensó en modificar de una forma sutil los efectos producidos al juntar determinados trajes y cascos, basándonos en la premisa de únicamente hacerlo para aumentar los efectos que producían, sin cambiar por ello el comportamiento que tienen al usarlos por separado.

Seguidamente pondremos las combinaciones dignas de ser comentadas. Las que no se listan será por que sus efectos son los mismos que si actuaran por separado:

- **Traje de paracaidista + Casco de hacha:** Al combinar estos complementos sumaremos la capacidad de maniobra que nos ofrece el traje de paracaidista con el efecto de cortar estructuras del casco de hacha. Ello provocará que al momento de impactar

podamos seguir modificando la trayectoria del movimiento mientras cortamos la estructura.

- **Traje de hombre bala + Casco de hacha:** Sumar estos dos complementos nos proporcionará la velocidad del traje de hombre bala con la capacidad de romper y penetrar en estructuras del casco de hacha. Todo ello sumado nos ayudará a llegar más lejos mientras se realiza el choque, y conseguir con ello más puntos de destrucción.

- **Traje de antigravedad + Casco explosivo:** Para finalizar encontramos la combinación de traje de antigravedad con el casco explosivo. A cada una de las paredes que toquemos al rebotar con el traje se le colocará una bomba que hará explosión segundos después.

d) Puntuaciones

Para Launchageddon se valoró la característica de diseñar un sistema de puntos. Gracias a él se podría valorar qué jugador cumplió los objetivos de un escenario o si superó los rankings de los demás jugadores. El sistema de puntos diseñado se basa en dos vertientes: una contempla los puntos que se consiguen destruyendo el escenario y la otra en la destrucción de los objetos clave que encontramos por el escenario.

Para el cálculo de puntos se tiene en cuenta la reacción que provoca la velocidad a la que impacta el primer objeto. Este provoca en el segundo una velocidad angular y una lineal. Cuanto más grande es la velocidad lineal provocada se conceden más puntos, ya que significa que se ha impactado más de lleno en el móvil.

El sistema de puntos está escalado de tal forma que la búsqueda de objetos clave para su destrucción por el escenario sea más importante, que por el contrario dedicarse a destruir las demás estructuras del escenario. Por ello cuando se impacta con un objeto especial no se aplica el cálculo de puntos anteriormente mencionado, si no que se suman un número

especial de puntos, lo bastante grande para hacer que ello valga la pena para conseguir nuestro objetivo en el nivel.

Más adelante **en sistema de puntuaciones** se detallará su implementación.

2.2. Metodología de trabajo

El flujo de trabajo y proceso ingenieril seguido para la realización del proyecto se puede resumir en el siguiente diagrama que se muestra (figura 2.5). Por él han pasado todos los archivos generados del proyecto y en él queda resumido el proceso seguido.

Cada una de las tareas realizadas puede dividirse en tres fases distintas: concepto, desarrollo y validación. Y cada una de las fases consta de diferentes tareas a realizar:

- **Fase de concepto**: En primer lugar se realiza una búsqueda de todo lo necesario para la tarea en cuestión, búsqueda de información por la red, búsqueda en libros, teoremas, leyes, ejemplos a considerar, problemática de la tarea, soluciones conocidas, etc. Luego se toman decisiones acerca de qué método para la resolución de la tarea se va a implementar, luego se empieza a pensar la estructuración que tendrá a nivel de código y una vez diseñada se pasa a la fase de desarrollo.

- **Fase de desarrollo**: Se empieza la codificación del algoritmo, clases necesarias, dependencias con librerías, patrones de diseño útiles para la codificación, etc. Una vez finalizado se valora la refactorización de código para intentar hacer el código más legible y organizado, también se intenta evitar dependencias entre partes y funciones de diferente naturaleza. A la par se intenta optimizar el código, evitando bucles innecesarios y minimizando el uso de operaciones costosas.

- **Fase de validación**: Durante la fase de validación se realizan dos tipos de tests: uno unitario donde se ejecuta paso a paso el archivo generado para comprobar su funcionamiento correcto. Si su funcionamiento es correcto se realiza un segundo test en conjunto para ver cómo interactúa con los otros elementos del proyecto. Si ésta parte falla (por test unitario) se vuelve a la fase de desarrollo para intentar encontrar y

solucionar los problemas que han ocurrido. Si por el contrario falla por test en conjunto hay que volver a la fase de concepto con la idea de estudiar mejor cómo realizar la tarea.

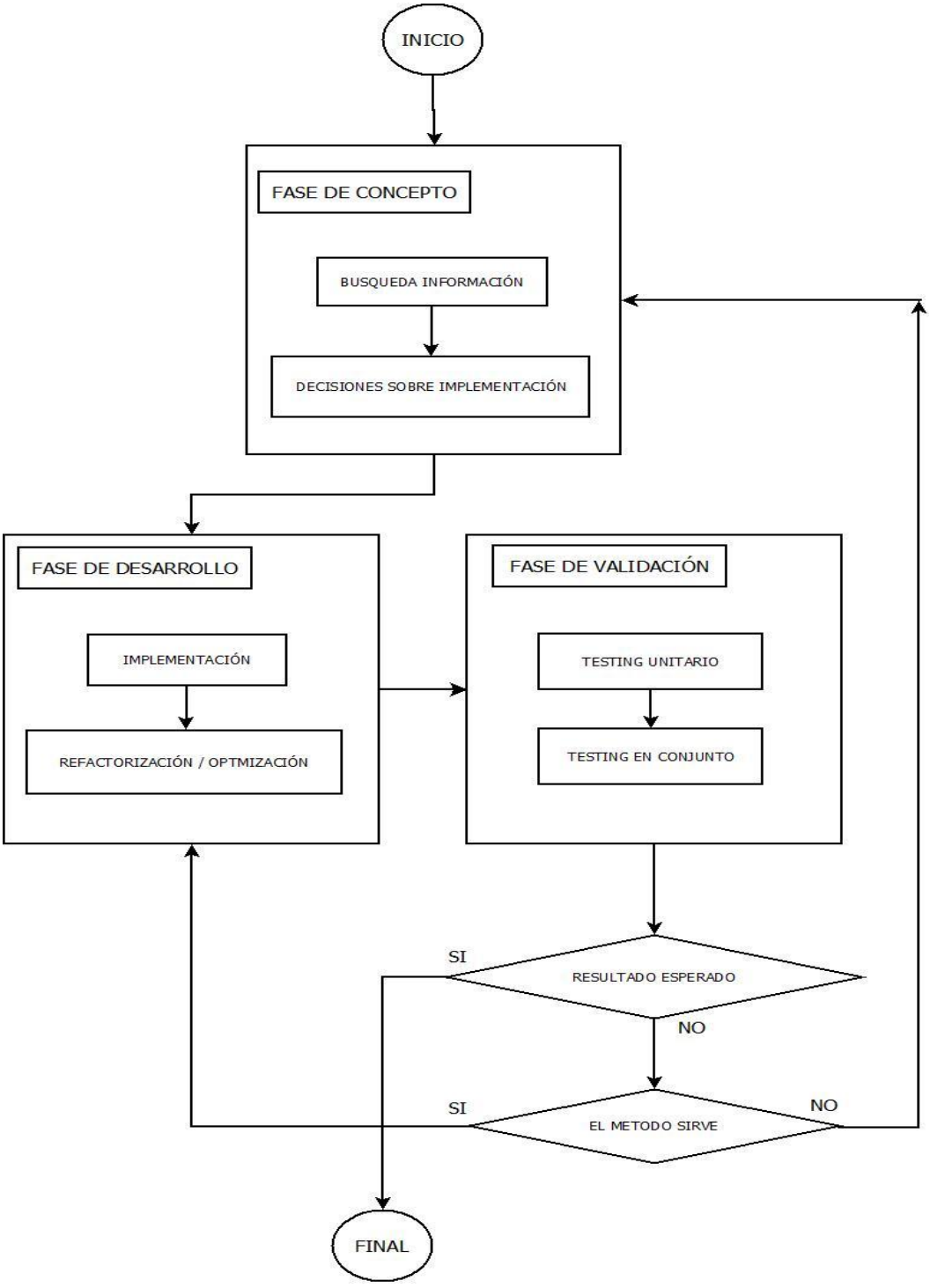

Figura 2.5: Metodología de trabajo

2.3. Diagrama de módulos

Seguidamente detallaremos el proceso de desarrollo de nuestro proyecto, que corresponde a hacer un zoom de la parte de Gameplay de la figura 1.7. Aquí veremos cada uno de los módulos contenidos en esa parte y cada uno de ellos tratará una parte bien diferenciada de éste (se pueden observar en la figura 2.6).

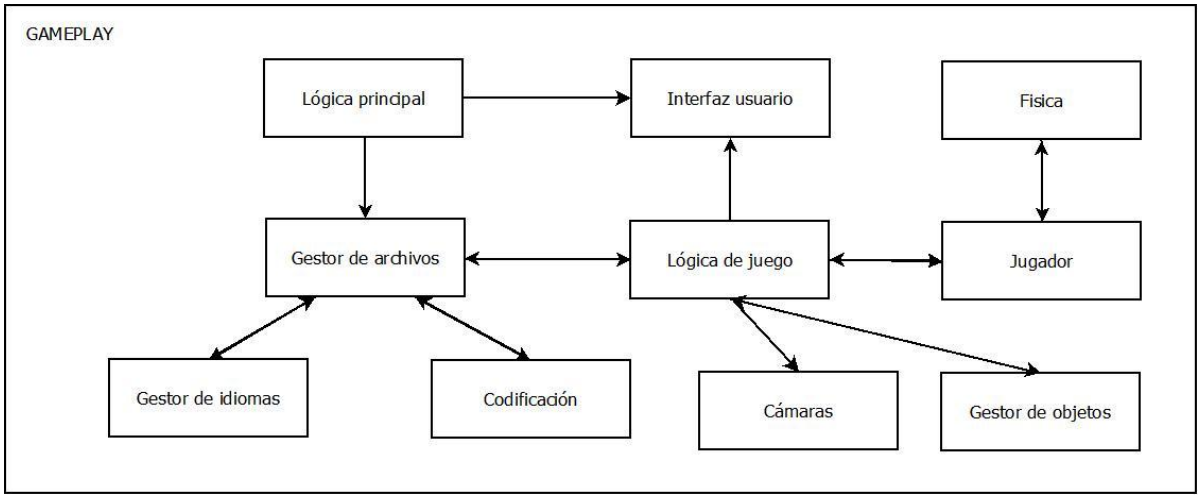

Figura 2.6: Diagrama de módulos del juego

Seguidamente encontraremos la parte de desarrollo para cada uno de los módulos:

- En **módulos de lógica** encontraremos la explicación de: lógica principal y lógica de juego, ellos corresponden a la lógica de juego que se ha implementado tanto para el punto de entrada a la aplicación principal como a la lógica en juego, ambos son las clases centrales del proyecto sobre las cuales se montan los demás módulos.

- En **módulo de GUI** se encuentran las diferentes interfaces de usuario que se han implementado.

- En **módulo de física** encontraremos los detalles sobre la física que se ha implementado para Launchageddon.

- En **módulo de cámara** encontraremos la explicación detallada de la cámara que se ha implementado para Launchageddon.

- En **módulos de acceso a datos** se incluye el desarrollo del módulo gestor de archivos, módulo de codificación y módulo gestor de idiomas.

- En **sistema de puntuación** encontraremos detallado el sistema que se ha tenido que implementar para introducir puntuación en el videojuego, eso incluye el funcionamiento del módulo gestor de objetos y las comunicaciones entre módulos necesarias.

- El módulo jugador es el encargado de gestionar toda la información del jugador, no hay un apartado para él ya que su funcionamiento prácticamente es el de establecer relaciones entre otros módulos para iniciar la partida y activar el módulo de física o por ejemplo informar a la lógica de juego que un choque se ha producido, para que arranque el sistema que cuenta las puntuaciones.

2.4. Módulos de lógica

Las diferentes fases en la que está dividida la jugabilidad de un juego es lo que se denomina como estados de juego y su lógica se encuentra implementada dentro del módulo lógica de juego. Su implementación suele ser bastante complicada ya que hay que pensar como gestionar todas las variables que intervienen sin que haya problemas de sincronización, comportamientos no contemplados, etc. Como mencionamos en los requisitos de aplicaciones en tiempo real, las tareas que se implementan dentro del bucle principal de ejecución han de ser lo más rápidas posible, deberemos diseñar esta parte lo más natural y fluida posible.

Esto significa que hay que seguir un patrón bien claro a la hora de hacer las cosas. El que hemos utilizado se basa en permitir realizar acciones dentro del videojuego tipo actualización de puntos, procesado de eventos de ratón y teclado, etc, sólo en determinados momentos (cuando la fase de juego lo permita). Gracias a ello conseguimos crear transiciones de juego y la sensación de que el juego avanza. Cuando se ejecuta el bucle principal de juego se ha de saber en qué estado de juego se está. Gracias a ello tan solo ejecutaremos las tareas permitidas en ese estado de juego.

Para el juego se han implementado dos lógicas de juego: lógica de la aplicación principal y la lógica en juego.

a) Lógica aplicación principal

El motivo de existencia de la lógica para la aplicación principal es para tener un punto de entrada a la aplicación. Éste se encarga de secuencializar todos los procesos e inicializar todos los módulos que intervienen en el juego. Además gestiona el menú principal de la aplicación, el cual veremos detallado en el aparto de **diseño de menús**.

b) Lógica de juego

La implementación de la lógica de juego principal corresponde al diagrama de estados que podemos observar en la figura 2.7.

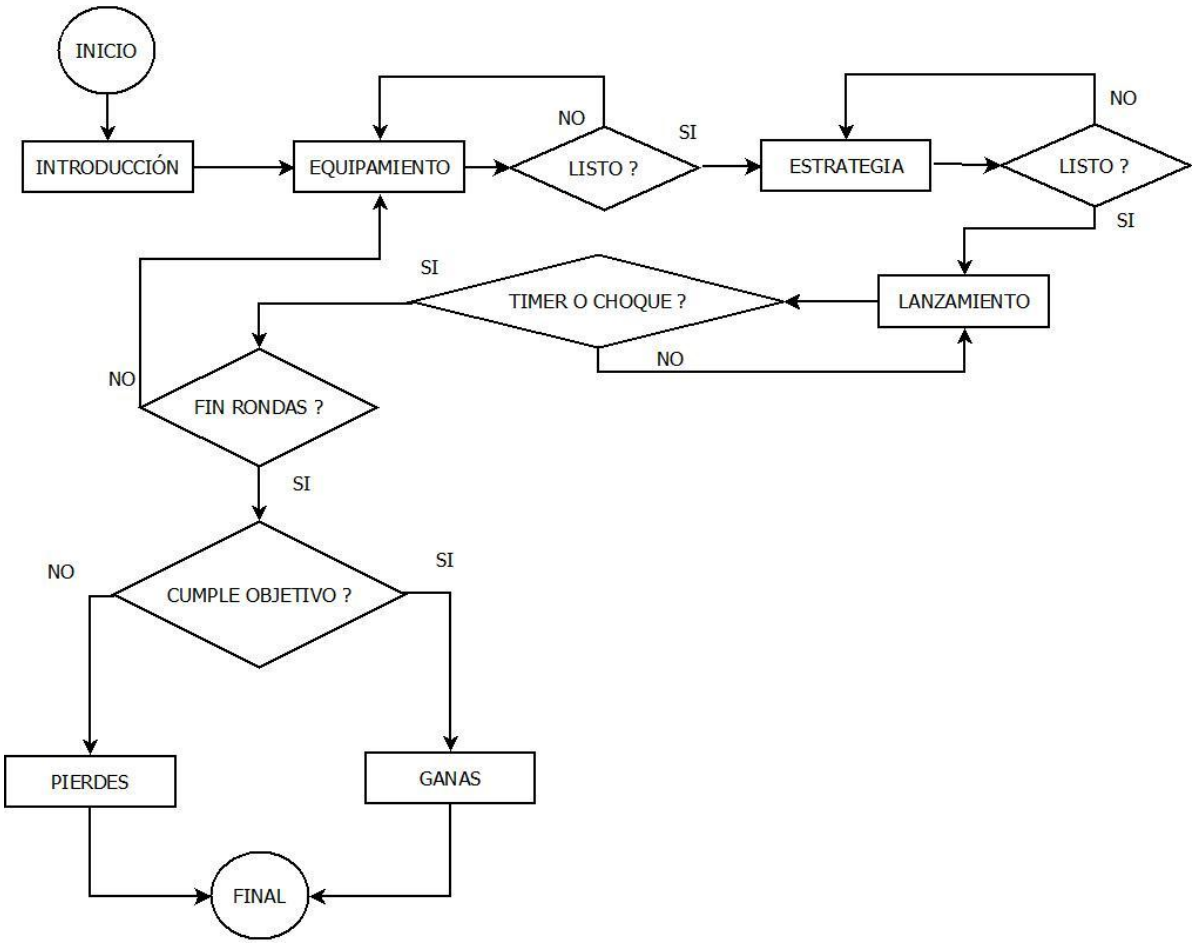

Figura 2.7: Diagrama de estados de juego

Cada uno de los cuadrados de la figura 2.7 hacen referencia a un estado de juego. Dentro de él se podrán realizar determinadas acciones como controlar el personaje, utilizar el teclado, ratón. Los estados también definirán las transiciones automáticas de cámara que se realizarán, la aparición o no de información en pantalla, interfaz de usuario, reproducción de efectos, etc. Los rombos de la figura 2.8 representan las condiciones que se han de cumplir para pasar de un estado a otro, estas condiciones están basadas en eventos que ocurren en el juego, y son modificadas desde dentro de la lógica o desde fuera empleando el envío de mensajes entre módulos.

2.5. Módulo de física

Siempre que se requiera para el desarrollo de un videojuego dotar de un realismo al mundo que creemos deberemos simular la naturaleza del mundo real, ya sea para ofrecer una experiencia de juego creíble y dotar de un movimiento lo más real posible a los objetos que interactúan, que chocan, que navegan, que vuelan, en definitiva, toda aquella tarea que necesite de algún tipo de cálculo de cinética, dinámica y fuerzas, deberemos ahondar en temas de física.

Antes de ponerse manos a la obra hay que valorar, entre otras cosas: qué se quiere y de dónde se parte, es decir que es lo que se necesita implementar y si en la plataforma en la que se desarrolla la aplicación ya existe alguna librería que nos facilite la tarea. En función de esto cabe decidir qué se quiere utilizar de la herramienta o qué se quiere implementar de nuevo, pensando en las mecánicas de juego que se desean, si van a requerir o no la modificación o inclusión de física, etc.

En el módulo de física trataremos las cosas que hemos necesitado para el proyecto y las que se han implementado, incluyendo las diferentes teorías que se proponen para solucionar estos problemas. En física para enfrentarse a un mismo problema casi siempre suele haber diferentes puntos de vista y soluciones, ninguno de ellos suele ser el acertado para todo. Por ejemplo si se busca reducir el coste de computación usaremos determinado método, mientras que si se busca la precisión de cálculo usaremos otro.

La herramienta que utilizamos es Unity, una de las cosas que incluye es precisamente abundante documentación e implementación de física. La física que implementa Unity hace uso de la librería PhysX [Phy], una capa de de software intermedia desarrollada por NVidia Corporation destinada a llevar a cabo cálculos de física directamente en la tarjeta gráfica.

El desarrollo de la librería PhysX, facilitó en gran medida a muchísimas compañías que se dedican al desarrollo de videojuegos una tarea bastante compleja como es la inclusión de física en un juego. La librería les permite abstraer durante el desarrollo del juego la parte de física, eso les permite aumentar la productividad y dedicarse a escribir código de juego en lugar de código para simulaciones complejas de física.

Los temas que hemos abordado el diseño de la física corresponden a soluciones a problemas de mecánica clásica, la utilizada para describir el comportamiento de elementos macroscópicos a nivel de partícula y sólido rígido que viajan a velocidades pequeñas, en comparación a la velocidad de la luz.

Entre los temas se incluyen cinemática [Anexo II], dinámica [Anexo III], detección de choques y resolución de choques [Anexo IV], para los interesados en el tema de programación en el anexo número VI se incluye el diagrama de clases y relaciones utilizados para la implementación de la física.

En la figura 2.8 podemos observar el diagrama de módulos que corresponde a la parte de física y las relaciones con los demás módulos del proyecto.

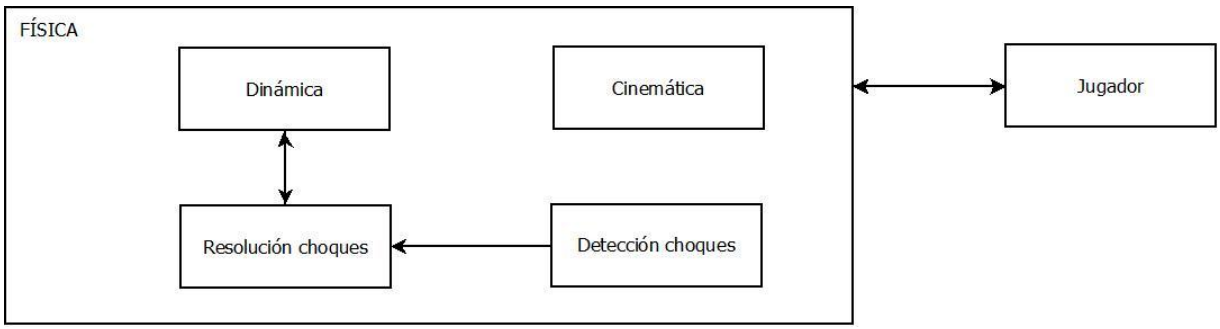

Figura 2.8: Diagrama de módulos de física

El módulo de física está compuesto de todas las partes que hemos comentado antes, cinemática, dinámica, detección de choques y resolución de choques. Seguidamente especificaremos cada uno de ellos.

2.5.1. Cinemática

El módulo de cinemática es el encargado de realizar toda la simulación de movimientos para el jugador que se han implementado, todos en tres dimensiones, en él se incluyen:

- Movimiento rectilíneo uniforme
- Movimiento rectilíneo uniformemente acelerado
- Movimiento parabólico

Para implementar éste módulo se ha utilizado el esquema que se puede observar en la figura 2.9.

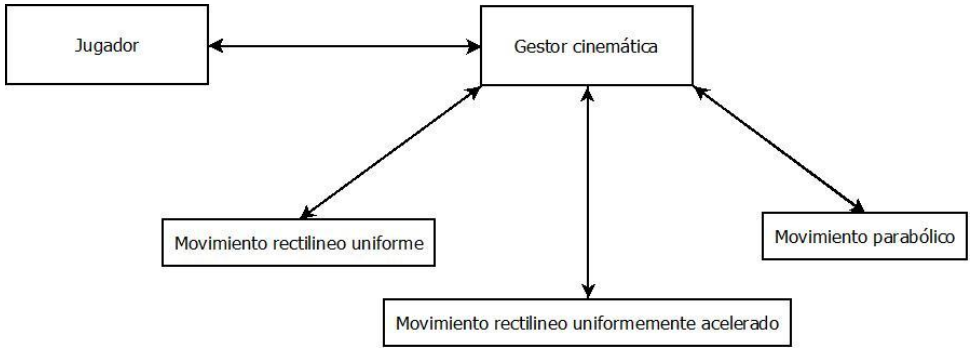

Figura 2.9: Diagrama módulos cinemática

El módulo que gestiona al jugador se comunica con el gestor de cinemática, éste internamente realiza la simulación con el movimiento específico que se esté simulando (rectilíneo uniforme, uniformemente acelerado o parabólico) sin que el jugador tenga que saber de la existencia de ninguno de ellos. En el anexo número II se puede consultar en detalle más acerca de cinemática.

2.5.2. Dinámica

El módulo de dinámica es el encargado de aplicar a los teoremas necesarios a la hora de resolver los choques, entre ellos encontramos:

- Conservación cantidad de movimiento
- Conservación de la energía
- Coeficiente de restitución

Ellos se utilizan para calcular la respuesta a los choques que se producen en el juego, una explicación más detallada de cada uno de ellos se puede encontrar en el anexo número III.

Éste módulo es utilizado por el modulo de resolución de choques para implementar la resolución de choques, más adelante encontraremos la explicación de éste módulo.

2.5.3. Detección de choques

Para la detección de choques se hace uso de la tecnología que ofrece Unity, el motivo de ello es que es un aspecto bastante delicado a la hora de diseñar un motor de física. Esta suele ser una de las tareas que más tiempo ocupan en computación en los motores de física. Ello se debe a que hay que tener constancia a cada instante de simulación de todos los objetos que intervienen en pantalla, de sus posiciones, sus velocidades, etc.

Para la implementación de éste apartado se ha tenido que aprender a utilizar la interfaz que ofrece Unity para ello [Rig]. Su relación con los demás módulos es bien sencillo, una vez éste detecta un choque informa al módulo encargado de la resolución de choques de ello.

2.5.4. Resolución de choques

El módulo encargado de resolución de choques se encarga de resolver las colisiones que el módulo de detección le envía. Para ello tiene en cuenta el estado anterior de los objetos que chocan y los teoremas y leyes de física conservativa que le facilita el módulo de dinámica.

Para el proyecto se ha valorado el estudio e implementación de dos métodos diferentes, ellos son los siguientes:

- Respuesta de choques de partículas
- Respuesta de choques basada en solidos rígidos

El motivo de ello es para probar la funcionalidad de ambos y ver cómo se comportan para decidir cual se utilizará finalmente. Ambos se han implementado utilizando su variante en las tres dimensiones.

Una explicación detallada de cada uno de los métodos implementados puede encontrarse en el anexo número IV.

2.6. Módulo de cámara

Para observar lo que ocurre en pantalla en el juego se ha implementado un tipo de cámara en tercera persona.

Las cámaras en tercera persona suelen ir siempre a una distancia fija de un punto del escenario o siguen a un determinado objeto de éste. Para Launchageddon se ha valorado la codificación de las dos modalidades: se puede establecer una cámara fija en tercera persona que realice transición de zoom hacia fuera o hacia dentro y gire en torno a un punto del mapa, y una cámara en tercera persona que realiza un seguimiento a determinada distancia de un objeto que está en movimiento.

Antes de empezar a explicar la teoría que se ha implementado para la implementación de la cámara en tercera persona se definirán unos cuantos conceptos sencillos acerca de cámaras en el mundo de los gráficos. Podemos observar algunos de ellos en la figura 2.10.

- **Posición de la cámara:** corresponde a la posición en tres dimensiones donde se sitúa la cámara.
- **Planos near y far:** cuando colocamos una cámara en la escena hay que definir dos planos: el near (cerca) define el campo de visión mínima que tendrá la cámara, el far (lejos) definirá el campo de visión máximo que se podrá ver a través de la cámara.

- **Ojo de la cámara (eyepoint):** corresponde a la posición del mundo donde la cámara mira.
- **Vector cielo:** corresponde al vector que define cómo está colocada la cámara, es el vector que define su orientación, lo normal es ponerlo a x=0; y=1; z=0.
- **Fustrum de la cámara:** corresponde a todo el espacio del mundo que se puede ver a través de la cámara viene definido por seis planos, near, far, left,right, bottom y top.

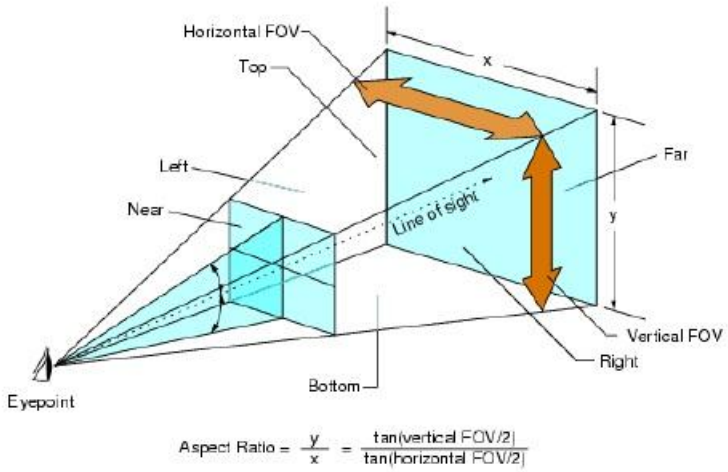

Figura 2.10: Características de las cámaras

Para actualizar el movimiento que realiza una cámara en tercera persona se necesitan dos ángulos que definen su funcionamiento, estos son los que se pueden observar en la figura número 2.11, el pitch y el yaw.

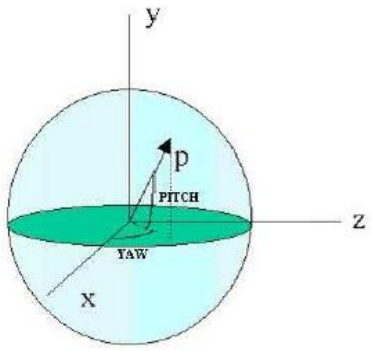

Figura 2.11: Esquema de los ángulos que definen la cámara

Las ecuaciones que definen cómo se ha de actualizar la posición de la cámara son las que podemos observar en (2.1).

$$\begin{aligned} x &= r * cos(pitch) * sin(yaw) + eye_x \\ y &= r * sin(pitch) + eye_y \\ z &= r * cos(pitch) * sin(yaw) + eye_z \end{aligned} \tag{2.1}$$

$D\acute{o}nde: (x, y, z) es\ la\ posición\ de\ la\ cámara,$

$pitch\ y\ yaw\ son\ los\ ángulos\ en\ radianes,$

$eye_x, eye_y, eye_z, es\ el\ ojo\ de\ la\ cámara.$

Por otro lado también podemos definir el vector hacia dónde mira la cámara, se han implementado dos, el primero define el vector donde mira la cámara con la dirección hacia donde mira el objeto que sigue, y el otro para una cámara que mira a una posición concreta.

a) La cámara mira donde mira el objeto que sigue

Para esto se calcula el vector distancia entre el ojo de la cámara y su posición, a esta distancia se le suma el vector dirección donde mira el objeto multiplicado por un valor escalar, el resultado se normaliza, de esta forma conseguiremos que nuestra cámara mire hacia la dirección donde mira el objeto que sigue.

b) La cámara mira a una posición concreta

Para ello se calcula el vector distancia entre el ojo de la cámara y su posición; el vector resultante se normaliza, eso nos dará el vector unitario que define la dirección hacia dónde ha de mirar.

2.7. Módulo de GUI

Éste módulo va a realizar la labor de intermediario entre el programa y el usuario, entre otras cosas le va a abstraer del uso de acciones del programa mediante línea de comandos, o

análogos, etc. uno de los objetivos, a la hora de diseñar una GUI es que sea útil, clara y concisa.

Normalmente las ventanas que componen una GUI están compuestas de diferentes tipos de elementos, entre ellos podemos encontrar botones, flechas, etiquetas con información, selectores, menús desplegables, los cuales van a servir para informar de algo al usuario, advertirle. En definitiva abstraer funcionalidades que realiza el programa a un click de ratón o a un vistazo de un desplegable.

Para el desarrollo del videojuego hemos buscado, además de proporcionar una interfaz que cumpla con los puntos mencionados, convertirla en un elemento que resalte más. Para ello la hemos dotado de animaciones, intentando mostrarla moviéndose con naturalidad cuando aparecen los botones y elementos por pantalla. Para ello hemos creado la interfaz mediante una máquina de estados que se encarga de desplazarla por la pantalla y modificar el tamaño de los elementos en función de algunas condiciones, en cada una de las interfaces implementadas encontraremos la figura correspondiente de sus estados y condiciones.

Además de todo lo comentado todas las interfaces diseñadas para el juego se crean en función de la resolución de pantalla. Eso quiere decir que la interfaz se verá igual en cualquier tipo de pantalla, el tamaño de los botones y sus posiciones se ajustarán en consecuencia de ella.

Básicamente hemos programado dos interfaces distintas, una para el menú principal y otra para el menú en juego. Seguidamente encontraremos algunos de sus detalles de implementación y unos pequeños esquemas para hacernos una idea de cómo están montados.

2.7.1. Menú principal

Para el diseño del menú principal de la aplicación (figura 2.12) se ha realizado un concepto de un menú lateral, a través del cual podremos acceder a toda una serie de submenús que nos llevarán hacia otras opciones del programa, en él podremos encontrar las opciones de nuevo

juego para ir al selector de niveles, editor de niveles para editar niveles, configuración de parámetros del juego y salida para salir del juego.

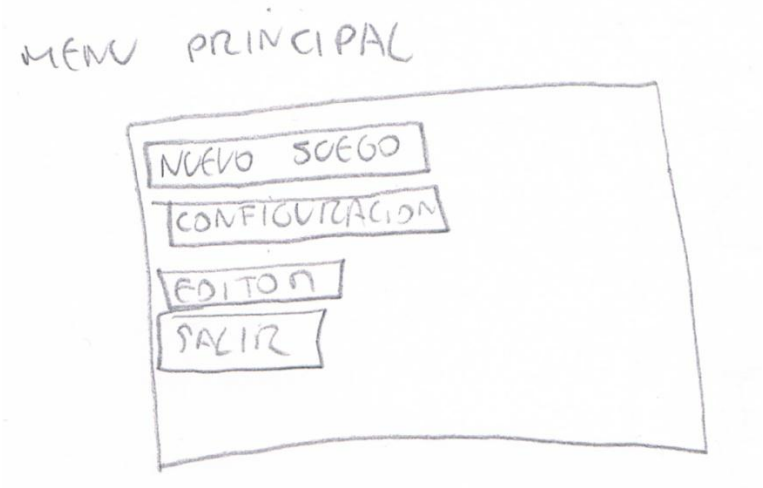

Figura 2.12: Concepto de menú principal del juego

Al pulsar en cualquier botón de los que tenemos iremos al submenú correspondiente. La transición entre menú y menú se realizará de una forma bastante suave, escondiendo los botones hacia la izquierda y apareciendo los nuevos hacia la derecha, siguiendo un diagrama de flujo como el que podemos observar en la figura 2.13.

La animación básicamente se basa en desplazar los botones de izquierda a derecha, cuando llega a una posición de referencia se acaba la animación. Cuando se pulsa un botón se inicia la animación de esconder botones en ellos se mostrarán los actuales, mientras que cuando vuelvan a aparecer para dar paso al siguiente menú saldrán los botones correspondientes al nuevo menú produciendo un efecto bastante dinámico.

Aquí veremos algunos de los submenús interesantes de mencionar. Por ejemplo “nuevo juego” o “editor de niveles”, ya que los demás únicamente mostrarán información y otros botones. En “configuración” encontraremos botones para seleccionar entre los diferentes idiomas y para realizar codificación y descodificación de archivos. No obstante éste último sólo aparecería en la versión de desarrolladores de la aplicación, así podrían acceder

de una forma más fácil los archivos. Por otro lado en créditos encontraremos algo de información acerca de los desarrolladores del juego.

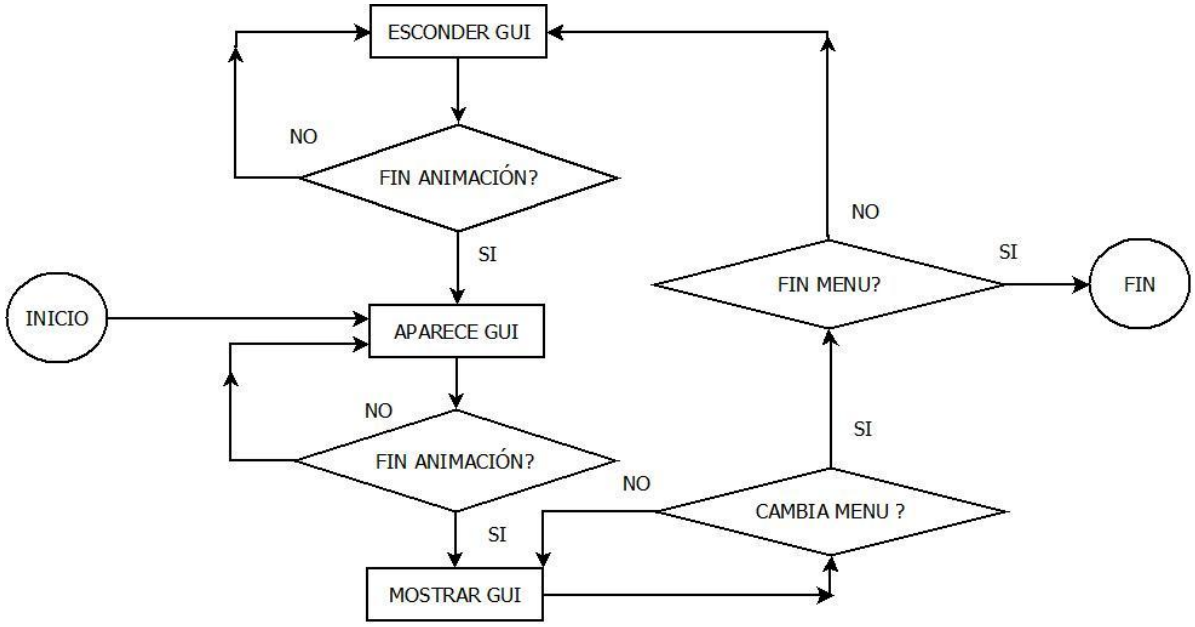

Figura 2.13: Diagrama de flujo animaciones para el menú principal

Por ejemplo al pulsar en el submenú de "nuevo juego" encontraremos lo mostrado en la figura 2.14. En la imagen se han marcado algunos puntos para enfocar varios de los detalles diferentes que contendrá.

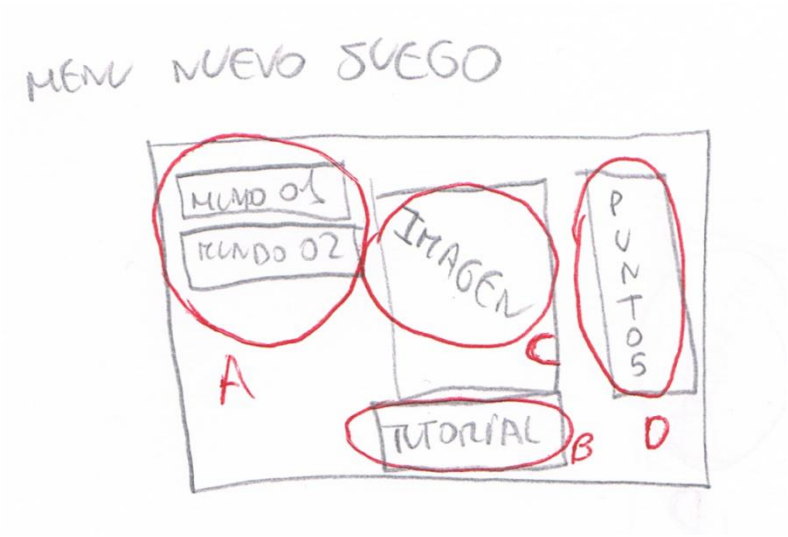

Figura 2.14: Concepto de submenú de nuevo juego en menú principal de la aplicación

Cómo podemos observar en la figura 2.14 existen diferentes focos de interés (puntos A, B, C y D), ellos son los siguientes:

- **A:** esta parte corresponde a todos los diferentes niveles que contiene el juego, esta parte se crea de una forma dinámica, se creará en tiempo de ejecución de la aplicación utilizando un archivo que hay de configuración. A través de él se lee toda la información para crear la etiqueta que contiene (ejemplo en imagen: Mundo01 o Mundo02), se leerá la información de tutorial (foco B) y se realizará la lectura de todos los rankings de puntos (foco D). Al situarnos encima de él con el cursor se desplegarán las demás partes del menú (B, C y D).

- **B:** esta parte se encarga de mostrar la información de tutorial del nivel el cual hemos situado encima el puntero. Esta información nos dará pistas para completar el nivel.

- **C:** en esta parte podremos ver una imagen de referencia del nivel el cual hemos situado el puntero encima. Nos servirá para tener una vista previa del escenario que contiene el nivel.

- **D:** aquí podremos observar el récord de puntuaciones que se han conseguido para este nivel. Nos mostrará de una forma ordenada las personas que mayor puntuación han hecho (arriba el mejor, abajo el peor), siempre mostrando los cinco mejores jugadores. En el momento de finalizar un nivel esta información se actualizará mostrando siempre la última registrada.

En la figura 2.15 podemos observar el concepto de diseño para el submenú de editor de niveles.

En ella podemos observar también varios focos de interés (puntos A, B y C), ellos son los siguientes:

- **A**: este botón servirá para acceder al editor de niveles (parte implementada por el compañero Jordi Cartes).

- **B**: en esta zona se mostrarán los nombres de las personas que han creado los niveles en el editor.

- **C**: en esta zona podremos seleccionar para jugar los niveles creados por cada una de las personas.

Figura 2.15: Concepto de submenú de editor de niveles

2.7.2. Menú en juego

Para la implementación del menú de juego se pensó en implementar uno que fuera bastante útil para el control que iba a tener el juego, en este caso Kinect. Se tenía que pensar en algo que funcionara bastante bien con Kinect. De entre todos los bocetos realizados se seleccionó e implementó un concepto de sistema como el que se puede observar en la figura 2.16.

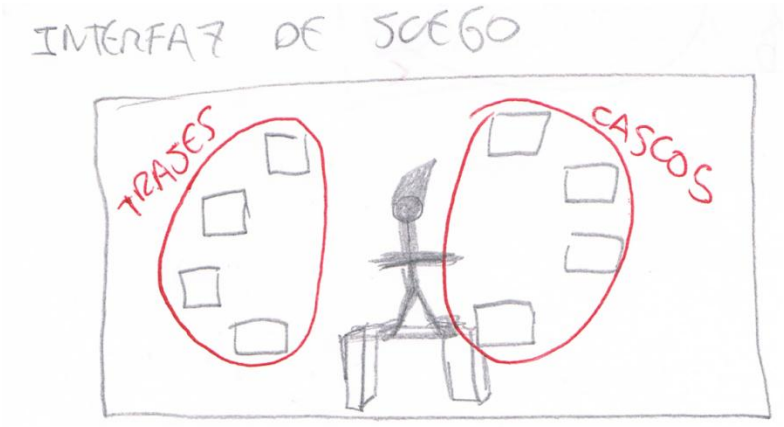

Figura 2.16: Concepto de diseño de la interfaz de usuario para el juego

Cómo se puede observar en la figura 2.16 la interfaz de usuario estará dividida en dos partes, a la izquierda de esta encontraremos los diferentes tipos de trajes que nuestro jugador podrá vestir y a la derecha los diferentes tipos de cascos con los que combinar.

Además si situamos el cursor encima de uno de los botones aparecerá una caja que explicará en qué consiste el traje o casco sobre el que estamos situados, ofreciendo el tutorial a la misma vez que se juega.

Al empezar el estado de juego que muestra esta interfaz los botones saldrán todos en la misma posición y de una forma fluida irán desplegándose conformando una animación que dará como resultado una semicircunferencia, al seleccionar uno de los botones su semicircunferencia se replegará de nuevo hasta desaparecer, la animación seguirá el diagrama de flujo mostrado en la figura 2.17, la condición de finalización de la animación la marcará el módulo de lógica de juego, una vez se seleccione el traje y el casco a utilizar éste se encargará de esconder los botones.

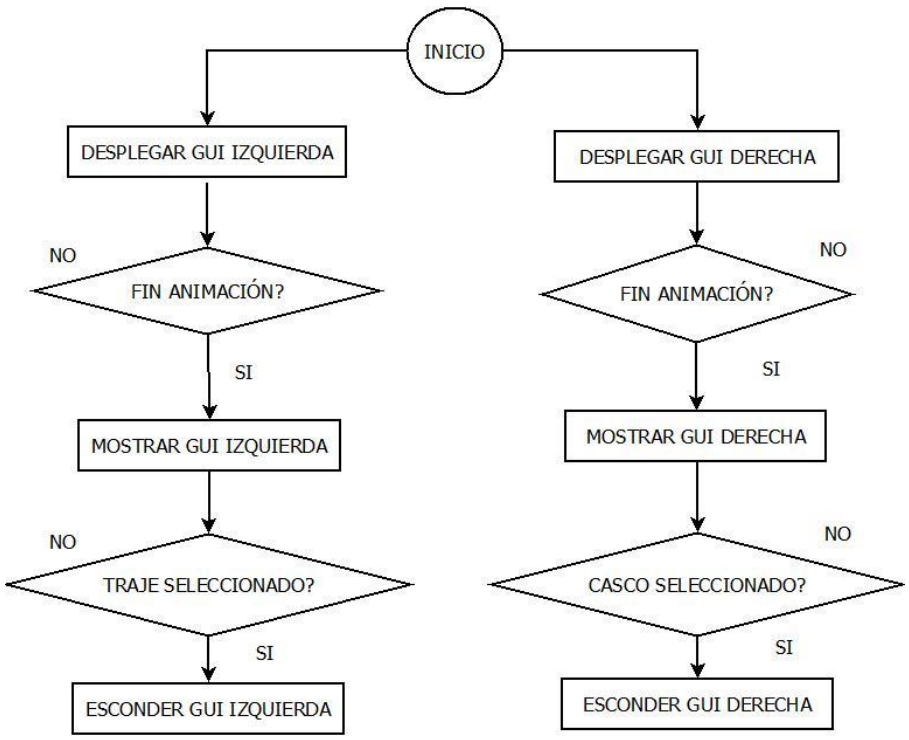

Figura 2.17: Diagrama de flujo para animaciones de GUI de juego

Esta interfaz de usuario no tendrá siempre el mismo número de botones a izquierda y derecha, o sí, todo depende de la configuración que tenga el nivel que se está ejecutando. Gracias a ello se podrán crear los niveles con restricciones de uso de trajes o cascos. Así el jugador deberá pensar mejor sus estrategias. Cuando hayan menos botones, éstos se situarán de una forma totalmente simétrica, dependiendo del número de ellos estos se colocarán en una coordenada diferente de la semicircunferencia que conforman.

2.8. Módulos de acceso a datos

Algunas de las partes que posteriormente explicaremos hacen uso de lectura y escritura a datos situados en disco, se utilizan archivos para crear los menús del juego, leer y grabar las puntuaciones de los niveles y acceder a funcionalidades de configuración e idiomas.

A lo largo de este apartado explicaremos como se ha realizado la tarea, qué clases se han implementado para ello y que medidas de seguridad se han tomado para que los archivos que utiliza el juego no puedan ser modificados por alguien que no sea el desarrollador del mismo. También se incluye la explicación sobre el funcionamiento del módulo de idiomas.

Seguidamente encontramos cada uno de ellos.

2.8.1. Módulo gestor de archivos

El acceso a datos es una de las tareas que más tiempo de computación requieren. El programa ha de depender al completo de la velocidad de acceso a disco que tiene el sistema.

Por ello se partió de la idea de realizar esta tarea únicamente en momentos puntuales a lo largo de la ejecución de la aplicación. En concreto se realiza al principio de su ejecución, en la fase de inicialización y carga. El objetivo de esto es llevar a memoria RAM todo lo necesario de los archivos de disco. Una vez en RAM su acceso en tiempo será mucho más rápido y no comportará mucha dedicación de la CPU.

La forma en la que interactúa el manager de archivos con las demás clases del proyecto es la que se puede observar en la figura 2.18.

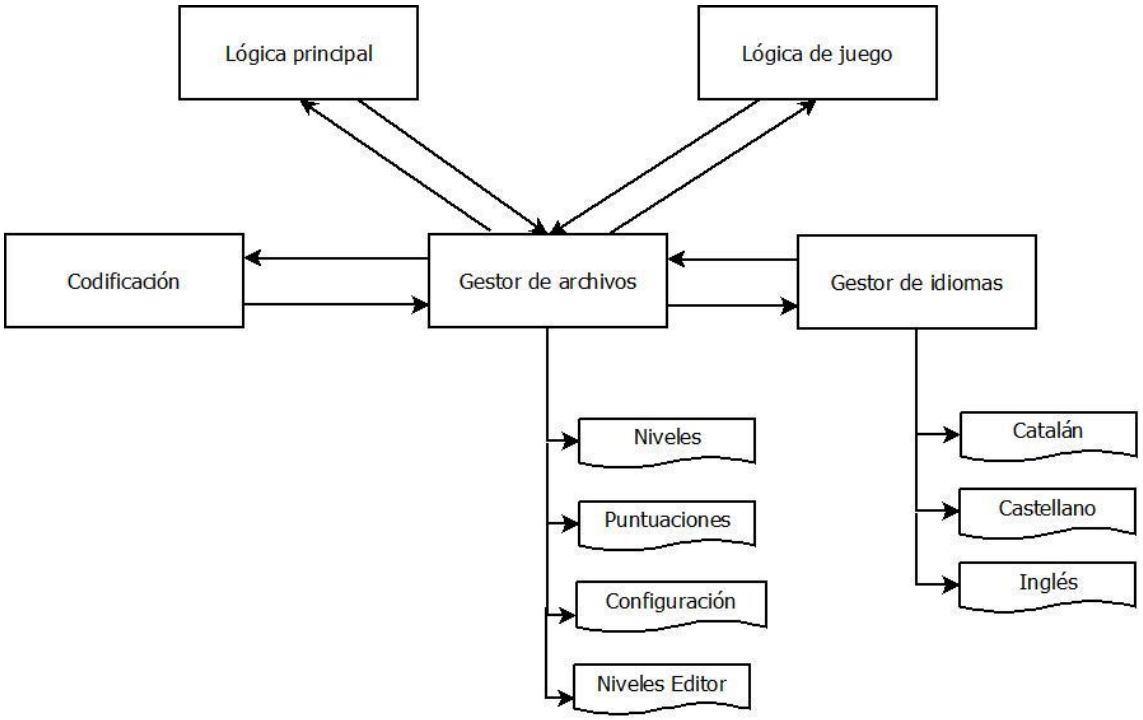

Figura 2.18: Diagrama de módulos de el sistema de acceso a archivos implementado

Los dos módulos centrales del proyecto Lógica principal y Lógica de juego en su etapa de inicialización piden al Gestor de archivos las cosas que necesitan. Mientras que Lógica principal pide todo el tema relacionado con la creación del menú principal, la lectura de niveles, puntuaciones de cada nivel e idiomas. Lógica de juego pide información acerca del nivel específico que carga, detalles como el número de lanzamientos disponibles, trajes y cascos que se pueden utilizar, así como información útil de tutorial para el nivel, información al finalizar el nivel, etc.

A lo largo del anexo número VIII se puede encontrar más información acerca de la estructura interna en programación que dispone el sistema.

2.8.2. Módulo de codificación

Que el proyecto implemente el acceso a archivos implica que estos se situaran en el ordenador de la persona que esté ejecutando el juego, ello comporta el problema que cualquier persona puede acceder a ellos y alterar su estado, provocando que el programa no funcione correctamente o incluso modificar condiciones que afectan a la jugabilidad, como por ejemplo cambiar los puntos que ha conseguido, añadir a un escenario más lanzamientos disponibles, bajar la puntuación para completar el nivel, en definitiva toda una serie de acciones que no han de estar permitidas.

Por ello se investigó acerca de la implementación de un algoritmo que ocultara la información que hay en los archivos, convirtiéndolos de alguna forma en ilegibles utilizando cifrado de información y buscando un método que ofreciera por igual un difícil descifrado.

Entre todas las posibilidades buscadas se encontró un algoritmo llamado Rjndael Managed [Rjn1], su especificación es denominada *Advanced Encryption Standard* (AES) desde el año 2002, año en el que fue declarado estándar de encriptación para datos por la comisión de seguridad de los Estados Unidos. Aparte de ello se necesitaba un algoritmo que fuera eficiente en tiempo de ejecución. Rjndael lo es implementado en software como en hardware cómo se puede leer a través de varios boletines oficiales [Rjn1].

Es un algoritmo ampliamente usado en encriptación de archivos, denominado de clave simétrica ya que emplea la misma clave para encriptar y desencriptar los datos. Es capaz de utilizar claves de 128, 192 o 256 bits. Para la implementación del proyecto hemos utilizado una clave de 256 bits. Para implementarlo se ha aprendido a utilizar el estándar definido por Microsoft en la API de C# [Rjn2], para el cual ya hay un algoritmo y con tan solo definir la clave de encriptado y el modo de cifrado basta.

2.8.3. Módulo gestor de idiomas

El motivo de querer localizar el juego a varios idiomas, es el de poder llegar a un mayor número de personas.

Para ello se creó un módulo nuevo que dependiera del gestor de archivos (figura 2.18). Que accedería a varios archivos de idioma según el que hubiera seleccionado en ese momento. La función principal que realiza este módulo es la de mezclar la información de idioma con la que utilizan la aplicación principal y la lógica de juego – que leen del gestor de archivos -. De ésta forma el acceso a la información se realiza de forma transparente al idioma que se tiene que cargar y desde la parte de la aplicación no se sabe nada acerca de éste proceso.

Para el juego se han añadido tres idiomas: catalán, castellano e inglés. No obstante el sistema se ha montado de tal forma que es muy fácil de escalar este aspecto. Para añadir un idioma nuevo tan solo hace falta coger uno de los existentes de muestra, traducir las frases al idioma y especificar en el archivo de configuración que se quiere cargar ese idioma.

2.9. Sistema de puntuaciones

Para diseñar el sistema se ha tenido que montar una infraestructura que gestiona un sistema de paso de mensajes entre los módulos que interactúan en el juego, de manera que la puntuación empieza a contar cuando es necesario. En este caso cuando se produce el primer choque del jugador contra el escenario. Podemos observar el sistema en la figura 2.19.

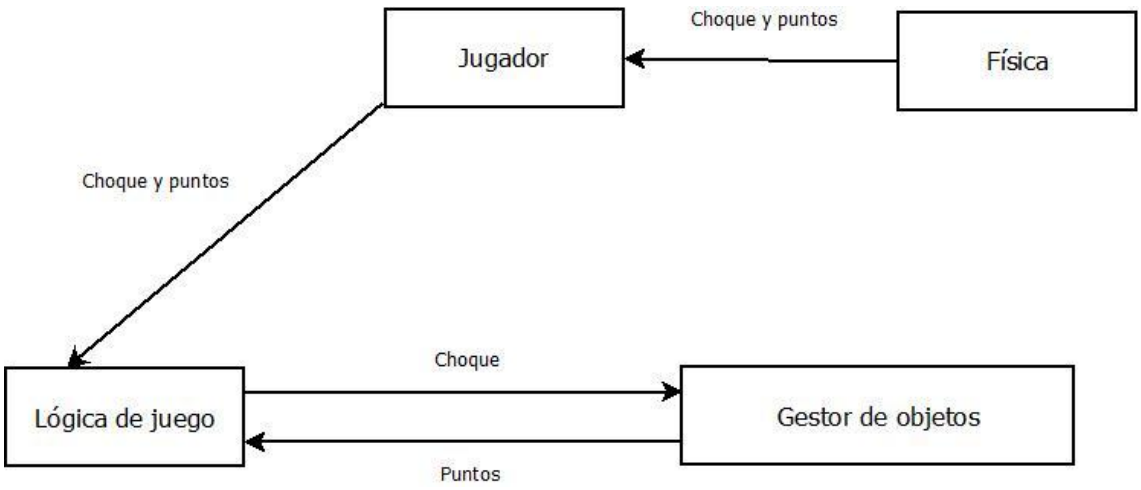

Figura 2.19: Diagrama de módulos del sistema de puntuaciones

Una vez el detector de colisiones del jugador (en el módulo de Física) choca contra algo se activa el procedimiento: el módulo de física resuelve la colisión pertinente y calcula la puntuación de este choque. Seguidamente la notificación de choque y el primer sumatorio de puntos al módulo jugador, éste envía al módulo encargado de gestionar la lógica de juego (lógica de juego) todo lo anterior. Este módulo avisa al módulo gestor de objetos del mapa (Gestor de objetos) del choque, que a su vez notifica el mensaje de choque a cada uno de los tipos de objeto que gestiona. Cuando los diferentes tipos de objetos que conforman las estructuras choquen debido a la reacción del primer choque enviarán de vuelta al módulo gestor de objetos su puntuación por separado, y este la enviará de vuelta a la lógica de juego para realizar el cálculo final de la ronda.

2.10. Efectos gráficos y de sonido

Una vez implementados todos los módulos necesarios para el funcionamiento de Launchageddon, se añadieron como extra algunos efectos gráficos y de sonido.

En un juego nunca pueden faltar elementos de este tipo, ya que hacen la experiencia de juego más gratificante. Cuando haces un juego hay que intentar que lo que hagas sea espectacular, que tenga elementos que hagan que destaque.

Para la implementación de efectos gráficos se ha utilizado un paquete de Unity que contiene recursos prefabricados para tal objetivo [Det], para utilizarlos en nuestro juego tan solo es necesario bajar el paquete, añadirlo a nuestro proyecto y aprender a utilizarlo.

Para implementar efectos de sonido se ha hecho uso de bibliotecas de internet gratuitas de sonidos [Sou].

3. Resultados

El apartado de resultados corresponde a la validación que se ha realizado de Lauchageddon, correspondiendo a las pruebas de funcionamiento que se han realizado para comprobar que las funcionalidades se han implementado de una forma satisfactoria.

Las pruebas consisten en dos tipos: un test unitario de funcionamiento donde se comprueba que el módulo funciona correctamente y un test de funcionamiento en grupo para comprobar que el módulo se integra bien con los demás módulos del juego.

Esta parte se ha divido en cuatro apartados:

- En primer lugar se mostrarán los resultados del juego en conjunto. Eso incluye física, lógicas de juego, cámaras, efectos visuales implementados y diseño final de Launchageddon.
- En segundo lugar se mostrará el diseño de las GUI para la aplicación principal y para el juego.
- Luego se mostrará el resultado de los módulos encargados de gestión de archivos, codificación e idiomas.
- Luego se detallarán los resultados de la integración con los otros proyectos [Enr][Jor].
- Al final encontraremos la valoración que han realizado personas a las que se les ha dejado probar el videojuego, para ello se ha elaborado una pequeña encuesta que han respondido cada uno de los usuarios que han jugado.
- Y para finalizar se escribirán unas cuantas líneas acerca del futuro del proyecto más allá de su entrega.

3.1. Resultados de Launchageddon

Para realizar la validación de la lógica de juego se mostrarán algunas imágenes sobre la secuencia de acciones que se realizan en Launchageddon, las cuales se podrán comparar con la figura 2.7 que muestra el diagrama de estados de juego en el capítulo de **desarrollo.**

En primer lugar encontramos el estado de introducción. En él podemos mediante una cámara aérea en tercera persona observar el escenario que vamos a jugar (figura 3.1). La cámara se moverá de una forma automática dando vueltas alrededor de un punto fijo, en este caso el centro del escenario.

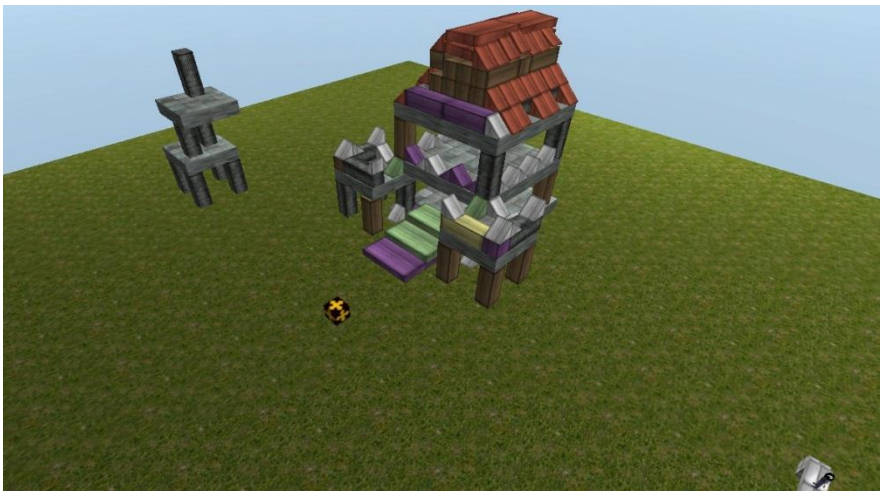

Figura 3.1: Estado de juego de introducción

Seguidamente nos encontraremos en el estado de juego de equipamiento (figura 3.2). En él deberemos seleccionar el traje y casco que nuestro jugador llevará en el lanzamiento. En éste momento también se podrá hacer uso de la cámara en primera persona implementada en el proyecto de Enric Martinez [Enr] para navegar por el escenario con la idea de buscar puntos débiles en las estructuras. Esto se hará pulsando el botón de acción.

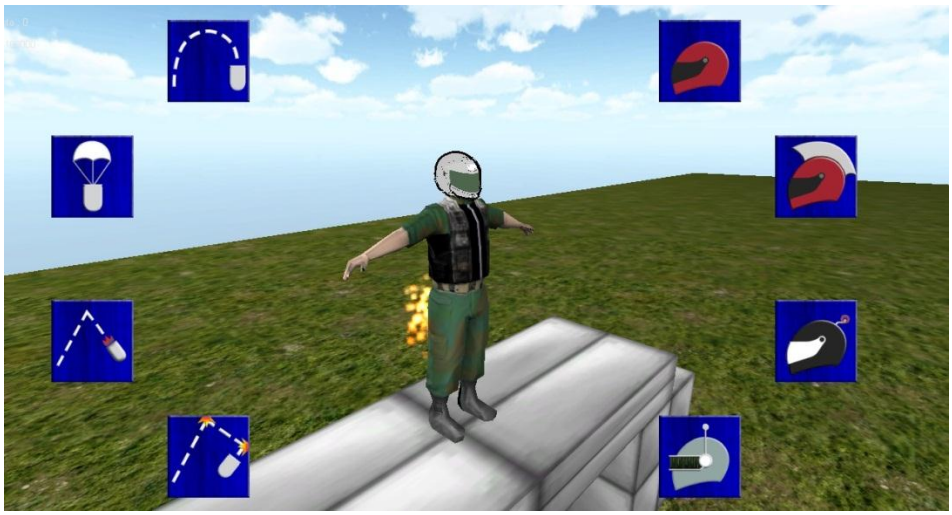

Figura 3.2: Estado de juego de equipamiento

Una vez seleccionado se producirá una transición de cámara (figura 3.3) que nos llevará a situar ésta justo detrás del personaje, desde un lugar donde se tenga buena visibilidad para planificar una estrategia de lanzamiento.

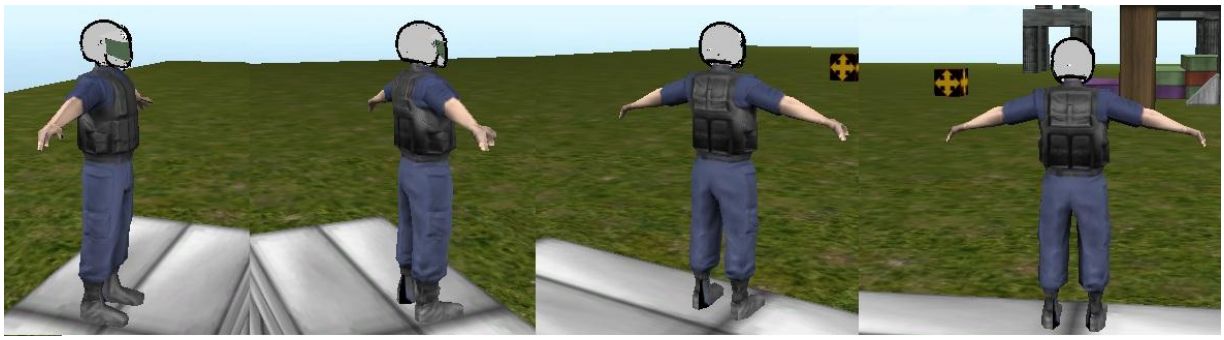

Figura 3.3: Transición de cámara entre los estados de juego equipamiento y estrategia

Una vez posicionada la cámara estaremos en el estado de juego estrategia. En él podremos seleccionar la trayectoria de disparo que hará nuestro personaje. Gracias a una previsualización de ésta podremos ver hacía dónde saldrá disparado el personaje (figura 3.4).

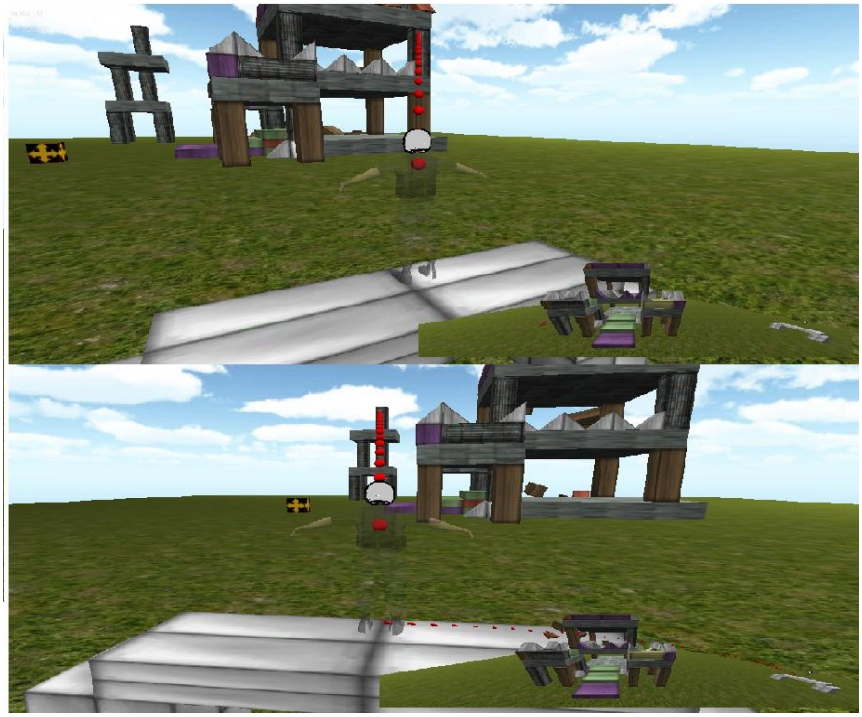

Figura 3.4: Estado de juego de estrategia

En la figura 3.4 podemos observar cómo las bolas rojas que aparecen en pantalla nos marcan la trayectoria de lanzamiento que seguirá el jugador si nos lanzamos con esa estrategia. Abajo a la derecha de cada imagen se puede observar una segunda cámara lateral en tercera persona que nos ayudará a planificar aún más nuestra estrategia.

Una vez hemos acabado pasamos al estado de juego de lanzamiento. Mientras se valida esta parte también se mostrará la física de cinemática y la resolución de choques para cada uno de los cascos y trajes.

En primer lugar en la secuencia que se muestra en la figura 3.5 observamos el movimiento parabólico.

Figura 3.5: Movimiento parabólico

Como se puede observar (figura 3.5) el personaje sigue en simulación a tiempo real perfectamente la trayectoria que anteriormente se ha simulado.

En la figura (3.6) podemos observar los movimientos rectilíneos implementados.

Una vez vistos los movimientos que realiza nuestro personaje pasaremos a mostrar los efectos que producen los choques. También veremos la reacción que produce la colisión.

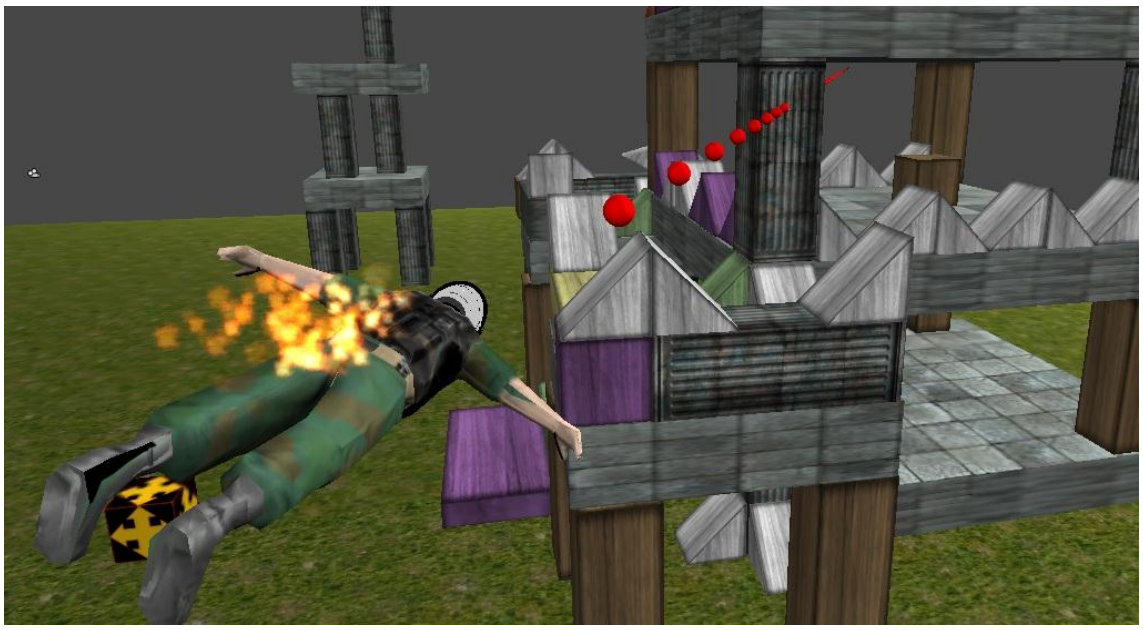

Figura 3.6: Movimiento rectilíneo uniforme y uniformemente acelerado

En la figura 3.7 podemos observar un choque que se realiza con el casco normal, éste se resolverá utilizando física de partículas utilizando las ecuaciones que se detallan en el anexo número IV.

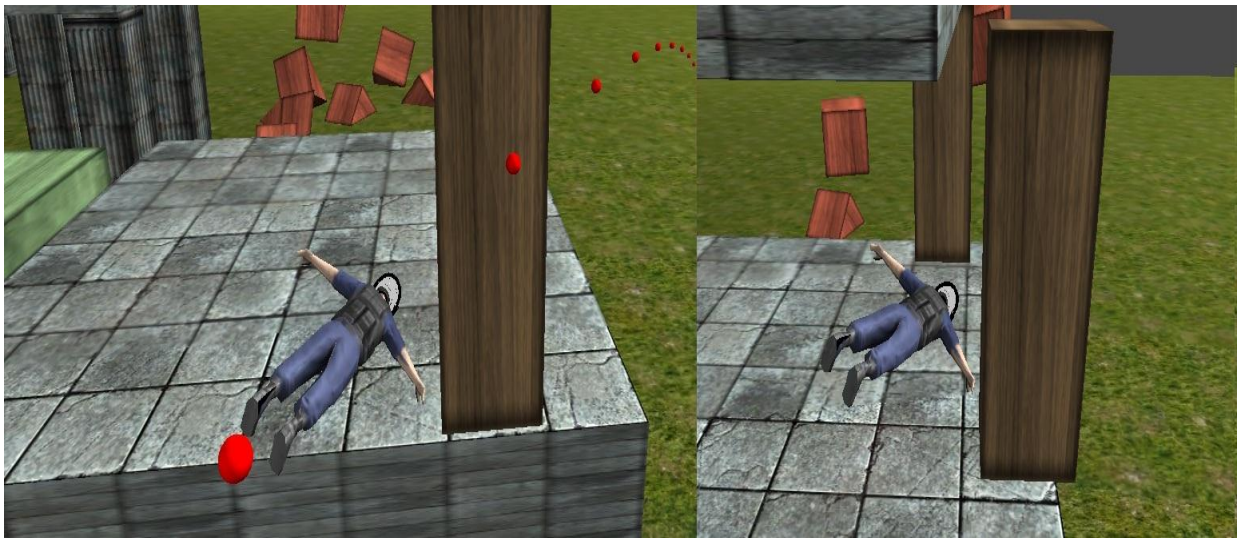

Figura 3.7: Choque resuelto con física de partículas

En cambio en la figura 3.8 observamos un choque resuelto con física de sólidos rígidos.

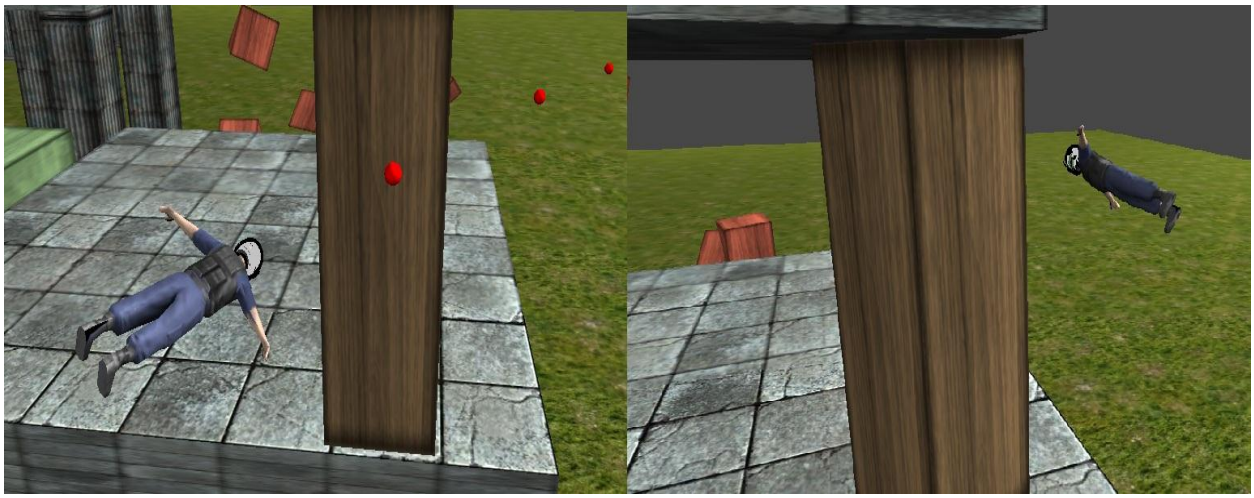

Figura 3.8: Choque resuelto con física de sólidos rígidos

Como se puede observar en la figura 3.8 la resolución de choques por física de sólidos rígidos es un modelo mucho más realista que el de física de partículas (figura 3.7). Mientras

que en la resolución mediante partículas el objeto contra el que se choca sufre un desplazamiento únicamente lineal, sin tener en cuenta la forma de éste, en la resolución por sólido rígido el jugador sufre una variación de su trayectoria acorde a la cara y la inclinación con la que se ha realizado el choque, de igual forma el otro objeto reacciona sufriendo una pequeña rotación como se puede observar en 3.8.

Como la simulación de choques con física de sólidos rígidos da mejores resultados, es la que se utilizará para la implementación final de Launchageddon.

Lo siguiente será mostrar la capacidad de la física de sólidos rígidos de simular choques contra objetos inamovibles. Para ello haremos uso del traje de antigravedad (figura 3.9).

Como observamos en la figura 3.9 una vez el jugador choca contra el objeto, éste sale despedido como si contra lo que hubiera chocado fuera inamovible.

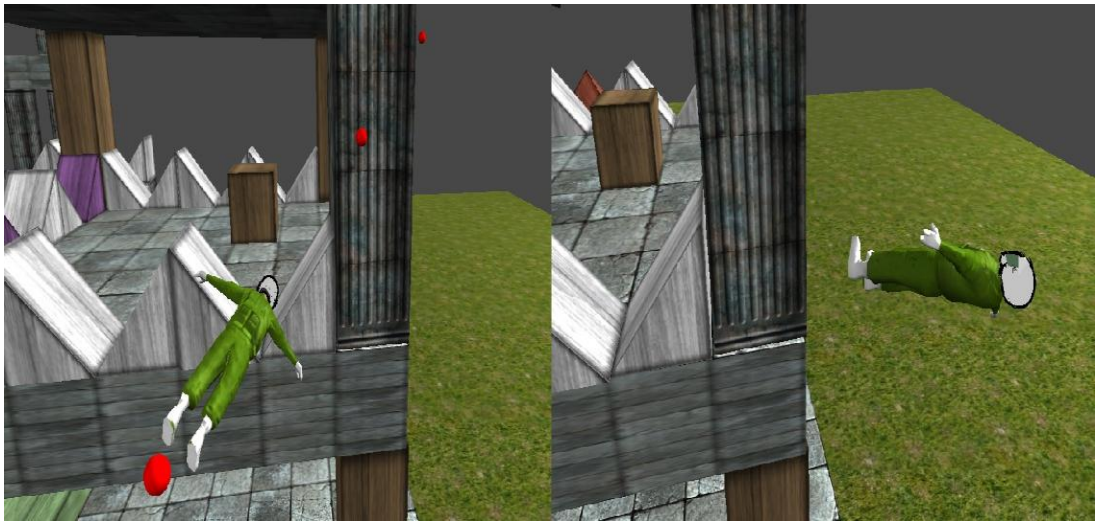

Figura 3.9: Simulación de choque contra objeto inamovible utilizando el traje de antigravedad

Para seguir con más choques veremos los demás efectos que producen los cascos cuando se choca, el primero de ellos será el casco destructor (figura 3.10).

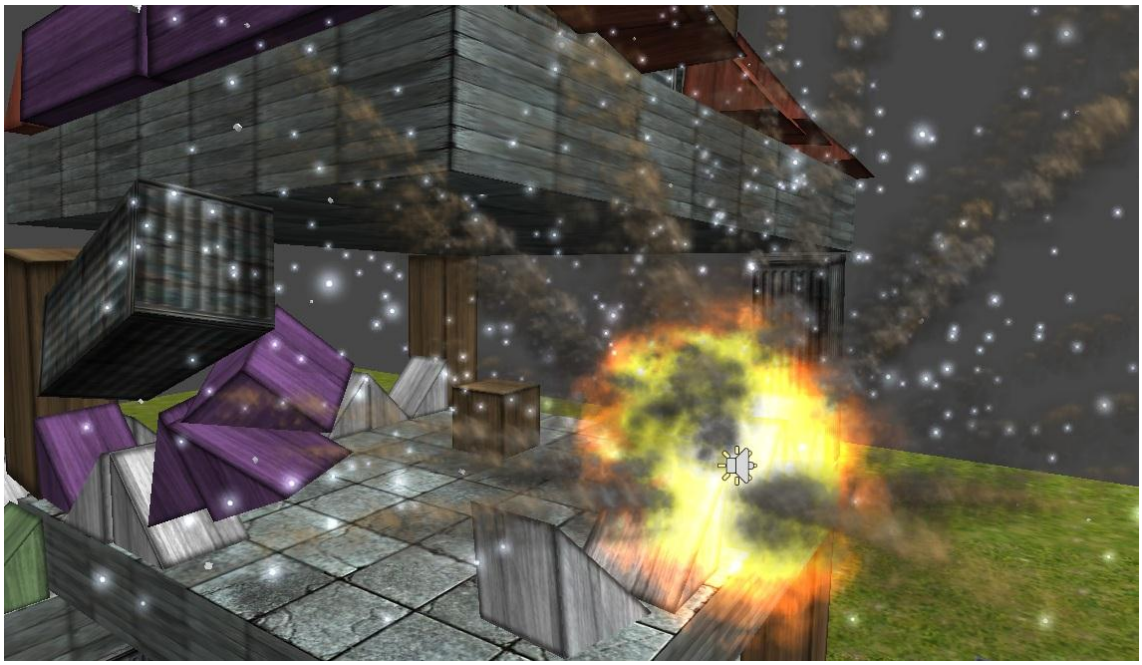

Figura 3.10: Choque con casco destructor

Una vez el jugador impacte contra el escenario se creará una explosión que provocará más daño del habitual y que además tendrá una zona de efecto. Todo aquel objeto que se halle dentro de ella sufrirá las consecuencias.

El siguiente que veremos será el casco de hacha (figura 3.11).

Figura 3.11: Choque con casco hacha

Cuando el jugador impacte contra el escenario podrá atravesar los edificios gracias al casco. Las piezas que toque se irán haciendo añicos, ellas son las grises pequeñas que se pueden observar en la figura 3.11.

El siguiente casco que veremos funcionando será el científico, gracias a él podremos ver a través de las paredes para buscar objetos clave (figura 3.12).

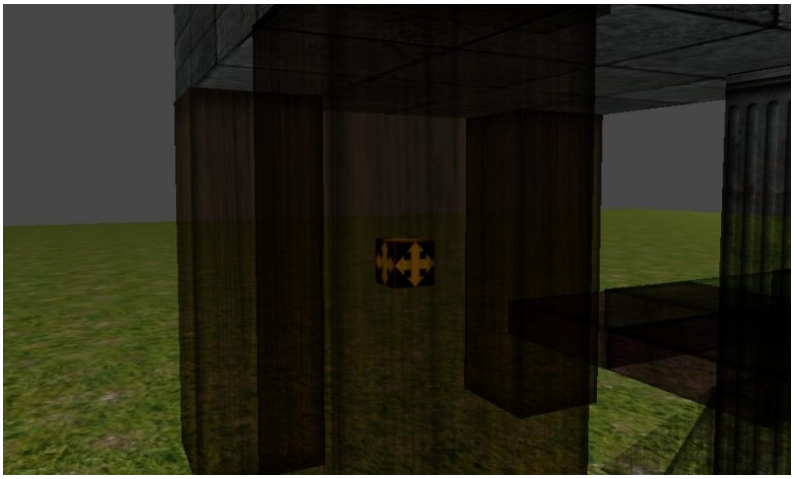

Figura 3.12: Efecto que produce el usar el casco de científico

Una vez vistos los trajes y cascos por separado procederemos a ver cómo funcionan las combinaciones especiales. En primer lugar veremos el traje de paracaidista con el casco de hacha (figura 3.13).

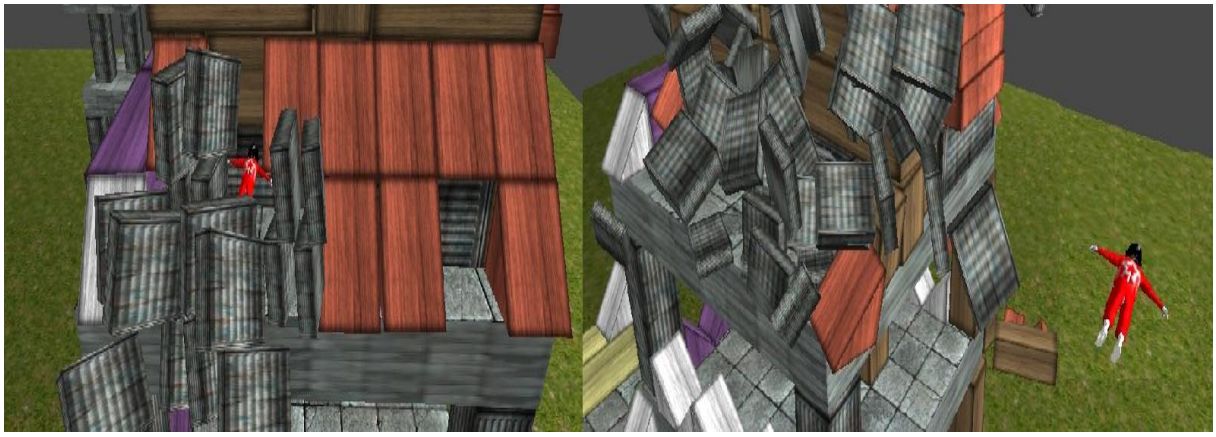

Figura 3.13: Combinación de traje paracaidista con casco de hacha

Como podemos observar mientras se realiza la caída se puede seguir controlando el personaje. Eso ayuda a poder destruir todo por donde se pasa.

La combinación de traje de hombre bala con casco hacha no requiere mucha validación, ya que realizará lo mismo pero realizando una penetración en el edificio a una velocidad aún mayor.

La última combinación que veremos será la de traje de antigravedad con casco explosivo (figura 3.14).

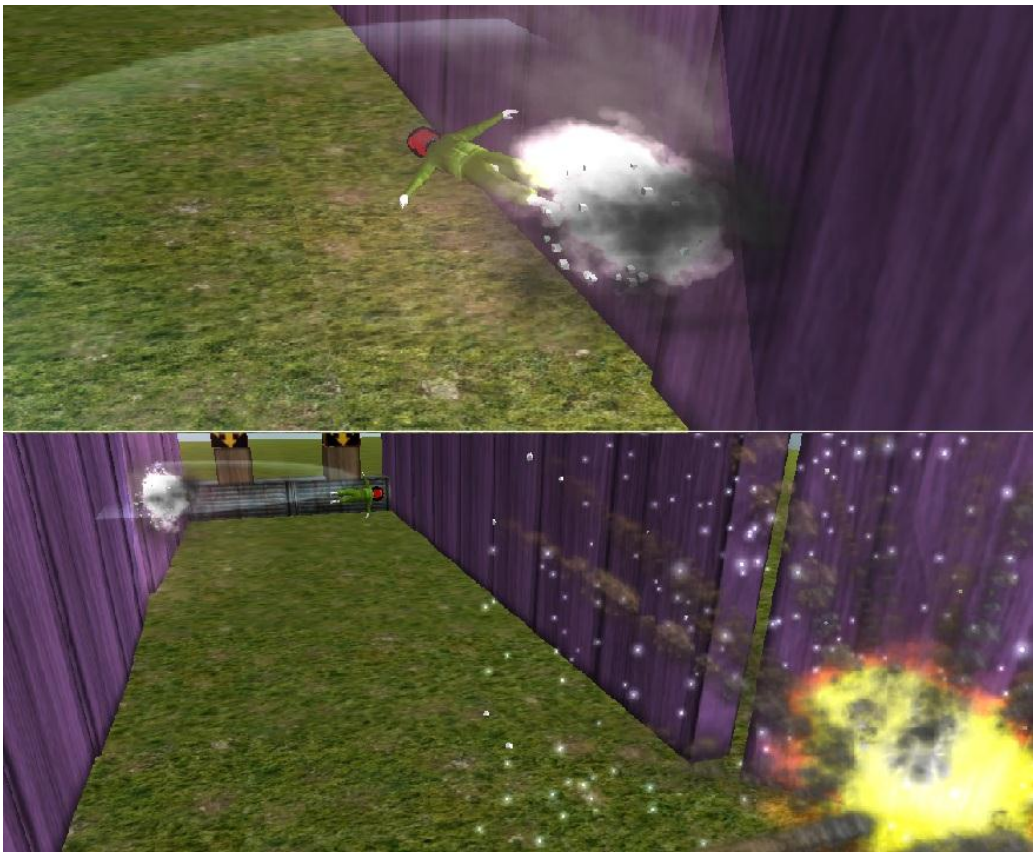

Figura 3.14: Combinación traje antigravedad con casco explosivo

En la figura 3.14 se puede observar como en primer lugar el jugador rebota, hace explotar una bomba, sigue su movimiento de rebote, choca contra otra pared y activa otra bomba más.

Una vez la lógica de juego detecta el choque empieza el estado de juego de finalización. En él se cuentan los puntos obtenidos en la ronda y se vuelve al estado de juego equipamiento para volver a empezar. Si se ha llegado al número máximo de lanzamientos se realiza la cuenta de puntos final y se muestra un mensaje de felicitación en caso de haber superado la puntuación mínima para el nivel o de castigo, en caso de no haber llegado (figura 3.15).

En caso de superar el nivel se muestran los puntos y se pasa a introducir el nombre del jugador para el ranking, una vez esto se muestra la posición de ranking que se ha conseguido realizando la ordenación de información que se tiene de puntos para ese nivel.

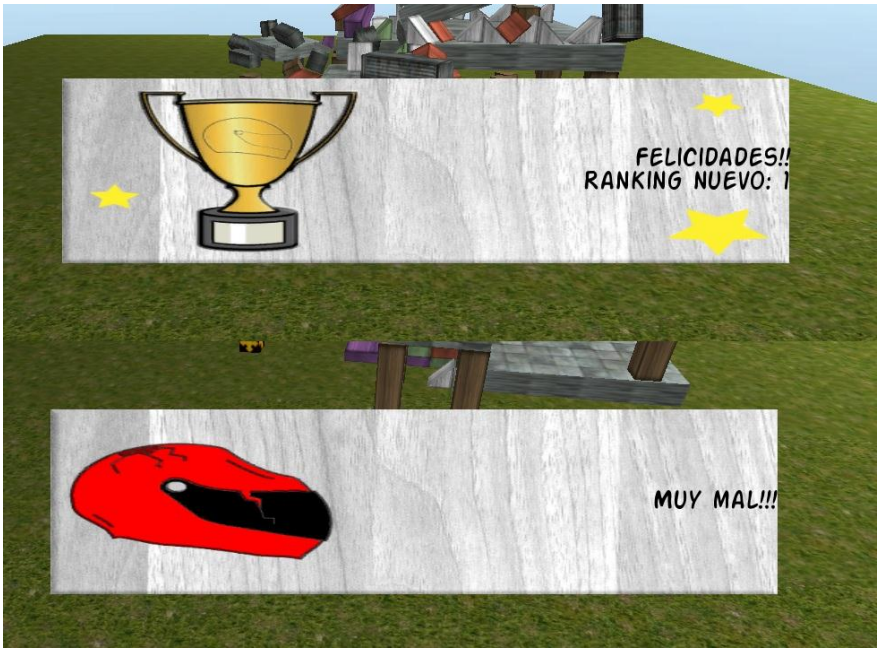

Figura 3.15: Mensaje de nivel superado con ranking (arriba), mensaje de nivel no superado (abajo)

3.2. Resultados de GUI

Para mostrar los resultados de los módulos de GUI se han realizado principalmente capturas de imagen. Ellas servirán para comparar el diseño conceptual inicial que se pudo observar en la parte de desarrollo con el diseño final de la aplicación. También se realizarán capturas para intentar mostrar cómo funcionan las animaciones que tienen los menús de aplicación y de

juego. También se han realizado pruebas de test ejecutando la aplicación muchas veces e intentando hacer que fallaran las animaciones, y menús.

La validación de esta parte se dividirá en dos puntos:

- Validación del menú de la aplicación
- Validación del menú de juego

Seguidamente detallaremos cada una de ellas.

3.2.1. Resultados del menú principal

Seguidamente se muestran capturas de pantalla para los diferentes menús que se han realizado.

En la figura 3.16 podemos observar la comparación entre el concepto inicial y el aspecto final de la pantalla de menú principal.

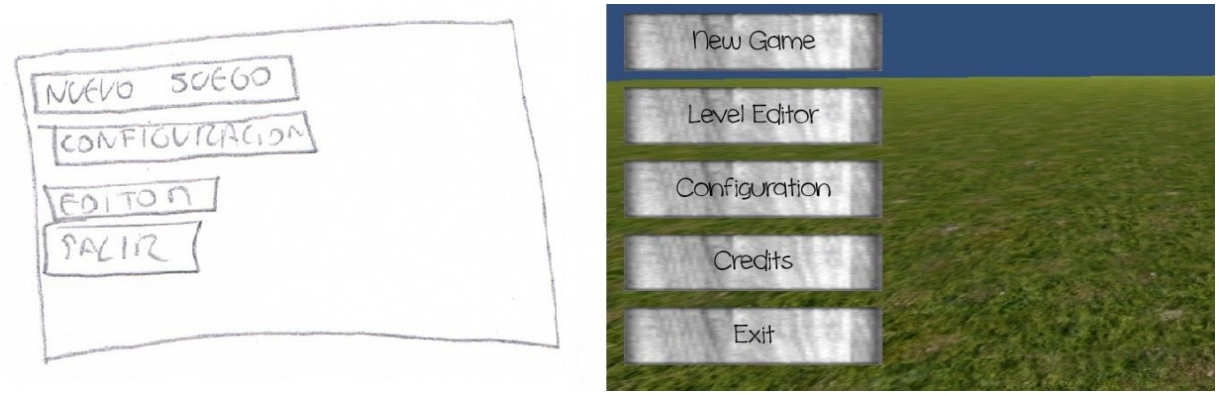

Figura 3.16: Comparativa entre concepto inicial (izquierda) y aspecto final (derecha) pantalla menú principal

Las animaciones implementadas para el menú de la figura 3.16, se pueden observar en la figura 3.17.

En ella podemos observar la transición de movimiento lateral para el menú, en este caso la transición va desde el menú principal al menú de “nueva partida”, se puede ver cómo

cuando se esconden los botones muestra el menú principal mientras que cuando aparecen se muestra el menú “nueva partida”.

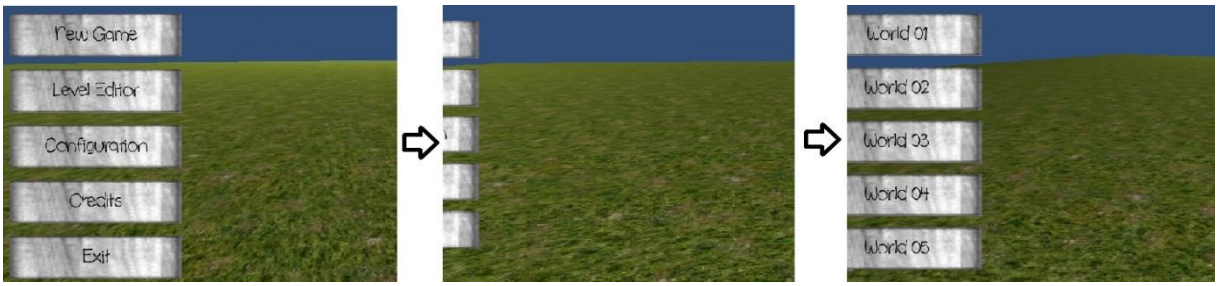

Figura 3.17: Ejemplo de animación del menú principal

En la figura 3.18 se puede observar el aspecto final de la pantalla de “nuevo juego” y la comparativa con el diseño conceptual.

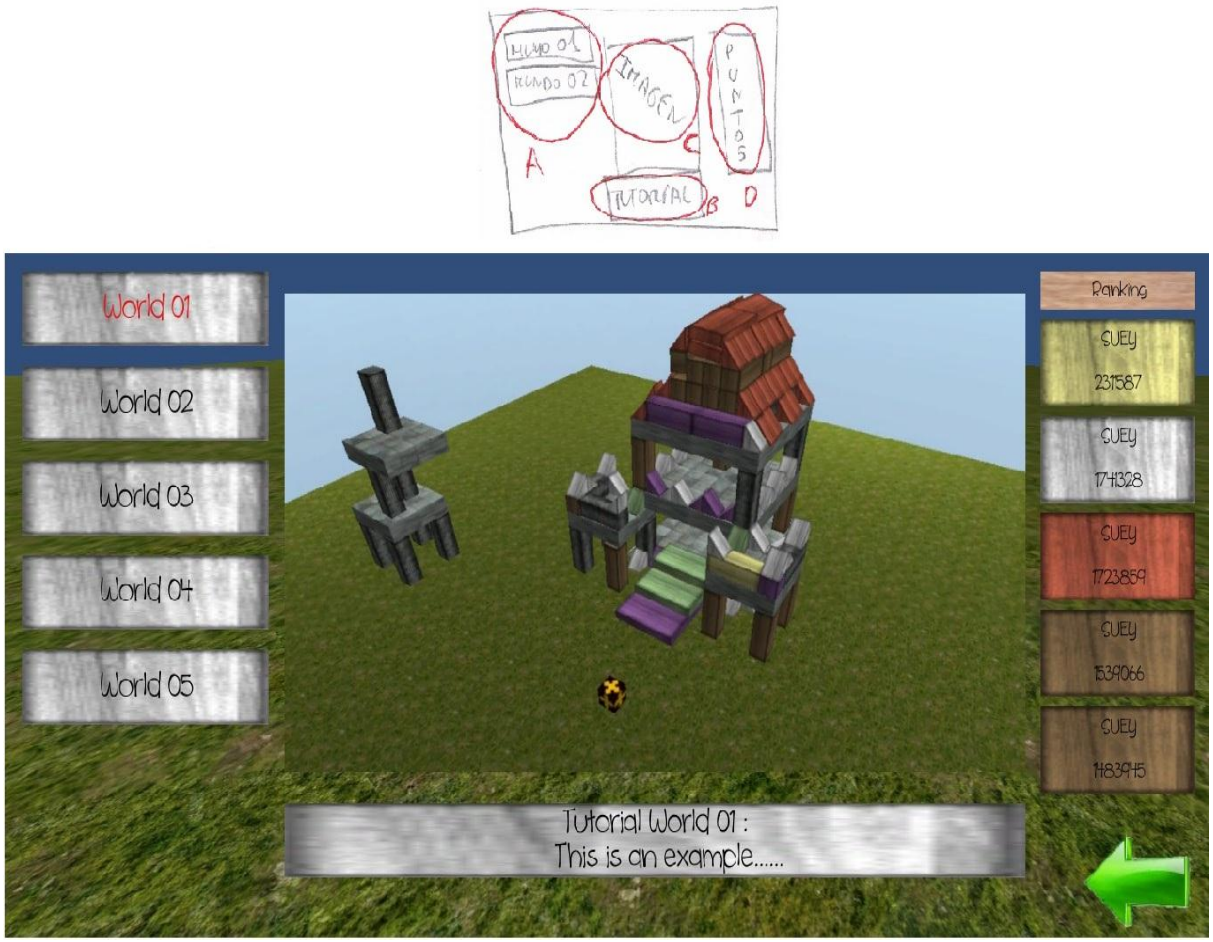

Figura 3.18: Comparativa entre concepto inicial (arriba) y aspecto final (abajo) pantalla de “nueva partida”

En la figura 3.18 se pueden observar cómo se han implementado todos los focos de interés (A, B, C y D) que se pueden ver en el concepto inicial de diseño. Tales cómo escenarios disponibles (A), tutorial de la pantalla que tiene el foco (B), previsualización del escenario (C) y ranking de puntuaciones para el escenario (D).

En la siguiente figura (3.19) podemos observar la animación que se ha implementado para el menú de “nueva partida”.

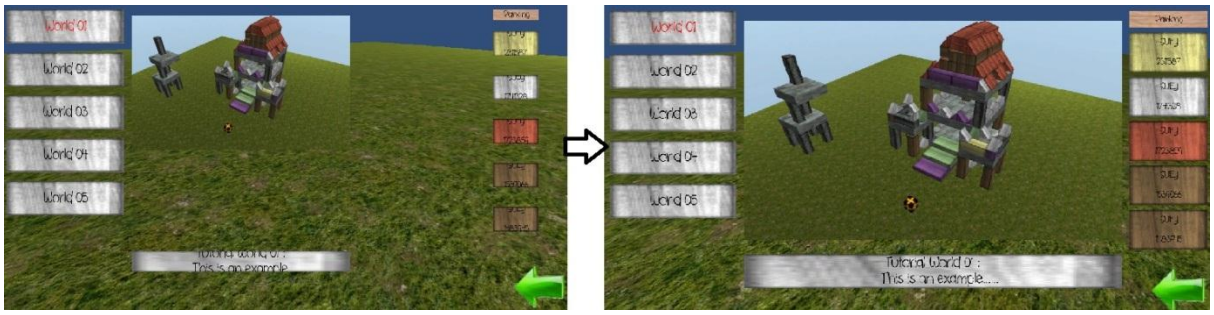

Figura 3.19: Animación de submenú “nueva partida”

Como se puede observar en la figura 3.19, en la figura de la izquierda el tamaño de los elementos en pantalla es más pequeño y gradualmente van incrementando su tamaño produciendo una animación bastante dinámica. La animación se producirá al situar el puntero del ratón sobre cualquiera de los escenarios disponibles para jugar, mostrando en los elementos que aparece la información correspondiente al seleccionado.

El siguiente submenú que veremos será el implementado para visualizar los niveles generados por el editor de niveles. En la figura 3.20 podemos ver la comparativa entre el concepto inicial y la implementación final.

Se puede observar (figura 3.20) como cada uno de los focos de atención que contenía el concepto inicial del menú de edición de niveles se han implementado en la versión final, tales como el botón para acceder al editor (A), los jugadores que han creado los escenarios (B), y una imagen de vista previa de los escenarios (C).

Después de ver las imágenes comparativas y las que muestran a grandes rasgos las animaciones de los menús se puede afirmar que se ha cumplido con el objetivo propuesto en cuanto a diseño del menú principal de la aplicación.

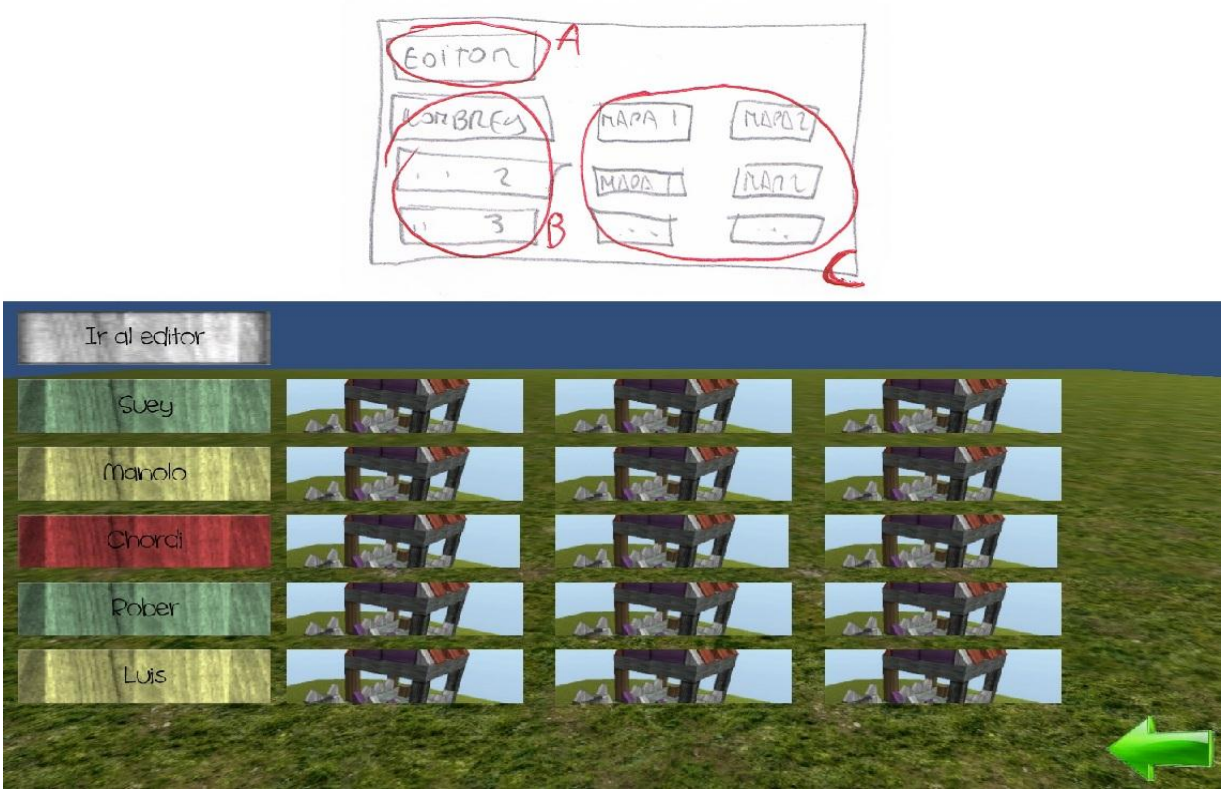

Figura 3.20: Comparativa entre concepto inicial (arriba) y aspecto final (abajo) pantalla de "editor de niveles"

3.2.2. Resultados del menú de juego

La validación de ésta parte se mostrará también con una imagen comparativa. En la figura 3.21 podemos observar el concepto inicial del menú de juego y la implementación final. Como se puede observar en la figura 3.21 se ha respetado el concepto inicial para el menú de juego. A la izquierda podemos encontrar los diferentes trajes que nuestro personaje podrá utilizar y a la derecha los cascos, de forma extra se ha añadido una caja abajo que informa sobre el funcionamiento del traje o casco sobre el que tenemos situado el puntero de ratón. Además cada vez que nos situemos sobre un traje o casco diferente podremos ver cómo el modelo del

personaje cambia, obteniendo una vista previa del diseño del modelo, tal como se puede observar en la figura 3.22.

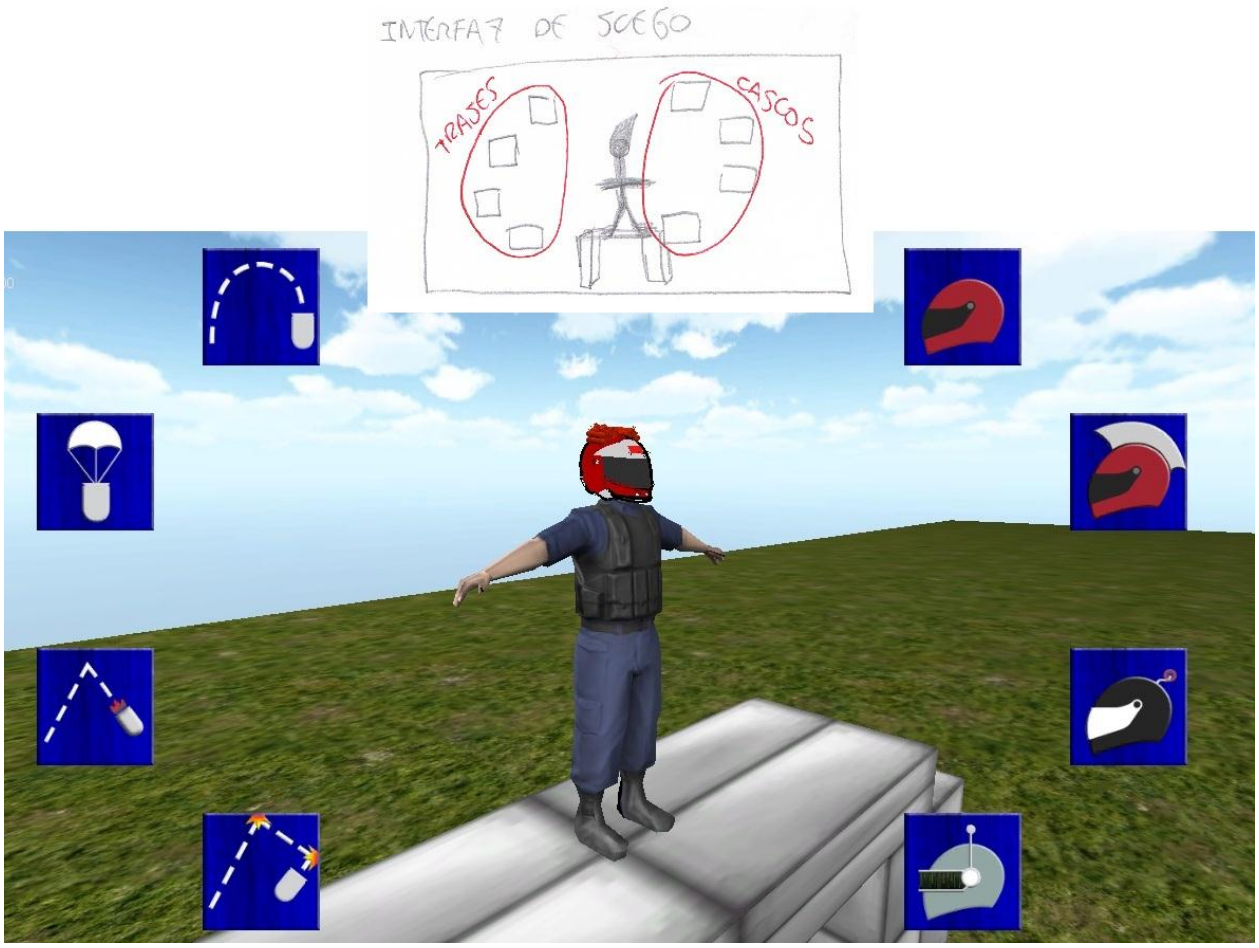

Figura 3.21: Comparativa concepto inicial (arriba) con diseño final (abajo) de la interfaz de juego

Figura 3.22: Diseño final de algunos de los modelos de personaje en el menú de selección

En desarrollo se comentó que cada uno de los niveles podían tener restricciones de trajes o cascos a utilizar, y la GUI de juego se crearía en función de esto. En la figura 3.23 y 3.24 se puede observar el funcionamiento correcto y la colocación correcta de botones cuando existen restricciones de este tipo.

Figura 3.23: Ejemplo de colocación de tres botones a la izquierda y dos a la derecha

Figura 3.24: Ejemplo de colocación de un botón a la izquierda y tres a la derecha

Después de ver la comparativa de imágenes y las vistas previas de los trajes se puede dar el menú de juego como validado.

3.3. Resultados de módulos de acceso a datos

Dentro de las pruebas de validación de los módulos de acceso a datos encontramos la validación para módulo gestor de archivos, el módulo de codificación y el módulo gestor de idiomas.

Seguidamente detallaremos cada uno de ellos.

3.3.1. Resultados de módulo gestor de archivos

El módulo de acceso a datos del proyecto se utiliza para crear dinámicamente la GUI por ejemplo o para leer información sobre las características de cada uno de los niveles como por ejemplo: los rankings de puntuaciones por nivel, los niveles jugables generados por el editor de niveles o la información propia de cada nivel como puntuación para superarlo, trajes disponibles y cascos.

Para comprobar la validez de ésta parte se han realizado muchas pruebas de validación, básicamente se han basado en la ejecución de la aplicación muchas veces. Ésta se ha probado de tal manera que el flujo de ejecución ha pasado varias veces por todas las condiciones posibles del programa, de esta forma se ha asegurado que funciona bien.

También se ha comprobado empíricamente que los datos que estaban en los archivos quedaban perfectamente reflejados en los menús a la hora de crear la interfaz de usuario en ejecución, todos y cada uno de los niveles de juego mostraban perfectamente sus características detalladas en los archivos de menú principal, niveles, opciones, menú de juego y editor de niveles.

3.3.2. Resultados del módulo de codificación

Siempre que el módulo gestor de archivos desee acceder a un archivo accederá a través del módulo de codificación. Cuando se requiera un archivo el módulo de codificación lo descodificará, el gestor de archivos utilizará la información que necesite de él y volverá a pedir al módulo de codificación que codifique el archivo. Eso provocará que el archivo siempre se encuentre protegido, ya que mientras el gestor lo esté utilizándolo el sistema operativo lo mantendrá bloqueado.

Para comprobar los resultados de la codificación se ha accedido a los archivos de texto que utiliza el programa mientras el programa no se está ejecutando, y cuando se está ejecutando. Se ha podido ver como no hay un solo instante de tiempo en el que se pueda ver el contenido de los archivos, provocando así que no se puedan modificar.

El aspecto de un archivo codificado es indescifrable, ello ayudará a proteger toda la información de valor para el juego de todo acceso indebido.

3.3.3. Resultados del módulo gestor de idiomas

Para validar el módulo gestor de idiomas se muestra una captura de pantalla de diferentes menús para ver su texto. Todos los textos que se colocan en el juego pasan a través del gestor de idiomas para garantizar que estén traducidos. Los archivos de idiomas se pueden seleccionar en tiempo de ejecución para poder cambiar de idioma en cualquier momento desde el menú de configuración.

Para comprobar que funciona bien ésta parte se ha ejecutado el juego con diferentes idiomas mediante el archivo de configuración inicial, posteriormente se ha navegado por todos y cada uno de los menús. También se ha modificado mientras se juega el idioma de juego, el cambio de idioma se produce de una forma dinámica y bastante rápida.

En la figura 3.25 se puede observar un ejemplo, en ésta podemos ver el menú principal con diferentes idiomas.

Figura 3.25: Menú principal en inglés (izquierda), castellano (centro) y catalán (derecha)

Además para añadir un idioma nuevo al juego tan solo es necesario crear su archivo de idioma, y poner el botón en la GUI para poder seleccionarlo.

3.4. Resultados de la integración

Seguidamente se muestran los resultados de la integración de este proyecto con los proyectos de “Launchageddon: Editor” [Jor] y “Launchageddon: Tecnología” [Enr].

En ésta parte se muestran los puntos más críticos de la integración, las partes que han sido más importantes y las que han tenido más problemas.

Para validar los resultados de la integración se ha ejecutado la aplicación varias veces. En cada una de ellas hemos repetido las acciones que se pueden realizar en el programa varias veces, contemplando todas las posibles variaciones en ramas de ejecución que se pudieran producir.

Ésta se ha dividido en dos apartados en primer lugar se detallarán los resultados de la integración de éste proyecto con “Launchageddon: Editor” y seguidamente la integración de éste proyecto con “Launchageddon: Tecnología”.

a) Gameplay – Editor:

Adaptar la función de carga de escenarios para el módulo de Gameplay:

Para poder cargar escenarios creados con el módulo de **Gameplay**, se ha tenido que hacer una adaptación del módulo de guardado y carga del módulo **Editor**, dejando solo el apartado de carga.

Para hacerlo posible además se ha tenido que añadir en este nuevo módulo una inicialización especial con la carga de los objetos 3D y las texturas, puesto que en **Editor** esto lo hace otro módulo por separado.

Éste nuevo módulo, que hemos nombrado **LevelLoader** es el núcleo de la integración entre **Gameplay** y **Editor**. Desde el módulo de **Gameplay** se puede acceder a los archivos de mapa generados en el **Editor**.

Añadir menú para visualizar niveles de editor:

En el módulo de **Gameplay** se tuvo que diseñar e implementar un menú para mostrar todos los niveles creados en el editor, y las personas que los habían creado.

Para ello se integró la estructura de creación de niveles dinámica en menús detallada en éste proyecto para la gestión de archivos. Desde **Gameplay** se facilitó que **Editor** pudiera modificar los archivos necesarios para mostrar la información en el menú, recibiendo los datos del escenario a la hora de grabar la información.

b) Gameplay -Tecnología

Problema con los modelos 3D:

Al importar los modelos 3D diseñados en el módulo de tecnología, nos dimos cuenta de que los escalados de los modelos no eran los mismos, Éste cambio se produce al editar el modelo original en formato (.fbx). Al abrir éste formato, el programa 3D Studio Max (utilizado para el diseño de los modelos) no guarda la escala original del modelo, así pues, los modelos editados no comparten la misma escala que el modelo original (Traje normal).

La solución al problema propuesta fue la de utilizar el modelo original y modificar en tiempo de ejecución la textura que tenían los diferentes modelos. Ello ha provocado que estos tengan todos el mismo volumen y tamaño, en el aspecto final del juego ya no se muestran las deformaciones que se pueden ver en la figura 3.26.

Por otro lado, al integrar los cascos del jugador no sufrimos el mismo problema. Al ser unos modelos que no distaban tanto del original se pudieron integrar fácilmente sin demasiados problemas. Con modificar sus escalas y rotaciones para que fueran iguales y sustituir el casco de cada uno de los modelos bastó.

Figura 3.26: Comparativa de diseño en módulo de tecnología (arriba), diseño final después de la integración (abajo).

Éste problema nos obligó a modificar la manera de cambiar los modelos en el selector de personajes, en el estado de juego de **equipamiento**.

Integración de control por Kinect:

En las reuniones realizadas durante el proyecto se definieron todos los accesos a los controles del módulo de Tecnología, de modo que el resto de módulos pudieran empezar a pensar cómo acceder y utilizar los datos proporcionados.

Cuando se implementó del módulo de Tecnología la obtención del control por Kinect se decidió realizar en primer lugar un módulo que simulase el comportamiento de éste usando el teclado. Éste módulo fue especialmente útil para poder empezar a familiarizarse con el tipo de datos con los que trabajaría el módulo final, y poder ir haciendo pruebas. Gracias a éste módulo se pudo avanzar en este aspecto sin la necesidad de tener el dispositivo.

En el caso del módulo de reconocimiento de eventos, también se pactaron los estados correspondientes a cada evento, facilitando la posterior implementación final.

El resto de módulos como el puntero controlado por Kinect, no necesita integración ya que no proporciona datos a otro módulo, sino que directamente interactúa con el juego. De este modo sólo se ha debido modificar la puesta en marcha y el apagado de éste para que el resto de módulos puedan utilizarlo cuando les sea necesario.

Integración de la cámara en primera persona:

La integración de la cámara en primera persona que se realizó en el módulo de Tecnología fue bastante sencilla. En un primer lugar se analizaron las dependencias de código necesarias, luego se llevó todo el código y objetos de Unity asociados a Gameplay.

Para cambiar de cámara tan solo fue necesario activar y desactivar la que se quería en ese momento en tiempo de ejecución.

3.5. Validación de usuarios

En el anexo número VIII se puede encontrar la encuesta que se ha realizado a los usuarios que han probado Launchageddon. La encuesta se ha realizado a siete personas, seguidamente encontramos el análisis que se ha realizado:

- La distribución en pantalla de elementos ha gustado, pero ha habido alguna queja sobre decorados del videojuego y colorido. Comentaron que habría gustado más un aspecto más tipo de cómic y "divertido" para los escenarios.

- La experiencia de juego con teclado y ratón ha sido satisfactoria, no es nada difícil controlar el juego.

- El sistema de puntuación del juego es el punto que han castigado más, no era lo suficientemente claro. Faltaría algún tipo de tutorial dentro del juego, aunque el contador de puntos en pantalla era acertado.

- Que el juego esté en tres idiomas y se puedan añadir más es una característica que ha gustado, casi toda la gente que lo ha probado ha afirmado que ayudaría en un futuro si sale a la venta el juego o se muestra a más público.

- En general para la música y los efectos de sonido la valoración ha sido normal. Ello se debe a que tampoco hay demasiados de ellos en el juego, todos son utilizados de librerías de sonido gratuitas por tanto no han sido hechos a medida para el juego.

- En general la física implementada para el juego ha gustado. Los usuarios han experimentado entre todas las posibilidades que brindan los diferentes trajes y han visto que respondían como estaba previsto. Muchos de ellos han afirmado que la respuesta a los choques parecía bastante natural, a excepción de alguna vez si se producían choques contra muchos objetos a la vez.

- Por lo general las mecánicas de juego han gustado, no obstante cabe destacar que alguna persona comentó que se podrían haber hecho más combinaciones especiales de trajes y cascos.

- En cuanto a si el juego sería fácil de jugar en otra plataforma (móvil y/o Tablet) las respuestas son bastante dispersas. Algunos opinan que sería interesante ya que el juego es bastante rápido de jugar y sencillo, otros opinan que no se podría por el tamaño de la pantalla en comparación con el tamaño de los objetos que se ven en el juego, afirman que no se podría ver demasiado.

- En cuanto a si es un juego innovador la respuesta más extendida ha sido afirmativa. Hasta ahora no existen muchos juegos de este tipo sobretodo en tres dimensiones.

- Casi todos los usuarios opinan que es un juego que se puede jugar con amigos y/o familia, debido al carácter competitivo que tiene por los rankings. No obstante algunos de ellos afirmaban que antes de poder responder se debería mejorar el sistema de puntuaciones.

- A todo el mundo le ha parecido un juego fácil de jugar, bastante sencillo y divertido para jugar un rato y competir con amigos.

El resultado de las encuestas la verdad que ha sido bastante satisfactorio, a los usuarios les ha gustado el juego y les gustaría que se siguiera trabajando en él para poder mostrar un producto más sólido en un futuro.

3.6. Futuro del proyecto

Después de ver el resultado del proyecto y lo bien que ha funcionado el trabajo en equipo, se podría valorar la idea de seguir trabajando en el proyecto para llevarlo a plataformas de distribución digital tipo Steam [Ste] para intentar comercializarlo.

En caso de no realizarse por motivos laborales o académicos se tendría en cuenta el equipo para el desarrollo de otros proyectos en el futuro o incluso la creación de una empresa/equipo de desarrollo.

En el siguiente capítulo encontraremos las conclusiones del proyecto y las posibles mejoras.

4. Conclusiones

4.1. A nivel global

- Se ha desarrollado un videojuego 3D llamado Launchageddon.
- Se ha diseñado el concepto de juego inicial para Launchageddon. Documento que ha servido para diseñar la implementación, dividir las tareas entre el equipo y establecer los módulos y sus comunicaciones.
- Se ha realizado el proyecto entre tres personas, con tareas bien diferenciadas fruto de una repartición equitativa.
- Se ha llevado a cabo la integración de los tres módulos que conforman el proyecto de forma satisfactoria, se han solucionado los problemas provenientes de la integración.

4.2. A nivel del propio módulo de Gameplay

- Se han investigado acerca del uso del motor gráfico Unity, se ha aprendido a utilizar el entorno y sus componentes más importantes.
- Se ha diseñado el diagrama de módulos del módulo de gameplay y sus comunicaciones.
- Se ha estudiado la física necesaria para el jugador de Launchageddon. Se ha valorado la utilización de diversas técnicas en cinemática, dinámica y choques que posteriormente se han implementado y validado. Y se ha aprendido a utilizar la tecnología de detección de colisiones que ofrece Unity.
- Se ha investigado acerca de cámaras en tercera persona para juegos y se han implementado varias.
- Se ha diseñado el documento de concepto de juego para Launchageddon que detalla todas las mecánicas de juego, sistemas de puntuaciones y rankings.

- Se han diseñado las diversas interfaces de juego teniendo en cuenta la plataforma de ejecución y los métodos de comunicación entre usuario – programa.

- Se han diseñado varios niveles para el juego, cada uno de ellos muestra una faceta diferente en cuanto a estrategia a la hora de jugar.

- Se ha implementado un sistema de gestión de archivos para externalizar la creación de contenido en el juego. La información en archivos se ha codificado y descodificado utilizando un buen algoritmo. Se ha diseñado la estructura de juego para facilitar su localización a varios idiomas a través de archivos de idiomas externos.

- Se ha investigado acerca de colocación de efectos en pantalla de sonido y visuales y se ha aprendido a utilizarlos dentro de Unity.

- Se han realizado pruebas de test para validar la aplicación, se ha mostrado a personas ajenas al proyecto que la han evaluado con un resultado altamente satisfactorio.

Como incidencias cabe destacar:

- El entorno de desarrollo utilizado, Unity, funciona bastante bien, se volvería a usar en otro proyecto.

- El depurador de código de Unity no se integra demasiado bien con el editor de proyectos utilizado (MonoDevelop), ha sido difícil la tarea de depurar el correcto transcurso de la ejecución del programa.

Como mejoras cabe destacar:

- El diseño de un buen sistema de puntuaciones para un videojuego es una tarea complicada, así como el equilibrado de la dificultad. Este aspecto se podría mejorar si se hubiera tenido más tiempo.

- En cuanto al aspecto visual del juego se ha hecho lo que ha estado en nuestra mano, entre todo el equipo se han diseñado los modelos 3D, texturas e iconos.

- Habría estado interesante llevar los rankings de puntuaciones a un lugar en la red, para poder realizar clasificaciones de jugadores por puntuación entre amigos por red.

- Diseñar un juego que sea divertido es una tarea muy difícil, creemos que se ha cumplido con el objetivo pero siempre se puede refinar el sistema, pensar en nuevas mecánicas e implementarlas.

- Se podría valorar la inclusión de juego en línea, a nivel de destrucción de escenarios o a nivel de edición. No obstante sería una parte difícil de pensar, ya que el diseño principal del juego es para un solo jugador.

Bibliografía:

[Ant] : http://en.wikipedia.org/wiki/Antoine_Lavoisier

Información acerca de Antoine Lavoisier. Fecha último acceso: Agosto/2012.

[Ata]: http://www.atari.com/

Página web principal de la compañía Atari. Fecha último acceso: Agosto/2012.

[Bin]: http://www.radgametools.com/bnkmain.htm

Página web principal de Bink Video, motor gráfico para videos. Fecha último acceso: Agosto/2012.

[Ccd]: http://http.developer.nvidia.com/GPUGems3/gpugems3_ch33.html

Información de Nvidia sobre algoritmos de detección de colisiones. Fecha último acceso: Agosto/2012.

[Det]:http://unity3d.com/support/resources/unity-extensions/explosion-framework
Framework de explosiones que proporciona Unity. Fecha último acceso: Septiembre/2012.

[Enr]: Enric Martinez Ibarra, "Launchageddon: Tecnología", Proyecto de Ingeniería Informática, Escuela de Ingeniería, UAB, 2012,.

[Esc]: http://gameware.autodesk.com/scaleform

Librería para creación de interfaces de usuario. Fecha último acceso: Agosto/2012.

[Eup]: http://www.naturalmotion.com/technology.htm

Librería para animación de personajes. Fecha último acceso: Agosto/2012

[Gam]: Mike McShaffry, "Game Coding Complete", Charles River Media, 2009.

[Hav]: http://www.havok.com/

Librería de física. Fecha último acceso: Agosto/2012.

[Jor]: Jordi Cartes Rosell, “Launchageddon: Editor”, Proyecto de Ingeniería Informática, Escuela de Ingeniería, UAB, 2012.

[Mil]: http://www.radgametools.com/miles.htm

Librería de sonido. Fecha último acceso: Agosto/2012.

[New] : http://es.wikipedia.org/wiki/Leyes_de_Newton

Información acerca de las leyes de Newton. Fecha último acceso: Agosto/2012.

[Nin]: http://www.nintendo.com/

Página web principal de Nintendo. Fecha último acceso: Agosto/2012.

[Phy]: http://www.nvidia.es/object/physx_new_es.html

Página web oficial de Nvidia. Fecha último acceso: Agosto/2012.

[Phy2]: Ian Millington, “Game Physic Engine Development”, Morgan Kaufmann, 2007.

[Phy3]: David M.Bourg, “Physics For Game Developers”, O’Reilly, 2002.

[Phy4]: Christer Ericson, “Real Time Collision Detection”, Elsevier, 2005.

[Ral]: http://en.wikipedia.org/wiki/Ralph_H._Baer

Información acerca de Ralph H.Baer. Fecha último acceso: Agosto/2012.

[Rjn1]: http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf

Información acerca del algoritmo Rijndael. Fecha último acceso: Agosto/2012.

[Rjn2]: http://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndael.aspx
Información de Microsoft sobre el algoritmo Rijndael. Fecha último acceso: Agosto/2012.

[Rig]: http://docs.unity3d.com/Documentation/ScriptReference/Rigidbody.html .

Aquí podemos encontrar la guía de referencia para la interfaz de Unity que facilita la detección de colisiones. Fecha último acceso: Agosto/2012

[Rov]: http://www.rovio.com/

Página web oficial de Rovio Mobile. Fecha último acceso: Agosto/2012.

[Seg]: http://www.sega.com/

Página web oficial de Sega. Fecha último acceso: Agosto/2012.

[Sin]: http://en.wikipedia.org/wiki/Singleton_pattern

Patrón de diseño singleton. Fecha último acceso: Agosto/2012.

[Sou]: http://www.free-sound-clips.com/

Biblioteca online de sonidos gratuitos. Fecha último acceso: Septiembre/2012.

[Ste]: http://store.steampowered.com/

Página web del servicio de descarga de juegos Steam. Fecha último acceso: Septiembre/2012.

[Tem]: http://en.w.ikipedia.org/wiki/Template_method_pattern

Patrón de diseño template method. Fecha último acceso: Agosto/2012

[Tut]: http://unity3d.com/support/resources/tutorials/

En la página podemos encontrar algunos tutoriales oficiales de Unity. Fecha último acceso: Agosto/2012.

[Uni]: http://unity3d.com/

Página web oficial de Unity Technologies. Fecha último acceso: Agosto/2012.

[Unr]: http://www.unrealengine.com/

Página web oficial de Unreal Engine. Fecha último acceso: Agosto/2012.

[Ure]: http://unity3d.com/support/documentation/

Aquí tenemos la referencia de documentación para Unity. Fecha ultimo acceso: Agosto/2012.

Anexo I: Motores Gráficos

Definición:

Motor: aquella máquina destinada a producir movimiento utilizando una fuente de energía.

Gráfico: representación de algo mediante un dibujo, ya sea a mano o por computador.

Hasta aquí la definición de diccionario para ambas palabras. Si hacemos una analogía, como si un motor de coche se tratara, podemos pensar que el motor gráfico es algo que hay debajo del capó del coche (detrás de la aplicación/videojuego) y que permite mover el automóvil (los gráficos).

Ese "motor" utiliza como energía código del lenguaje de programación para el que haya sido diseñado, script propio, o incluso instrucciones de bajo nivel, formando toda una serie de rutinas, subrutinas, procedimientos y funciones que permiten trabajar de una forma más fácil a los programadores. El motor proporciona primitivas que abstraen todo tipo de funcionalidades y ocultan sus detalles al programador.

Para el caso de los desarrolladores de videojuegos, esto permite ahorrar tiempo en la creación del videojuego. Nadie pensaría en hacer un videojuego si a la hora de, por ejemplo, que nuestro personaje disparara una bala (si el juego fuera de acción) el programador tuviera que generar para cada instante la ecuación de la recta que seguiría, aplicar coeficientes de rozamiento con el aire, crear la física de caída libre de la bala, etc. O si de un juego de coches se tratara que a cada momento se actualizara la luz que afecta al escenario, a las sombras, a los brillos de la carrocería, o que hiciera los cálculos de velocidad, aceleración, fricción, rozamiento al girar una curva.

Los motores gráficos permiten centrar el desarrollo en el diseño en sí y en su implementación, facilita el uso de muchas herramientas ya creadas, que de otra forma, si se empezará de cero en el desarrollo, sería un proceso mucho más complicado (ya lo es de por sí) y largo.

Toda compañía que desarrolle videojuegos hoy en día utiliza como mínimo un motor gráfico[1], ya sea creándolo o mejorando de una versión anterior, comprando la licencia para utilizar uno ya creado o usando uno libre y gratuito. Así cada compañía puede escoger el motor que más se adecúa a las necesidades que tiene.

Todas estas necesidades son las que marcan la diferencia a la hora de escoger uno, ya que además de permitir generar gráficos (renderizar), moverlos, realizar varias operaciones con texturas y modelos 3D o 2D, los motores gráficos son capaces de incluir otras funcionalidades, como, por ejemplo, facilitar el uso de dispositivos de entrada/salida, ratón, teclado, sonido, cd-rom, video, redes, o el uso de ecuaciones matemáticas, algoritmos complejos, estadística, física, vectores, etc. En definitiva, que la elección de motor gráfico estará marcada en base a lo que ofrece, y en su caso su relación calidad/precio.

Dado el gran número de funcionalidades que pueden aportar los motores gráficos, es de suponer que éstos han sido diseñados siguiendo un proceso de ingeniería de software. Por tanto podemos hablar de una jerarquía de módulos y de una organización en estos. Seguidamente se hablará de esta organización, ella mostrará una clasificación de motores gráficos según sus características más diferenciables. Más adelante también se podrá ver la estructura modular que suelen implementar cada uno de ellos.

a) Organización:

Según el motor gráfico con el que nos encontremos, éste habrá sido creado en torno a un paradigma de programación diferente, o dependiendo del lenguaje que utilice la forma de usarlo será diferente.

Nos encontramos por ejemplo con motores gráficos, que para poder funcionar utilizan un lenguaje de programación padre. Un ejemplo de ellos son los situados en la tabla 1.

[1] Esto se puede extraer directamente de cómo está organizado, más adelante se hablará de ello.

Nombre	Lenguaje	Logotipo
OpenGL	C++	
Pygame	Python	
SDL	C/C++	

Tabla I.1: Motores gráficos basados en un lenguaje de programación existente

Todos los motores gráficos que podemos observar en la tabla 1 se utilizan de la forma en que se añade una librería a un proyecto, importando los archivos para usarlos donde se requieran.

Por otro lado tenemos motores que tienen un lenguaje propio, él no se ha especificado ya que no se ha encontrado información al respecto, podemos observar unos cuantos de ellos en la tabla número 2.

Nombre	Logotipo
UnrealEngine	
Id Tech 5	
Cry Engine 3	

Tabla 2: Motores gráficos que utilizan un lenguaje de programación propio

Los motores gráficos que podemos observar en la tabla 2 utilizan su propio lenguaje de scripting, por lo tanto su propio compilador, intérprete y en su caso editor. Y a pesar de utilizar un script propio, la mayoría suelen estar basados en lenguajes de alto nivel como C++.

b) Estructura modular

En su mayoría, todo motor gráfico está compuesto de diversos módulos o subsistemas, cada uno de ellos engloba funcionalidades que tienen un objetivo común. Seguidamente explicaremos en detalle las principales funciones que realizan cada uno de los módulos.

- **Módulo de video:** es el encargado de renderizar las imágenes en 2D o 3D. Es capaz de construir líneas, píxeles, construir polígonos, también de manejar los modos de video, configurar viewport, gestionar el acceso a la memoria de video, etc.

- **Módulo de física:** es el encargado de gestionar la física de nuestro mundo gráfico, gestionar las colisiones y gestionar todos los movimientos de los elementos que interfieren en el mismo.

- **Módulo de sonido:** gestiona la música y todos los efectos de sonido que tengamos: el volumen, ecualización y balance.

- **Módulo de eventos**: controla todos los eventos de entrada al sistema, teclado, ratón, pantalla, joystick, gestiona una cola de eventos con información de toda la entrada/salida ejecutada, para procesarla o ignorarla en su caso.

- **Módulo de red:** facilita las opciones de red (en caso de que se necesiten) en nuestro programa/juego. Facilita las tareas de juego online, habilita la conexión entre computadores.

- **Módulo de timers:** en caso de los videojuegos el tiempo es importante, no por tener una máquina más potente desearemos que todo el juego vaya más rápido[2]. Este módulo se encarga precisamente de eso, de actualizar todas las acciones de la aplicación en consecuencia del tiempo que tarda entre cada frame, es útil para evitar

[2] A todo el mundo le ha pasado que intenta jugar a un juego viejo con un ordenador de ahora, y no se puede jugar de lo rápido que va.

que lo que vemos por pantalla vaya al tiempo fijado por la CPU, si no que se muestre de una forma más natural.

- **Módulo IA:** es capaz de controlar la inteligencia artificial del videojuego, en caso de colocar oponentes, o personajes no controlados por el jugador.

Como se ha dicho antes, cada compañía utiliza un motor gráfico, pero gracias a que cada uno de ellos puede ser dividido en módulos, cada compañía puede escoger utilizar módulos de otro para conformar su entorno de desarrollo. Es decir una compañía puede decidir usar el módulo de video de un motor gráfico X y el módulo de física de un motor Y, y conformar de esta forma su propio puzle de motor gráfico.

Éste hecho ha provocado que el mercado de los motores gráficos se haya diseminado aún más, creando empresas que se dedican a trabajar solo sobre un módulo en concreto y venderlo. Ejemplo de ello podemos encontrar Havok [Hav] o Euphoria [Eup] para la física, Miles Sound System [Mil] para el sonido, Bink [Bin] o Scaleform [Sca] para el video.

Anexo II: Cinemática

La cinemática es la rama de la mecánica clásica que estudia los fenómenos de movimiento de objetos y partículas, basándose únicamente en velocidad, posición, aceleración y trayectoria, sin tener en cuenta las causas que provocan estos sucesos.

Básicamente detalla cómo se va a mover un objeto dada una ecuación. Esta ecuación responderá a cómo expresar la variación de coordenadas de posición de un objeto en función del tiempo de simulación, utilizando los parámetros básicos mencionados.

Seguidamente iremos desgranando cada uno de los diferentes movimientos que hemos empleado para el videojuego, y sus connotaciones para la simulación en 3D. Los tipos de movimientos implementados son movimiento rectilíneo uniforme, movimiento rectilíneo uniformemente acelerado y movimiento parabólico.

a) **Movimiento rectilíneo uniforme**

Éste tipo de movimiento da el nombre a aquel donde no existe aceleración y la velocidad de desplazamiento es constante.

El sistema va regido por las ecuaciones (II.1):

$$x(t) = x_0 + v_x * t$$

$$y(t) = y_0 + v_y * t \qquad \text{(II.1)}$$

$$z(t) = z_0 + v_z * t$$

$D\acute{o}nde: x(t), y(t), z(t) es\ la\ posición\ respecto\ al\ tiempo\ de\ cada\ coordenada,$

$(x_0, y_0, z_0)\ las\ posiciones\ iniciales\ de\ cada\ coordenada,$

$(v_x, v_y, v_z) las\ velocidades\ de\ cada\ coordenada,$

$t\ el\ tiempo\ de\ simulación.$

La figura II.1 muestra una gráfica característica de la evolución posición/tiempo para cada una de las componentes de un objeto que se mueve regido por este sistema, la pendiente de la recta que forma dependerá de la velocidad a la que se mueva el móvil, se puede observar como al ser la velocidad constante la posición cambia de una forma gradual.

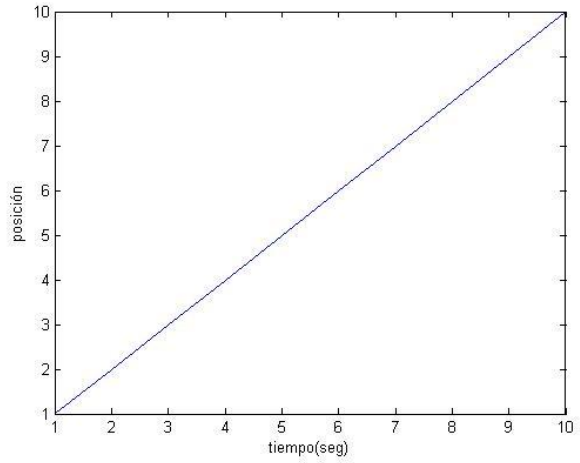

Figura II.1: Gráfica posición/tiempo característica de un movimiento rectilíneo uniforme

b) **Movimiento rectilíneo uniformemente acelerado**

Si nos encontramos ante un movimiento donde la trayectoria del móvil es una recta y la aceleración es constante, nos encontraremos ante este tipo de movimiento. La aceleración provocara un cambio gradual en la velocidad a la que se mueve el móvil. El sistema viene definido por las ecuaciones (II.2):

$$\begin{aligned} x(t) &= x_0 + v_{x0} * t + 1/2 * a_x * t^2 \\ y(t) &= y_0 + v_{y0} * t + 1/2 * a_y * t^2 \\ z(t) &= z_0 + v_{z0} * t + 1/2 * a_z * t^2 \end{aligned} \qquad \text{(II.2)}$$

$D\acute{o}nde: (v_{x_0}, v_{y_0}, v_{z_0})$ *son las velocidades iniciales de cada coordenada,* (a_x, a_y, a_z)*las aceleraciones de cada coordenada.*

Para el modelo también podemos calcular las velocidades de cada coordenada y en cualquier instante de tiempo, para ello utilizaremos las ecuaciones (II.3):

$$\begin{aligned} v_y(t) &= v_{y0} + a_y * t \\ v_x(t) &= v_{x0} + a_x * t \\ v_z(t) &= v_{z0} + a_z * t \end{aligned} \qquad \text{(II.3)}$$

Las gráficas características de posición/tiempo y velocidad/tiempo de cada coordenada cuando el móvil se mueve a una aceleración constante y positiva son las que podemos observar en las figuras II.2 y II.3. Podemos observar esta vez cómo la posición respecto al tiempo cambia de una forma exponencial, mientras que la velocidad incrementa linealmente.

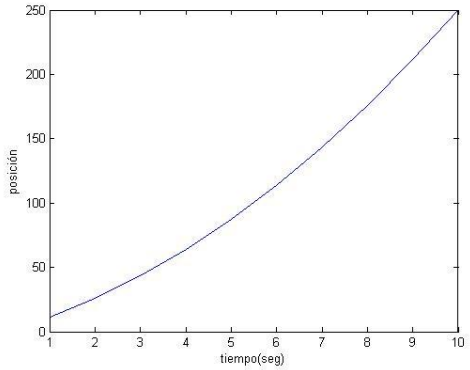

Figura II.2: Gráfica posición/tiempo característica de un movimiento rectilíneo uniformemente acelerado

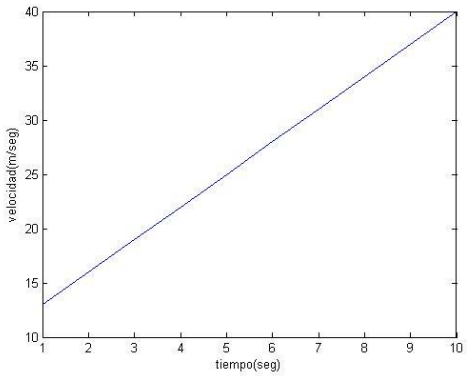

Figura II.3: Gráfica velocidad/tiempo característica de un movimiento rectilíneo uniformemente acelerado

c) **Movimiento parabólico**

Cuando en el movimiento que estudiamos la componente de aceleración actúa en el mismo plano que la velocidad y la trayectoria, tendremos un movimiento parabólico.

Éste puede dividirse en dos componentes: la componente de la velocidad que actúa en la dirección de la aceleración será un movimiento rectilíneo uniformemente acelerado, las componentes perpendiculares serán movimientos rectilíneos uniformes.

El esquema que tenemos es el que se puede observar en la figura II.4:

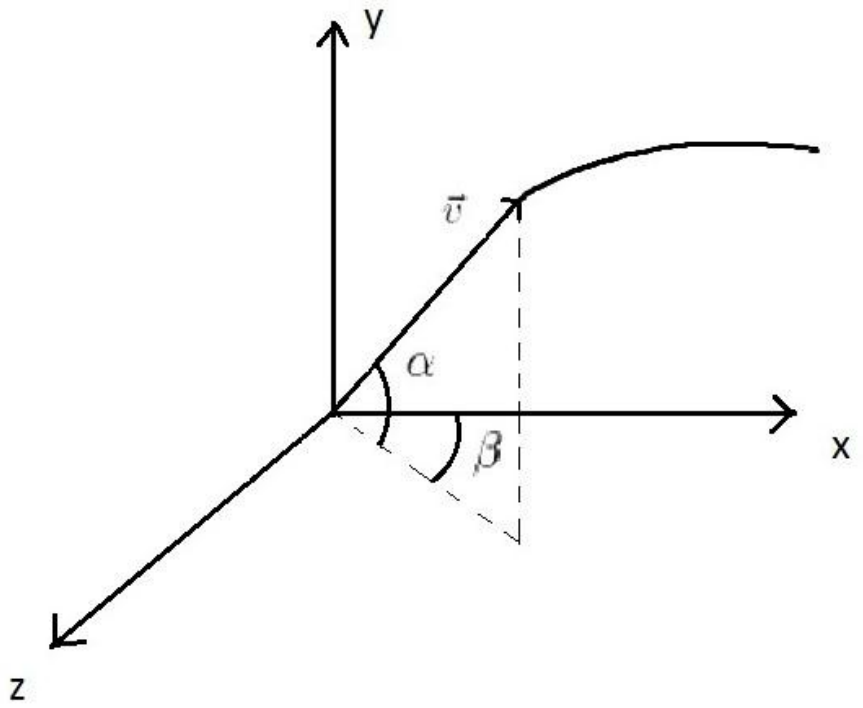

Figura II.4: Esquema del movimiento parabólico en tres dimensiones

Como podemos observar en la figura II.4, el movimiento parabólico 3D tiene en cuenta dos ángulos para el lanzamiento. De ellos dependen las componentes que tendrá la velocidad inicial al desglosarse. Mientras que la componente vertical de la velocidad se verá afectada por la gravedad, las otras dos se mantendrán constantes durante todo el movimiento, provocando que la trayectoria siga una parábola.

El sistema queda descrito en las ecuaciones (II.4), (II.5) y (II.6):

- Para los movimientos horizontales:

$$x(t) = x_0 + v_0 * cos(\alpha) * sin(\beta) * t$$
$$v_x = v_0 * cos(\alpha) * sin(\beta) \quad \text{(II.4)}$$

$$z(t) = z_0 + v_0 * cos(\alpha) * cos(\beta) * t$$
$$v_z = v_0 * cos(\alpha) * cos(\beta) \quad \text{(II.5)}$$

- Para el movimiento vertical:

$$y(t) = y_0 + v_0 * sin(\alpha) * t - 1/2 * g * t^2$$
$$v_y(t) = v_0 * sin(\alpha) - g * t \quad \text{(II.6)}$$

Anexo III: Dinámica

La dinámica hace referencia a cómo se describen los factores capaces de alterar el estado de un sistema físico. Intenta ir un paso más allá que la cinemática e intenta dar explicaciones a los fenómenos que produce, traducidos en las ecuaciones que hemos visto en el anexo número II. Su principal objetivo es dar sentido a las características sobre cómo evoluciona un sistema entre dos estados de tiempo.

Para poder entender algunos de los temas que vendrán más adelante y que hemos abarcado en el apartado de física es necesario conocer algún que otro teorema sobre dinámica. Muchos de ellos están directamente relacionados con la frase: "La energía ni se crea ni se destruye, se transforma" de Antoine-Laurent de Lavoisier [Ant].

Seguidamente detallaremos los teoremas que se han estudiado, entre ellos encontramos el teorema de la conservación de cantidad de movimiento, el teorema de conservación de energía basado en la tercera ley de Newton y algún que otro detalle como el coeficiente de restitución que mide el grado de conservación de energía para choques.

a) Conservación cantidad de movimiento

El principio de la conservación de cantidad de movimiento es una de muchas leyes importantes de la física que expresa que una cierta magnitud física se mantiene constante –se conserva- cuando se cumplen determinadas condiciones.

La ley se deduce de una forma un tanto sencilla a partir de la segunda ley de Newton. Se expresa en función de la cantidad de movimiento y se puede resumir en la siguiente frase: "Cuando la fuerza absoluta aplicada a un cuerpo es nula, la cantidad de movimiento del cuerpo se conserva".

Para comprobar la validez de la teoría empezaremos explicando qué dice la segunda ley de Newton. Ésta afirma que la fuerza neta aplicada sobre un objeto es el resultado de multiplicar su aceleración por su masa, siempre que éste no se vea afectado por fuerzas externas.

El teorema de la conservación de cantidad de movimiento se expresa en la ecuación (III.1).

$$\vec{F} = \Delta p/\Delta t \qquad \text{(III.1)}$$

$D\acute{o}nde: \rightarrow \overrightarrow{F}\ es\ la\ fuerza,$

$\Delta p\ la\ variación\ de\ la\ cantidad\ de\ movimiento,$

$\Delta t\ la\ variación\ de\ tiempo.$

Para demostrar su relación con la segunda ley de Newton tan solo hay que hacer unos cálculos, teniendo en cuenta que la masa del objeto, los podemos observar en la ecuación (III.2).

$$\vec{F} = \Delta p/\Delta t = (m * \vec{v_1} - m * \vec{v_0})/\Delta t = m * (\vec{v_1} - \vec{v_0})/\Delta t$$

$$= m * \vec{v}/\Delta t = m * \vec{a} \qquad \text{(III.2)}$$

$D\acute{o}nde: m\ es\ la\ masa\ del\ objeto,$

$(\overrightarrow{v_1}, \overrightarrow{v_0}) las\ velocidades\ final\ e\ inicial,$

$\vec{a}\ es\ la\ aceleración.$

Por tanto de ella podemos deducir que si la fuerza neta que actúa sobre un cuerpo es nula, se tiene que cumplir que la variación de cantidad de movimiento dividido por la variación de tiempo también será nulo y por lo tanto la cantidad de movimiento no variará durante ese intervalo de tiempo.

La verdadera utilidad del teorema se puede apreciar cuando se aplica para un conjunto de cuerpos. Gracias a ello podremos saber a cada momento las transferencias de energía entre ellos y éste funciona de la misma forma que para un objeto, siempre y cuando se cumpla que todas las fuerzas que intervengan en el sistema sean internas a este. Su funcionamiento se expresa en la ecuación (III.3).

$$\vec{p} = \vec{p_1} + \vec{p_2} + \vec{p_3} + \ldots + \vec{p_n} = \text{constante si } \vec{F} \text{ externas } = 0 \quad \text{(III.3)}$$

b) Conservación energía

La teoría de la conservación de energía viene técnicamente de la formulación de la tercera ley de Newton, la cual dice:

"Toda fuerza que actúa está sujeta a una fuerza de reacción de igual valor y contraria en sentido".

También denominado el principio de acción y reacción. Su validez depende de si el sistema no se ve afectado por una fuerza externa se mantiene estable. Esa premisa se mantiene exacta en la teoría de la conservación de energía para cuando hablamos energía mecánica.

Existen diferentes tipos de energías mecánicas que conforman la naturaleza. Una de ellas es la energía potencial gravitatoria, que define la energía que tiene un cuerpo cuando se sitúa a una determinada altura de una superficie, en este caso la tierra. Su naturaleza viene definida por la masa del objeto que la posee, la gravedad de la tierra y la altura a la que se encuentra, como podemos observar en la ecuación (III.4).

$$E_p = m * g * h \quad \text{(III.4)}$$

$D\acute{o}nde: E_p \; es \; la \; energia \; potencial \; expresada \; en \; Joules,$

$m \; la \; masa \; del \; objeto, g \; la \; gravedad \; de \; la \; Tierra,$

$h \; altura \; a \; la \; que \; est\acute{a} \; situado \; el \; objeto.$

No obstante esta ecuación solo es válida para distancias pequeñas. Para expresar relaciones del tipo la energía potencial que posee un satélite en órbita se tendría que utilizar una aproximación más real, para ilustrar nuestro caso ya es suficiente.

El otro tipo de energía mecánica que estudiaremos es la cinética. Es un tipo de energía asociada al movimiento y representa la energía que tiene un objeto cuando se mueve, su ecuación viene detallada en (III.5).

$$E_c = 1/2 * m * v^2 \quad \text{(III.5)}$$

$D\acute{o}nde: E_c\ es\ la\ energia\ cinética\ expresada\ en\ Joules,$

$v^2 el\ cuadrado\ de\ la\ velocidad\ a\ la\ que\ se\ mueve\ el\ objeto.$

El teorema de la conservación de energía establece que la energía mecánica total que tiene un sistema siempre es constante. La suma de energía potencial y cinética da como resultado la mecánica. Por lo tanto la suma de estas dos se mantendrá constante. Eso querrá decir que se producirá a lo largo del tiempo un traspaso de energía entre la componente potencial y la cinética y al revés.

Por lo tanto podemos escribir las ecuaciones (III.6).

$$E_{m1} = E_{m2}$$

$$E_{p1} + E_{c1} = E_{p2} + E_{c2} \quad \text{(III.6)}$$

$D\acute{o}nde: E_{m1}\ y\ E_{m_2}\ son\ las\ energias\ mecánicas\ en\ el\ tiempo\ 1\ y\ 2,$

$Ep_1\ y\ E_{p2}\ sus\ energias\ potenciales,$

$E_{c1}\ y\ E_{c2}\ sus\ energias\ cinéticas.$

Es un principio muy útil que se puede aplicar para gran cantidad de fenómenos como por ejemplo choques, cálculos de movimiento, tiempos de caída de objetos, velocidad de impacto de un objeto, etc. Para ello se aplica la ecuación de conservación de energía para varios instantes de tiempo, a partir de ella se desglosan las ecuaciones para hallar las velocidades que queremos saber, masas, altura de caída, etc.

c) Coeficiente de restitución

El coeficiente de restitución es una propiedad física de los materiales. Cada uno tiene el suyo y básicamente depende de sus propiedades químicas. Representa en qué medida un objeto va a perder energía cuando se interactúa con él, es decir cuando sufre un choque, cuando se desplaza, cuando se deforma, etc.

Para el proyecto ha sido una formula que nos ha ayudado a entender y poder resolver ecuaciones que más adelante explicaremos, como por ejemplo para los cálculos que hemos realizado para la resolución de choques.

Su origen viene fruto de la experimentación con las teorías de conservación de la cantidad de movimiento y la conservación de la energía. Es una propiedad que puede hallarse de una forma empírica y nos informa del grado de conservación de energía cinética que el material de un cuerpo posee.

La ecuación que la define se puede observar en (III.7).

$$K = -(v_1 - v_2)/(v_1' - v_2') \quad \text{(III.7)}$$

$D\acute{o}nde: K\ es\ el\ coeficiente\ de\ restitución,$

$(v_1\ y\ v_2\ )\ los\ vectores\ velocidad\ en\ el\ primer\ instante\ de\ tiempo,$

$(v_1'\ y\ v_2')\ los\ vectores\ velocidad\ en\ el\ segundo\ instante\ de\ tiempo.$

El rango del coeficiente de restitución siempre estará entre 0 y 1. Un material que tenga un 1 conservará la energía mecánica al máximo, no producirá un traspaso de energía de otra forma que no sea mecánica, lo cual provocará que los choques que se realicen contra ella sean perfectamente elásticos. En cambio un factor cercano al 0 producirá una enorme transformación energética, traduciéndose en deformación en el objeto o en producción de calor.

Anexo IV: Resolución de choques

Una vez que el sistema ha detectado que se ha producido un choque se pasa a la fase de resolución de choques. Encargada de calcular la reacción que se produce en los cuerpos que impactan partiendo de una serie de condiciones iniciales como por ejemplo la velocidad lineal y angular que llevaban, su masa, su momento de inercia, etc.

Los choques que se producen en un videojuego se suelen resolver por pares de objetos. Este modelo no se asemeja a la realidad, ya que estamos condicionando el cálculo del par siguiente a las condiciones de respuesta del anterior, pero de lo contrario se convertiría en un problema muy difícil de resolver, incluso imposible según el número de objetos que intervengan en el problema. Por ese motivo todos los modelos de resolución de choques que se detallarán aquí se realizarán entre choques de dos objetos.

Para el proyecto se ha valorado el estudio e implementación de dos métodos diferentes: el primero basado en respuesta de choques de partículas y el segundo la respuesta de choques basada en solidos rígidos. Seguidamente encontraremos el estudio realizado para llevar a cabo la implementación. Para finalizar detallaremos algunos de los problemas que hemos encontrado a lo largo de la implementación de esta parte, e intentaremos explicar las soluciones que hemos propuesto.

a) Partículas

La resolución de choques de partículas se basa en dos teorías ya comentadas en la parte de dinámica: la conservación de la cantidad de movimiento y el coeficiente de restitución. Su objetivo es saber la velocidad lineal de los dos objetos que han chocado. Este modelo de resolución de colisiones es un modelo que no tiene en cuenta factores como por ejemplo en qué lugar de un objeto se ha impactado o qué forma tiene este objeto. Por lo tanto nos encontramos ante un modelo no demasiado realista pero que funciona bien para choques a muy pequeña escala (y sobretodo para juegos en dos dimensiones).

Toda la serie de pasos que detallan el procedimiento realizado para la resolución de choques de partículas es el que podemos observar a continuación:

1) Partiendo de la ecuación del coeficiente de restitución que mostramos en (IV.1).

$$v_1 - v_2 = -K * (v'_1 - v'_2) \quad \text{(IV.1)}$$

$D\acute{o}nde: (v_1, v_2)\ son\ los\ vectores\ de\ velocidad\ en\ el\ instante\ inicial,$

$(v'_1, v'_2)\ los\ vectores\ velocidad\ en\ el\ instante\ siguiente,$

$K\ el\ coeficiente\ de\ restitución.$

2) Aislamos de (1) cualquiera de las dos velocidades a calcular y obtenemos (IV.2).

$$v'_2 = (v_1 - v_2) * K - v'_1 \quad \text{(IV.2)}$$

3) Seguidamente utilizaremos la ecuación de conservación de la cantidad de movimiento mostrada en (3)

$$m_1 * v_1 + m_2 * v_2 = m_1 * v'_1 + m_2 * v'_2 \quad \text{(IV.3)}$$

$D\acute{o}nde: m_1 es\ la\ masa\ del\ objeto\ 1,$

$m_2\ es\ la\ masa\ del\ objeto\ 2.$

4) Sustituimos la ecuación (IV.2) en la (IV.3) y seguidamente aislamos la velocidad que nos interesa calcular, en este caso v'_2 obteniendo (IV.4).

$$v_2' = ((m_2 - m_1 * K) * v_2 + m_1 * (1 + K) * v_1)/(m_1 + m_2) \quad \text{(IV.4)}$$

5) De esta forma calcularemos una de las dos velocidades de respuesta al choque, para calcular la otra tan solo aislaríamos de (IV.1) la otra velocidad (v'_1 y sustituiremos en (IV.3) como antes, obteniendo esta vez (IV.5).

$$v_1{}^{]} = ((m_1 - m_2 * K) * v_1 + m_2 * (1 + K) * v_2)/(m_1 + m_2) \quad \text{(IV.5)}$$

Cabe notar que todas las velocidades que aparecen son vectores de tres dimensiones. Para hacer el cálculo por cada dimensión tan solo hay que utilizar la componente de velocidad adecuada.

b) Sólidos rígidos

El procedimiento para la resolución de choques entre sólidos rígidos es bastante más complejo y realista que la simulación por partículas. Es un modelo que funciona bastante bien en el ámbito que vamos a trabajar, en el ámbito 3D.

En este modelo se tienen en cuenta muchos factores de los objetos que impactan, como por ejemplo su forma, su centro de masas, su tensor de inercia o su velocidad linear y angular.

Para empezar la resolución del choque entre dos sólidos necesitaremos saber una serie de datos que utilizaremos para los cálculos, entre ellos las masas, las velocidades lineales al impactar, sus velocidades angulares, sus centros de masas, el vector normal de la colisión y sus momentos de inercia. Antes de continuar describiremos que significan cada uno de ellos:

El **centro de masa** de un objeto es el punto geométrico donde se aplica la velocidad lineal que posee. Es el punto de referencia para su equilibrio y tiene la característica de que si aplicáramos justo sobre él una fuerza no provocaríamos ningún tipo de velocidad angular sobre el objeto.

La información sobre el centro de masas de los objetos que utilizamos es un dato que nos proporciona el motor de física de Unity. Para su cálculo hay que tener en cuenta cosas como la distribución de masas dentro de un objeto, si es una distribución homogénea o si no lo es. Para el proyecto hemos utilizado objetos que tienen una distribución de masa homogénea. Eso quiere decir que todas las partículas que los componen tienen la misma densidad. Para su cálculo teórico se utiliza el cálculo integral de tres dimensiones.

El **vector normal de colisión** que necesitamos es un dato que nos proporciona Unity al recopilar información sobre choques. Nos muestra el vector normal de la cara contra la que se ha chocado. Para la resolución del choque utilizaremos el vector normal que nos proporciona la cara contra el objeto que hemos impactado. Eso significa que no utilizaremos el del jugador si no el de la estructura contra la que se ha impactado. La elección del vector normal es indiferente, tan solo se escoge para dar un punto de referencia para los demás cálculos.

El **momento de inercia** de un objeto significa cómo responde un objeto cuando lo hacemos girar respecto a un punto de giro. Cada objeto tiene un momento de inercia diferente y es un valor de tipo escalar que nos ayuda a comprender cómo reacciona la velocidad angular cuando se aplica un movimiento rotacional sobre un objeto. Este valor depende principalmente de la distribución de masa del objeto cuando se hace girar. El momento de inercia de los objetos que necesitamos es un dato que hay que calcular dependiendo del tipo de objeto ante el que nos encontramos. Para el proyecto se han utilizado objetos sólidos con formas que tienen la característica de que la distancia de cada uno de sus puntos respecto al centro de masas es más o menos la misma. Eso es un factor que facilita el cálculo de sus momentos de inercia, ya que como mencionamos anteriormente su cálculo depende en gran medida de su forma. Para objetos del tipo que hemos utilizado existen tablas donde se puede consultar cómo calcular el momento de inercia para un cubo, una pirámide o una capsula por ejemplo, realizando una serie de cálculos utilizando sus características geométricas y físicas.

Una vez comentados todos estos factores procederemos a realizar los cálculos paso a paso de la resolución de la colisión, para entender más acerca de la resolución se ha elaborado un esquema del choque (ver figura IV.1).

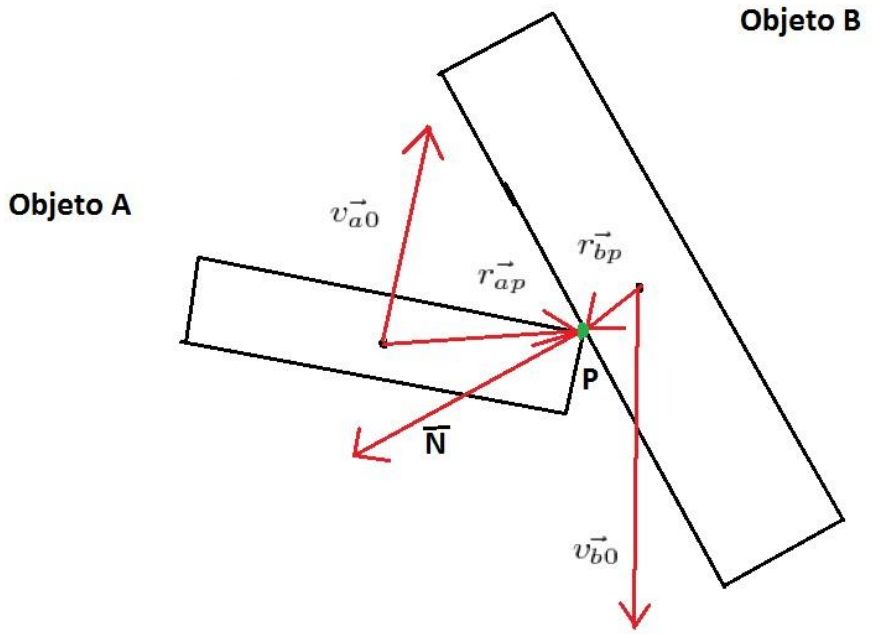

Figura IV.1: Colisión de dos objetos

En la figura IV.1 podemos observar cómo dos objetos están impactando, en ella podemos localizar seis datos de interés:

- $\overrightarrow{v_{a0}}$ *es la velocidad lineal inicial del objeto A,*
- $\overrightarrow{v_{b0}}$ *es la velocidad lineal inicial del objeto B,*
- *P es el punto de choque entre los dos objetos,*
- $\vec{n}$ *es el vector normal del choque,*
- $\overrightarrow{r_{ap}}$ *es el vector distancia entre el punto P y el centro de masas de A,*
- $\overrightarrow{r_{bp}}$ *es el vector distancia entre el punto P y el centro de masas de B.*

Los pasos para la resolución del choque son los siguientes:

1) En primer lugar calcularemos la velocidad que tienen los puntos de colisión para los objetos A y B (IV.6).

$$\vec{v_{ap0}} = \vec{v_{a0}} + \omega_{a0} \times \vec{r_{ap}}$$
$$\vec{v_{bp0}} = \vec{v_{b0}} + \omega_{b0} \times \vec{r_{bp}} \quad \text{(IV.6)}$$

Dónde: $\overrightarrow{v_{ap0}}$ *es la velocidad inicial del punto de colision en A,*

$\overrightarrow{v_{bp0}}$ *es la velocidad inicial del punto de colisión en B,*

$\overrightarrow{w_{a0}}$ *es la velocidad angular inicial de A,*

$\overrightarrow{w_{b0}}$ *es la velocidad angular inicial de B,*

la operacion X es el producto vectorial de los vectores.

2) Luego calculamos la velocidad relativa inicial de los dos objetos (IV.7).

$$\vec{v_{ab0}} = \vec{v_{ap0}} - \vec{v_{bp0}} \quad \text{(IV.7)}$$

Dónde: $\overrightarrow{v_{ab0}}$ *es la velocidad relativa inicial entre A y B.*

3) Seguidamente calcularemos el factor de impulso debido al choque (IV.8).

$$\mathbf{J} = \frac{-(1+\mathbf{K})\bar{\mathbf{v}}_{ab1} \cdot \bar{\mathbf{n}}}{1/m_a + 1/m_b + (\bar{\mathbf{r}}_{ap} \times \bar{\mathbf{n}})^2/I_a + (\bar{\mathbf{r}}_{bp} \times \bar{\mathbf{n}})^2/I_b} \quad \text{(IV.8)}$$

Dónde: J *es el factor de impulso fruto del choque,*

K *es el coeficiente de restitución de energía del choque,*

I_a *es el momento de inercia del objeto A.*

I_b *es el momento de inercia del objeto B.*

$\overrightarrow{r_{ap}}$ *es el vector distancia entre el punto P y el centro de masas de A,*

$\overrightarrow{r_{bp}}$ *es el vector distancia entre el punto P y el centro de masas de B.*

4) Una vez calculado el factor de impulso podremos resolver la velocidad final lineal de A y B y sus velocidades angulares finales (IV.9)(IV.10(IV.11)(IV.12).

$$\vec{v_{a1}} = \vec{v_{a0}} + (J * \vec{n})/m_a \quad \text{(IV.9)}$$

Dónde: $\overrightarrow{v_{a1}}$ *es la velocidad lineal de A después del choque.*

$$\vec{v_{b1}} = \vec{v_{b0}} - (J * \vec{n})/m_b \quad \text{(IV.10)}$$

Dónde: $\overrightarrow{v_{b1}}$ *es la velocidad lineal de B después del choque.*

$$\omega_{a1} = \omega_{a0} + (\vec{r_{ap}} \times (J * \vec{n}))/I_a \quad \text{(IV.11)}$$

Dónde: $\overrightarrow{w_{a1}}$ *es la velocidad angular de A después del choque.*

$$\omega_{b1} = \omega_{b0} - (\vec{r_{bp}} \times (J * \vec{n}))/I_b \quad \text{(IV.12)}$$

Dónde: $\overrightarrow{w_{b1}}$ *es la velocidad angular de B después del choque.*

Por otra parte, modificando la ecuación (IV.8) podemos simular en nuestro juego choques contra objetos inamovibles, para ello supondremos (IV.13), y la ecuación final quedará como (IV.14).

$$m_b \to \infty \qquad I_b \to \infty \quad \text{(IV.13)}$$

$$J = \frac{-(1+K)\,\bar{v}_{ap1} \cdot \bar{n}}{1/m_a + (\bar{r}_{ap} \times \bar{n})^2 / I_a} \quad \text{(IV.14)}$$

c) Problemas encontrados

Uno de los problemas más comunes en lo que a simulación de física por ordenador se refiere es el problema de la naturaleza discreta de ésta. Un problema que se agrava con la inclusión de física específica para interacciones entre más de un objeto, tales como los que ya hemos visto: detección de choques y resolución de choques.

Aunque actualmente utilicemos ordenadores muy rápidos, los cálculos para simulaciones se realizan a un paso fijo. Eso quiere decir que los cálculos de física se realizan mediante una discretización del tiempo contínuo. Por lo tanto por muy pequeño que sea el intérvalo de tiempo entre dos instantes de tiempo consecutivos, entre ellos dos no sabremos que estará ocurriendo.

Eso comporta que el intérvalo de tiempo que ocurre entre cálculo y cálculo pueda suponer un fallo grave de precisión en los motores de física.

Para ilustrar un problema de este tipo podemos observar la figura IV.2.

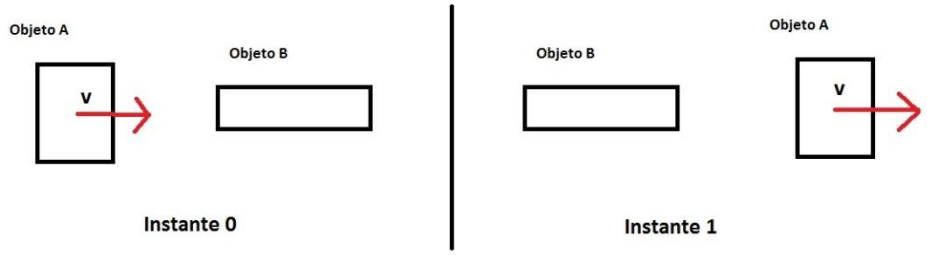

Figura IV.2: Simulación de fallo en detección de colisiones discretas

Cómo podemos observar la figura 2, entre el instante de tiempo 0 y el 1 la simulación de física no tiene constancia de que el objeto A haya colisionado con el objeto B, ya que A iba a una velocidad suficiente grande para producir el error.

Para poder evitar problemas de este tipo se utilizan unas técnicas llamadas detección de colisión continua. Ellas se basan en predecir los movimientos que va a realizar determinado objeto e intentar detectar colisiones que se produzcan. Para ello se utiliza un método llamado detección de colisión continua [Ccd] como el que podemos observar en la figura IV.3.

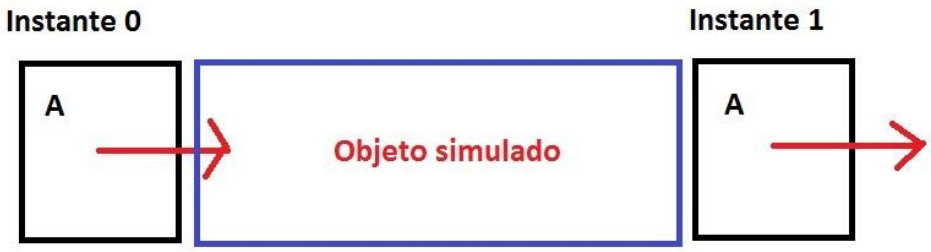

Figura IV.3: Implementación de CCD con simulación de objeto

Como vemos en la figura 3, entre cada instante de tiempo de simulación se coloca un objeto entre la posición del objeto A actual y la posición que tendrá en el siguiente instante de simulación. Si ese objeto choca contra algo sabremos que hay algo entre medio. Una vez detectado el choque se realizará una búsqueda de la zona de impacto. Se hará acotando cada vez más la zona de choque hasta encontrar la posición exacta. Una vez ahí se trasladará el objeto A al lugar y se empezará la resolución de la colisión. Otro método para la detección de colisión continua podría ser lanzar un rayo desde el objeto A en la dirección de la velocidad y de un módulo equivalente a la distancia que se recorriera entre el instante actual y el siguiente y detectar si choca contra algún objeto.

Anexo V: Diagrama de física

Para la implementación de las físicas hemos utilizado cuatro clases en total, la forma en la que se comunican con las demás del programa es mediante la clase que gestiona nuestro jugador en el juego, su estructura es la que podemos observar en la figura V.1.

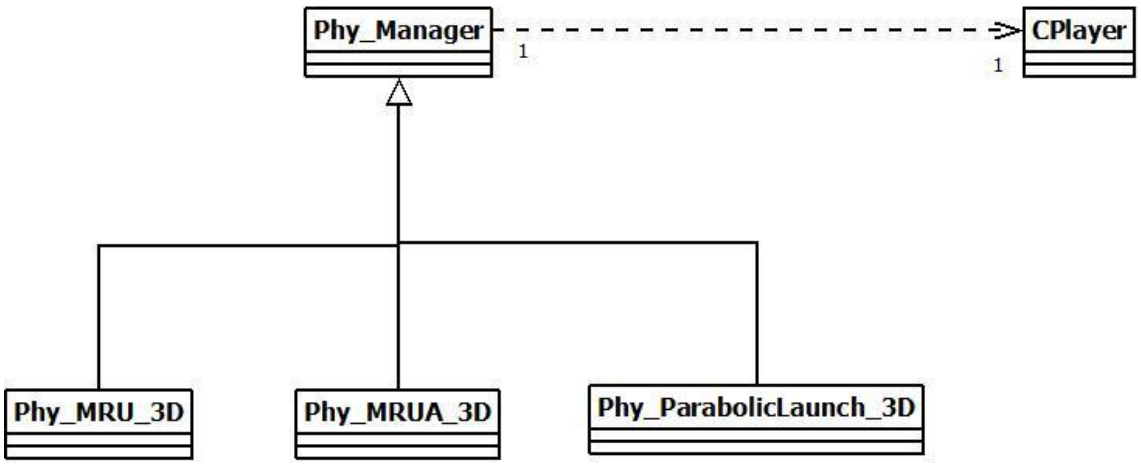

Figura V.1: Diagrama de clases para la implementación de la física

En la figura V.1 podemos observar la implementación que se ha realizado para los tres tipos de lanzamientos que se han utilizado: Phy_MRU_3D corresponde a la implementación del movimiento rectilíneo uniforme, Phy_MRUA_3D corresponde al movimiento rectilíneo uniformemente acelerado y para finalizar Phy_ParabolicLaunch_3D corresponde al movimiento parabólico. Todas y cada una de ellas corresponden a su implementación en tres dimensiones.

Las tres clases heredan de la clase Phy_Manager, la cual es un singleton [Sin]. Eso significa que solo habrá una única instancia de ella en todo el programa. Ella será la encargada de gestionar toda la física del juego.

El conjunto de la clase padre Phy_Manager y sus tres hijos (los tres tipos de movimientos) conforman un patrón denominado *template method* [Tem, que es un patrón que nos permite crear una superclase la cual contiene elementos que otras utilizarán, de forma que cuando vamos añadiendo subclases que heredan de ella estas podrán utilizar lo

que les interese de ella, añadir funcionalidades nuevas o incluso sobrescribir las funciones que se quieran modificar su comportamiento.

Su principal objetivo es el de reaprovechar código, pero un punto muy importante que tiene es que ayuda a simplificar la comunicación entre clases. El jugador (CPlayer) tan solo tiene un punto de acceso para utilizar la física. Este punto de acceso será a través de la interfaz del manager de física. Eso le permitirá abstraerse de su implementación y evitar dependencias fuertes de código. Además es un patrón de diseño que permite ser escalado fácilmente y en cualquier momento que se requiera se podrá añadir una nueva clase de física concreta que herede del manager de físicas sin que todas las clases que utilicen el manager se den cuenta.

Anexo VI: Diagrama gestor de archivos

Para implementar el sistema que accede a los archivos de datos del proyecto se ha implementado la estructura que podemos ver en la figura VI.1.

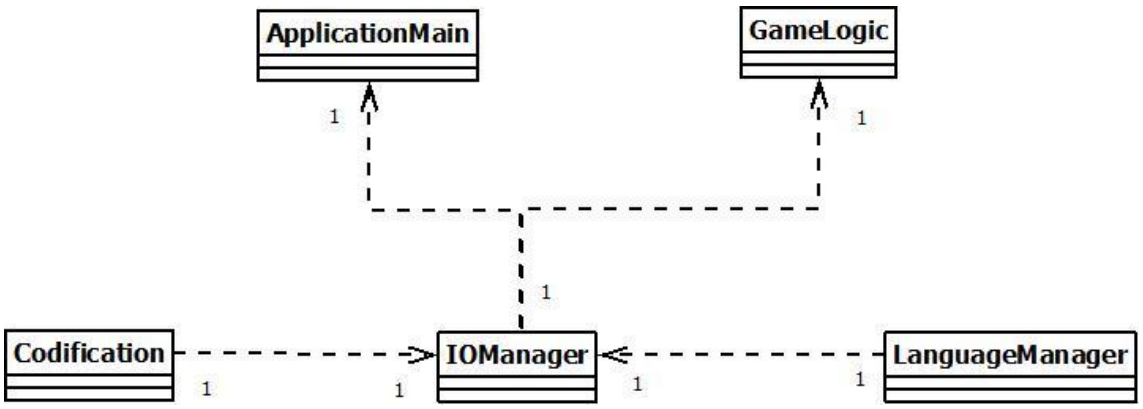

Figura VI.1: Diagrama de clases para el gestor de archivos

La estructura está compuesta de un manager (IOManager), éste depende de la existencia de la aplicación principal (AplicationMain) y de la lógica de juego (GameLogic). Para implementarlo se ha utilizado el patrón singleton para restringir su creación a una única instancia a lo largo del programa. El motivo es para evitar la posible desincronización entre archivos al leer y al escribir, al tener un único punto de acceso hacia ellos garantizamos que su acceso se realizará de una forma ordenada y sincronizada. Tampoco se abrirá más de una vez el mismo archivo, pudiendo provocar este hecho importantes fallos al escribir debido a la concurrencia en su acceso.

El gestor de idiomas (LanguageManager); por el mismo motivo también utiliza un patrón singleton, mientras que la clase encargada de la codificación y descodificación de archivos (Codification) es una clase estática para poder utilizarla a lo largo de todo el programa sin tener que crearla. La correspondencia con módulos del programa es: Application Main es “Lógica principal”, GameLogic es “Lógica de juego”, IOManager es “Gestor de archivos”, LanguageManager es “Gestor de idiomas”, Codification es “Codificación”.

Anexo VII: Manual de usuario

En el anexo siguiente se encuentra la lista de posibles acciones a ejecutar dentro de Launchageddon, se ha dividido en dos apartados:

- Utilizando teclado y ratón.
- Utilizando Kinect.

a) Utilizando teclado y ratón:

- **W-A-S-D:** Movimiento de cámara en el estado de equipamiento del juego, selección de estrategia de lanzamiento en el estado de juego de estrategia.
- **Espacio:** Tecla de acción, servirá para realizar todas las acciones del juego, lanzamiento, paso de transiciones de cámara, paro en mitad de un lanzamiento y seguir un lanzamiento parado.
- **Escape:** Mientras se juega pausa el juego, y muestra un menú de pausa.

b) Utilizando Kinect:

- Se utiliza la mano derecha para desplazarnos por los menús.
- Se realiza pulsación hacia dentro con la mano derecha para pulsar los botones de menú.
- En el estado de juego de estrategia podemos mover la cámara con la mano derecha.
- Para realizar las acciones el evento es el de levantar la mano izquierda hacia arriba.
- Para el traje paracaidista se utiliza la información del torso, el personaje seguirá la posición que tenga el torso de la persona que juega.

Anexo VIII: Encuesta usuarios

Edad: _____

Sexo: M ☐ **F** ☐

Puntúa del 1 al 5 los siguientes aspectos del videojuego: LAUNCHAGGEDON

1	No me gusta
2	Indiferente
3	Podría mejorar
4	Muy bien

1. Diseño del juego. (decorado, colorido, distribución de la pantalla). ☐

2. Jugar con el teclado y el ratón. (La familiarización con los controles) ☐

3. Sistema de puntuación del juego. ☐

4. ¿Qué te parece que el juego esté en tres idiomas (catalán, castellano, inglés)? ☐

5. Música y efectos de sonido. ☐

6. Cómo ves los choques en el juego, ¿son realistas? ☐

7. Mecánicas de juego, ¿es divertido? ☐

8. ¿Crees que sería un juego fácil de jugar en otras plataformas?(móvil y/o tablet) ☐

9. Innovación del juego.¿ Crees que la idea del juego es innovadora? ☐

10. ¿Lo consideras un juego para jugar con amigos y/o familia? ☐

11. ¿Es un juego sencillo de jugar y divertido? ☐