

DESARROLLO DE VIDEOJUEGO INDIE EN UNREAL ENGINE 4

FERNANDO JOSÉ GARCÍA PADILLA

Trabajo fin de Grado

Supervisado por Dr. Pablo Trinidad Martín-Arroyo



Universidad de Sevilla

agosto 2017

Publicado en agosto 2017 por
Fernando José García Padilla
Copyright © MMXVII

fergarpad@alum.us.es

Yo, D. Fernando José García Padilla con NIF número 47214882E,

DECLARO

mi autoría del trabajo que se presenta en la memoria de este trabajo fin de grado que tiene por título:

Desarrollo de videojuego indie en Unreal Engine 4

Lo cual firmo,

Fdo. D. Fernando José García Padilla
en la Universidad de Sevilla
29/08/2017

*Dedicado a todos los que me han apoyado
Y en especial a...*

*A mi familia, amigos, pareja y todos aquellos que me han dado ánimos para
realizar el proyecto.*



AGRADECIMIENTOS

No olvides añadir una nota de agradecimiento a quienes hayan contribuido emocionalmente al proyecto fin de Grado.

RESUMEN

El objetivo de este proyecto es la realización, haciendo uso de la ingeniería del software, de un videojuego independiente que sea exigente con sus mecánicas y se aventurare combinando varios de los géneros existentes en el mercado con el propósito de llegar a un público específico dentro del mismo. Para su ejecución se ha escogido hacer uso del motor gráfico Unreal Engine, un potente framework que nos ofrece las herramientas que vamos a necesitar durante el transcurso del desarrollo, cuyo uso es gratuito y además está muy extendido en la actualidad.

ÍNDICE GENERAL

I	Introducción	1
1.	Contexto	3
1.1.	El mundo del videojuego	4
1.1.1.	El boom de los e-sports y los juegos para móvil	7
1.1.2.	El auge del videojuego independiente	10
1.1.3.	La irrupción de los trofeos / logros y su impacto	12
1.2.	Estado del arte	14
2.	Objetivos del proyecto	17
2.1.	Motivación	18
2.2.	Listado de objetivos	20
2.2.1.	Objetivos del producto	20
2.2.2.	Objetivos individuales	22
II	Organización del proyecto	23
3.	Metodología	25
3.1.	Estructura organizacional del proyecto	26
3.2.	Metodología de desarrollo	27
3.2.1.	Presentación de la metodología	27

ÍNDICE GENERAL

3.2.2. Resumen de la metodología	27
3.2.3. Adaptación de la metodología	29
4. Planificación	31
4.1. Resumen temporal del proyecto	32
4.2. Planificación inicial	32
4.3. Informe de tiempos del proyecto	38
5. Costes	41
5.1. Resumen de costes del proyecto	42
5.2. Costes de personal	42
5.3. Costes materiales	50
5.4. Costes indirectos	53
III Desarrollo del proyecto	55
6. Arranque	57
6.1. Lista de características	58
6.2. Diseño arquitectónico	61
6.2.1. Módulos de juego	62
6.2.2. Plugins	63
6.2.3. Clases de gameplay	64
7. Iteración cero	67
7.1. Inicialización del proyecto	68
7.2. Puesta a punto y primeros pasos	69
7.3. Directrices generales y reseñas	69

7.3.1. Uso de animaciones profesionales «Mixamo»	70
7.3.2. Implementación del sistema de animación del personaje	70
7.3.3. Control de que el personaje no abandone el mapa	70
8. Primera iteración	71
8.1. Características a desarrollar	72
8.2. Diseño	74
8.3. Implementación	86
8.4. Pruebas	93
8.5. Despliegue	94
9. Segunda iteración	95
9.1. Características a desarrollar	96
9.2. Diseño	98
9.3. Implementación	106
9.4. Pruebas	119
9.5. Despliegue	120
10. Tercera iteración	121
10.1. Características a desarrollar	122
10.2. Diseño	122
10.3. Implementación	122
10.4. Pruebas	122
10.5. Despliegue	122
11. Cuarta iteración	123
11.1. Características a desarrollar	124

ÍNDICE GENERAL

11.2. Diseño	124
11.3. Implementación	124
11.4. Pruebas	124
11.5. Despliegue	124
12. Quinta iteración	125
12.1. Características a desarrollar	126
12.2. Diseño	126
12.3. Implementación	126
12.4. Pruebas	126
12.5. Despliegue	126
13. Sexta iteración	127
13.1. Características a desarrollar	128
13.2. Diseño	128
13.3. Implementación	128
13.4. Pruebas	128
13.5. Despliegue	128
IV Cierre del proyecto	129
14. Manual de usuario	131
14.1. Controles	132
14.1.1. Controles con teclado y ratón	132
14.1.2. Controles con mando	134
14.2. Menú principal	136

14.3. Menú de pausa	137
14.4. Menú de ajustes	138
14.5. HUD del usuario	140
15. Conclusiones	145
15.1. Informe post-mortem	146
15.1.1. Lo que ha ido bien	146
15.1.2. Lo que ha ido mal	149
15.2. Discusión	153
15.3. Trabajos futuros	154
V Appendices	157
A. Glosario	159
B. Documento de diseño	165
B.1. Visión de conjunto	166
B.1.1. Plataformas de destino	166
B.1.2. Requisitos mínimos objetivo	166
B.1.3. Estilo gráfico	167
B.1.4. Estilo sonoro	167
B.2. Comienzo	167
B.2.1. Menú principal	167
B.2.2. Comienzo del juego e introducción	167
B.2.3. In-game HUDs y menús	167
B.3. Interfaz de usuario	167

ÍNDICE GENERAL

B.3.1.	Menús	167
B.3.2.	In-game HUD	168
B.3.3.	In-game Options	168
B.3.4.	Pantalla de fin de juego	168
B.3.5.	Pantalla de selección de nivel	168
B.4.	Gameplay	169
B.4.1.	Mecánicas	169
B.4.2.	Controles	172
B.4.3.	Modos de juego	172
B.4.4.	Ganando el juego	172
B.4.5.	Logros / Trofeos	172
B.5.	Assets	182
B.5.1.	Personajes	182
B.5.2.	Enemigos	182
B.5.3.	Animaciones	183
B.5.4.	Equipo y mejoras	183
B.5.5.	Entorno	183
B.5.6.	Audio	183
	Referencias bibliográficas	183

ÍNDICE DE FIGURAS

1.1. Beneficios de los videojuegos en Europa Occidental (datos en billones americanos) [18]	4
1.2. Evolución gráfica de los videojuegos	5
1.3. Relación entre el dinero generado por los videojuegos, la música grabada y el cine durante 2015 en España [8]	6
1.4. Audiencia y previsión de crecimiento de los espectadores de deportes electrónicos [17]	7
1.5. Comparativa, beneficios y previsiones entre los juegos para móvil y el resto de aplicaciones (datos en billones americanos) [16]	8
1.6. Relevancia del programa «Greenlight» en los lanzamientos indies [28]	10
1.7. Ejemplos de videojuegos indies se hicieron un hueco en el mercado	11
1.8. Captura del juego «E-Motion» (1990), para la videoconsola «Amiga», uno de los primeros juegos en implementar algún tipo de logro	12
1.9. Imagen del sistema de logros interno de «Minecraft» (2011), que se mantuvo hasta la versión 1.11	13
1.10. Imagen del juego «E.T.» para la «Atari 2600», uno de los máximos detonantes de la crisis del videojuego [21]	14
1.11. Fotograma de «La abadía del crimen» (1987), uno de los máximos estandartes de «la edad de oro del software español»	15
1.12. Imagen de la primera demostración técnica en tiempo real de Unreal Engine 4, titulada «Infiltrator»	16
2.1. Fotograma de «Bioshock» (2007), uno de los pocos «First Person Shooter» («FPS») que se supieron diferenciar dentro de la saturación del género	18

ÍNDICE DE FIGURAS

2.2. Captura de «Enter the gungeon» (2016), ejemplo de la forma más tradicional de bullet hell en dos dimensiones	19
3.1. Organigrama del proyecto	26
3.2. Esquema del desarrollo basado en funcionalidades	28
3.3. Esquema del desarrollo basado en funcionalidades adaptado para el proyecto	30
4.1. Equivalencias fases-metodología	34
5.1. Diferencias euro-dólar entre Agosto de 2015 y Julio de 2017 [13]	45
6.1. Arquitectura Unreal Engine 4	61
7.1. Ventana del creador de proyectos de Unreal Engine	69
14.1. Diagrama de la asignación de acciones del teclado	132
14.2. Diagrama de la asignación de acciones del ratón	132
14.3. Diagrama de la asignación de acciones del controlador	134
14.4. Fotograma del menú principal del software	136
14.5. Fotograma del menú de pausa del software	137
14.6. Fotograma del menú de ajustes del software	138
14.7. Instantánea mostrando el widget blueprint del HUD del usuario, en el que se pueden observar todos los elementos que se pueden mostrar en pantalla	140
14.8. Captura del juego que muestra distintos componentes del hud: punto de referencia (centro), indicador de «blink» (alrededor de éste último), brújula (parte superior) y barra de adrenalina (parte inferior)	141
14.9. Captura del juego tomada durante la obtención de un logro	142
14.10. Captura del juego tomada durante un tutorial	144

15.1. Captura del menú principal durante versiones tempranas del desarrollo	151
15.2. Fotograma del título «Fallout 4» (2015) que muestra una farola que emite un cono de luz	156

ÍNDICE DE CUADROS

4.1. Tabla resumen de tiempos y planificación	32
4.2. Planificación temporal de fases e iteraciones (planificación)	35
4.3. Planificación temporal detallada de fases e iteraciones	36
4.4. Tabla del calculo de horas por rol en iteraciones	37
4.5. Tabla resumen de horas de trabajo por rol	37
4.6. Tabla de tiempos empleados en fases e iteraciones detallados	38
4.7. Tabla de desviación de tiempo por iteración	39
4.8. Tabla de desviación de tiempo por rol	39
4.9. Tabla de retrasos del proyecto	40
5.1. Tabla resumen de costes	42
5.2. Salarios medios brutos según la encuesta salarial de «Gamasutra» (2014) [11]	43
5.3. Tabla comparativa costes de vida [19]	44
5.4. Salarios medios brutos (adaptados) según la encuesta salarial de «Gamasutra» (2014) [11]	46
5.5. Tabla salarial (sueldos brutos) del proyecto por hora	47
5.6. Tabla costes sueldos brutos	47
5.7. Tablas costes sueldo rol «jefe de proyecto»	48
5.8. Tablas costes sueldo rol «diseñador»	48
5.9. Tablas costes sueldo rol «programador»	48

5.10. Tabla de características del dispositivo sobremesa	51
5.11. Tabla de características del dispositivo portátil	51
5.12. Tabla de amortización de materiales	52
5.13. Tabla de costes indirectos	53
6.1. Prefijos de clases de gameplay en Unreal Engine	64
8.1. Memorando técnico 1001: Creación del personaje	74
8.2. Memorando técnico 1002: Creación de la cámara TFP	75
8.3. Memorando técnico 1003: Creación de la cámara TP	76
8.4. Memorando técnico 1004: Movimiento del personaje	77
8.5. Memorando técnico 1005: Rotación de cámara y personaje	78
8.6. Memorando técnico 1006: Rotación de cámara y personaje (para controladores)	79
8.7. Memorando técnico 1007: Sprint	80
8.8. Memorando técnico 1009: Agachar	80
8.9. Memorando técnico 1009: Agachar	81
8.10. Memorando técnico 0010: Tiempo bala	82
8.11. Memorando técnico 0011: Teletransporte o «blink»	83
8.12. Memorando técnico 0012: Control de límites del mapa	84
8.13. Memorando técnico 0013: Control de máxima distancia de caída	84
8.14. Memorando técnico 0014: Movimiento de piernas del personaje al girarse.	85
9.1. Trazadores de la base del sistema de escalada 1/2	98
9.2. Trazadores de la base del sistema de escalada 2/2	99
9.3. Memorando técnico 2002: Agarrar un saliente	100
9.4. Memorando técnico 2003: Agarrar un saliente pequeño para escalarlo	101

9.5. Memorando técnico 2004: Escalar un saliente al que estemos agarrados	101
9.6. Memorando técnico 2005: Soltar un saliente	102
9.7. Memorando técnico 2006: Movernos por un saliente	102
9.8. Memorando técnico 2007: Saltar desde saliente lateralmente	103
9.9. Memorando técnico 2008: Mirar hacia atrás desde saliente	103
9.10. Memorando técnico 2009: Saltar desde saliente mirando hacia atrás	104
9.11. Memorando técnico 2010: Escalada en pared	105
9.12. Memorando técnico 2011: Giro y salto durante escalada en pared	105
 B.1. Logro historia - Juego	173
B.2. Logro historia - Tutorial	173
B.3. Logro historia - Capítulo I	174
B.4. Logro historia - Capítulo II	174
B.5. Logro prueba de tiempo - Capítulo I	175
B.6. Logro prueba de tiempo - Capítulo II	175
B.7. Logro colecciónables - 1 colecciónable	176
B.8. Logro colecciónables - 5 colecciónables	176
B.9. Logro colecciónables - 10 colecciónables	177
B.10. Logro colecciónables - 50 % de colecciónables	177
B.11. Logro colecciónables - 100 % de colecciónables	178
B.12. Logro misceláneos - 100 saltos	179
B.13. Logro misceláneos - 1000 saltos	179
B.14. Logro misceláneos - 10000 saltos	180
B.15. Logro 100%	181
B.16. Assets de personajes	182

PARTE I

INTRODUCCIÓN

CONTEXTO

1

2 *n este primer capítulo abordaremos el entorno que rodea al producto software que pre-*
3 *tendemos crear, la industria del videojuego, con el objetivo de situarnos para conocer*
4 *las oportunidades que ofrece el mercado y los riesgos, relacionados con el mismo, que*
5 *debemos tener presentes.*

1.1 EL MUNDO DEL VIDEOJUEGO

La industria del videojuego vive, desde hace años, una etapa de popularización y expansión. Según datos de la asociación española de videojuegos («AEVI») y del informe anual de Newzoo [3, 18] el pasado año el sector creció un 8.5%, lo que representa pasar de 91.800 millones de dólares en 2015 a 99.600 millones de dólares en 2016. Además, esta tendencia es global y aumenta en cada uno de los cinco continentes, siendo la región geográfica que representan África y Oriente medio la que más prosperó en el transcurso del año.

Teniendo siempre presente que estamos en un mercado global y que lanzar un producto desarrollado en nuestro país, si se satisfacen necesidades como la localización del producto, puede repercutir en ventas en todo el mundo, España se sitúa en la octava posición del ranking mundial, gracias a la nada despreciable cifra de 1.812 millones de dólares de beneficios generados en 2016, pero lejos de los dos principales países, China y Estados Unidos, que sobrepasan con holgura los 20.000 millones de beneficios en el pasado año [18].

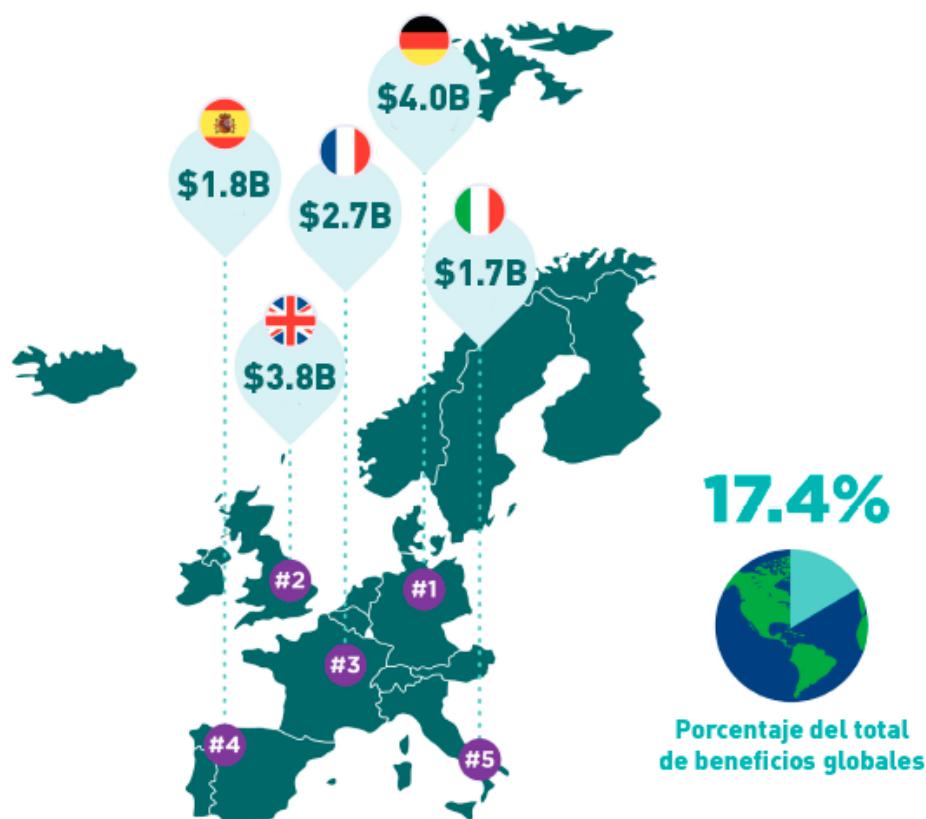


Figura 1.1: Beneficios de los videojuegos en Europa Occidental (datos en billones americanos) [18]

1 La transformación del sector no tiene sólo que ver con cifras económicas y no me
 2 gustaría pasar por alto algo que bajo mi punto de vista es bastante más interesante
 3 y que, aunque bien daría para un trabajo aparte, describe perfectamente el giro tan
 4 drástico al que se está viendo sometida la industria en los últimos años: hace no tanto
 5 tiempo los videojuegos eran un territorio un tanto desconocido, lo que provocaba que
 6 en algunas ocasiones fuesen vistos con cierto recelo por una parte de la población. Es-
 7 ta tendencia se ha revertido vertiginosamente en un breve lapso de tiempo, e incluso
 8 se podría decir que los videojuegos, en gran medida, han pasado a formar parte de la
 9 cultura popular: las grandes empresas textiles hacen camisetas con referencias a video-
 10 juegos o videoconsolas reconocidas; se realizan grandes eventos mediáticos en torno
 11 videojuegos o personas relacionadas con los videojuegos, a las que se les da el trato
 12 de auténticas estrellas del rock; se crean canales de televisión en grandes plataformas
 13 audiovisuales dirigidos únicamente a los videojuegos y podríamos decir que toda esta
 14 vorágine alcanza su súmmum cuando en un evento de tanta proyección internacional
 15 como es el acto de clausura de los juegos olímpicos, el primer ministro del país que re-
 16 cibe el testigo para organizarlos en cuatro años, en este caso Japón, aparece disfrazado
 17 de un reconocido personaje de videojuegos: «Mario».



Figura 1.2: Evolución gráfica de los videojuegos

18 Aún queda un largo camino por recorrer, es un problema que persigue al sector
 19 desde su creación, pero en los últimos años hemos vivido una apertura de los video-
 20 juegos a la sociedad de tal magnitud que una parte relevante de la misma lo ha pasado
 21 a ver de una forma distinta, dejando de lado ese citado recelo o cualquier connota-
 22 ción negativa, lo cual sin duda no puede ser más que favorable para la industria y su
 23 crecimiento.

Estableciendo un símil, podríamos comparar en algunos aspectos la fase en la que se encuentra la industria del videojuego con aquella en la que se podría encontrar la industria cinematográfica hace unas décadas: el sector está en etapa de expansión, cada vez más universidades empiezan a formar en ámbitos relacionados directamente con los videojuegos, se crean cada vez más puestos de trabajo bajo su cobijo [26] e incluso están creciendo verdaderas celebridades en torno a la industria, ya que es habitual que las personas que son consumidores de la industria conozcan directores de juegos, compositores o jugadores profesionales, por citar unos ejemplos. Igualmente, no todos los aspectos el sector cinematográfico está más avanzado, ya que no podemos comparar las industrias del videojuego y del cine pasando por alto que la primera, pese a su más que notable juventud, consigue doblar en beneficios a la segunda en nuestro país [8].

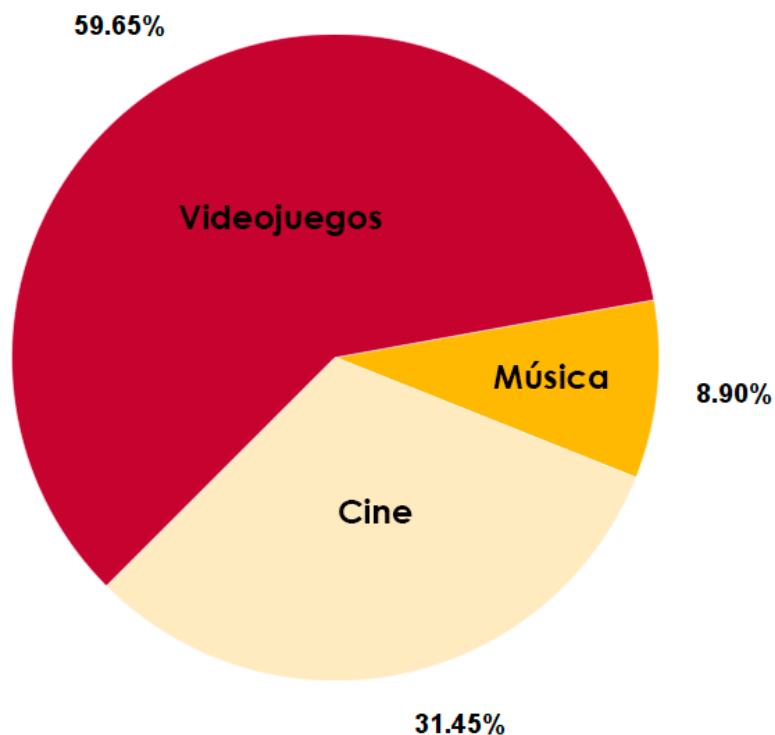


Figura 1.3: Relación entre el dinero generado por los videojuegos, la música grabada y el cine durante 2015 en España [8]

En conclusión, es cuestión de tiempo que a los videojuegos se le otorgue socialmente el mismo estatus que al cine: ser calificados de cultura o arte, y remarco socialmente porque en países como el nuestro ya están catalogados como cultura, pero ese mensaje aún está por calar en muchos estratos de la sociedad.

1.1.1 El boom de los e-sports y los juegos para móvil

Aunque ninguno de los dos sectores estén estrictamente relacionados con lo que nos ocupa en este proyecto, son dos importantes motores de la industria que debemos analizar para contextualizarnos y entender plenamente la magnitud del mercado de los videojuegos actualmente.

Los deportes electrónicos, también denominados e-sports, son prueba del auge que está viviendo la industria, de los puestos de trabajo que genera en diversos sectores y de las posibilidades de negocio que hay alrededor del éxito que están viviendo actualmente. Actualmente sufren un vertiginoso ascenso y muestra de ello son las cifras de espectadores, que suben año tras año. Actualmente se cifran en 385 millones las personas que siguen los deportes electrónicos (entre espectadores ocasionales y entusiastas, con una proporción similar para ambos), con el pronóstico de llegar a 589 millones de espectadores en 2020, lo que supone un incremento de más del 50% en 3 años [17].

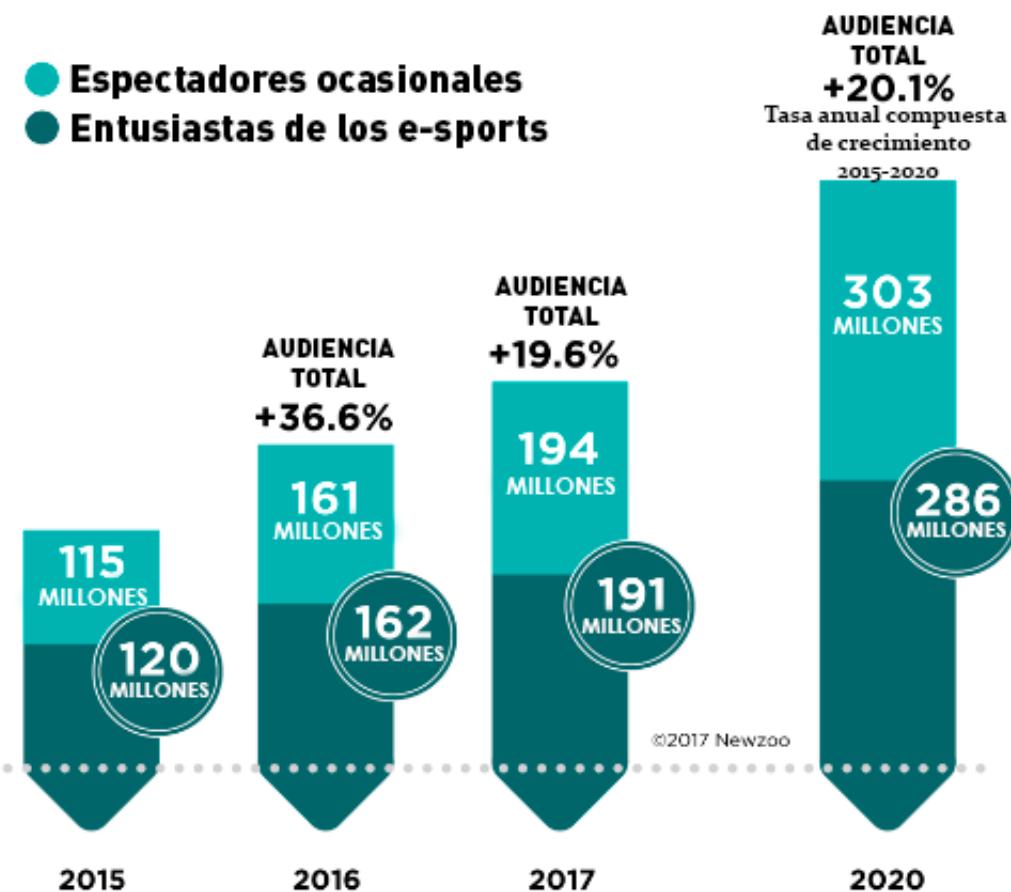


Figura 1.4: Audiencia y previsión de crecimiento de los espectadores de deportes electrónicos [17]

Igualmente, como se describía en la sección anterior, cadenas de ámbito nacional ya dedican algunos de sus diales a tiempo completo a los videojuegos y, realmente, casi la totalidad de estos suelen estar ligados a los e-sports, algo que actualmente es tendencia en los estados de nuestro entorno y algo normal desde hace años en algunos países concretos como Corea del Sur.

Aún así, la fuerza de los e-sports va más allá: los deportes electrónicos mueven millones de personas en todo el mundo, consiguen llenar estadios con sus eventos, reparten millones en premios y, quizás algo que es mucho más importante, generan un fenómeno fan que, aunque pueda impresionar si no se conoce a fondo este sector, está en vías de convertirse en uno tan poderoso como el que podría ser a día de hoy el fútbol: como se comentaba previamente los espectadores de este tipo de entretenimiento no paran de crecer, pero no sólo eso, un amplio sector conoce los equipos, sus jugadores y, de manera similar como ocurre en los deportes tradicionales, muchos de ellos marcan en el calendario el siguiente torneo de su juego preferido o el próximo partido de su equipo favorito. El fenómeno es tal que incluso se realizan realities de cómo estos equipos de deportes electrónicos viven o entranan.

Este fenómeno coincide en el tiempo con la expansión del mercado del videojuego en los terminales móviles, el cuál genera una importante cantidad de dinero para la industria y, de igual manera que pasa con los deportes electrónicos, está en alza con beneficios que superan los 36.000 millones de dólares en 2016 [16, 18], lo que supone en torno al 37% de los beneficios totales de la industria obtenidos el pasado año [18].



Figura 1.5: Comparativa, beneficios y previsiones entre los juegos para móvil y el resto de aplicaciones (datos en billones americanos) [16]

1 Además, los dispositivos móviles están ayudando a que la industria llegue a un
2 número mucho mayor de personas y, lo que es mucho más importante, de una manera
3 más indirecta: ya no es necesario que el usuario tenga el deseo específico de jugar a
4 videojuegos y realice un desembolso para adquirir una consola u ordenador gaming,
5 algo que sin duda puede hacer que muchos usuarios potenciales se pierdan en el proce-
6 so, sino que cualquiera que disponga de un teléfono móvil, sea cual sea su plataforma,
7 puede gozar de un amplio catálogo de juegos y de una manera muy accesible, por lo
8 que tarde o temprano la mayoría probará lanzar algún videojuego en su dispositivo y
9 parte de ellos pasarán a ser usuarios habituales de la industria, con las repercusiones
10 que ello conlleva.

11 Este cambio en la accesibilidad amplía no sólo el espectro de jugadores que llegarán
12 a poseer una consola o un ordenador gaming para tener más títulos a su disposición,
13 generando beneficios para la industria en el proceso, sino que llega a sectores de la
14 población a los que antes era difícil acceder, como el de las personas de mediana edad,
15 y provocan cambios en las tendencias tales como que las mujeres igualen a los hombres
16 [27], en un sector tradicionalmente de los últimos, en el número total de jugadores.

1.1.2 El auge del videojuego independiente

Centrándonos más en el proyecto que nos ocupa, cabe destacar el fenómeno de los videojuegos indie: juegos desarrollados con poco presupuesto, por un equipo de desarrollo pequeño y que, generalmente, se atreven a arriesgarse más que la media puesto que, tradicionalmente, los juegos desarrollados por grandes empresas y que cuentan con un gran presupuesto detrás tienden a asegurar más, a ser un producto similar a lo que ya está triunfando en ese momento en el mercado. En definitiva, no suelen estar dirigidos al gran público sino que suelen ser un producto de nicho dirigido a un sector muy específico dentro de la comunidad de jugadores. Además, cabe destacar que gran parte de la culpa de su irrupción la tuvo la consolidación distribución digital, que abrió puertas a muchos desarrolladores y abarató costes, ya que es más rentable para el desarrollador que negociar con una distribución tradicional.

Juegos publicados por año

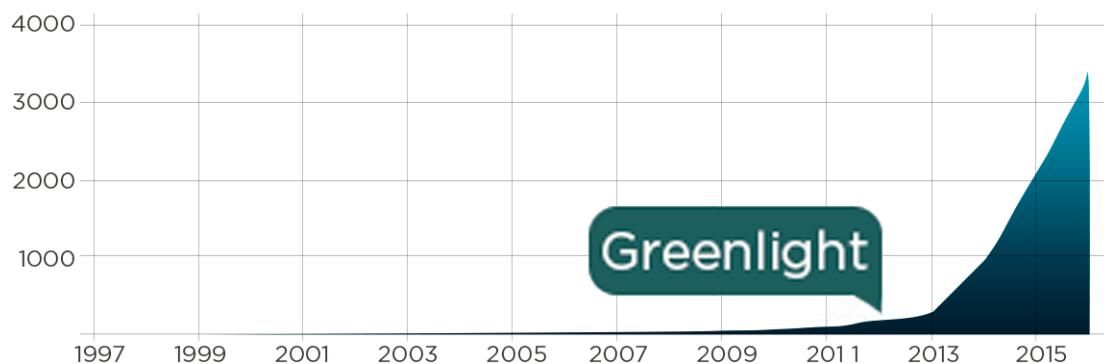


Figura 1.6: Relevancia del programa «Greenlight» en los lanzamientos indies [28]

Si bien se podría decir que nacieron en la clandestinidad, desde hace años gozan de una enorme aceptación y prueba de ello es que cada vez tienen mayor visibilidad: «Steam», la principal plataforma de distribución digital en PC, propiedad de «VALVe», apoya los videojuegos independientes gracias a su «Steam Greenlight», permitiendo a los usuarios de su plataforma valorar juegos desconocidos y, si la aceptación es buena, pasar a incluirlos en su catálogo. Sony con su videoconsola PlayStation y Microsoft con su homónima Xbox incorporan desde hace ya algunos años indies en su catálogo, y si bien en un principio eran bastante escasos es un hecho que cada vez se apuesta más por ellos y se fomenta la aparición de nuevos [4, 15]. Más recientemente ha sido Nintendo quien no ha podido dejar pasar el tren de los videojuegos indies incorporándolos al catálogo de lanzamiento de Nintendo Switch.

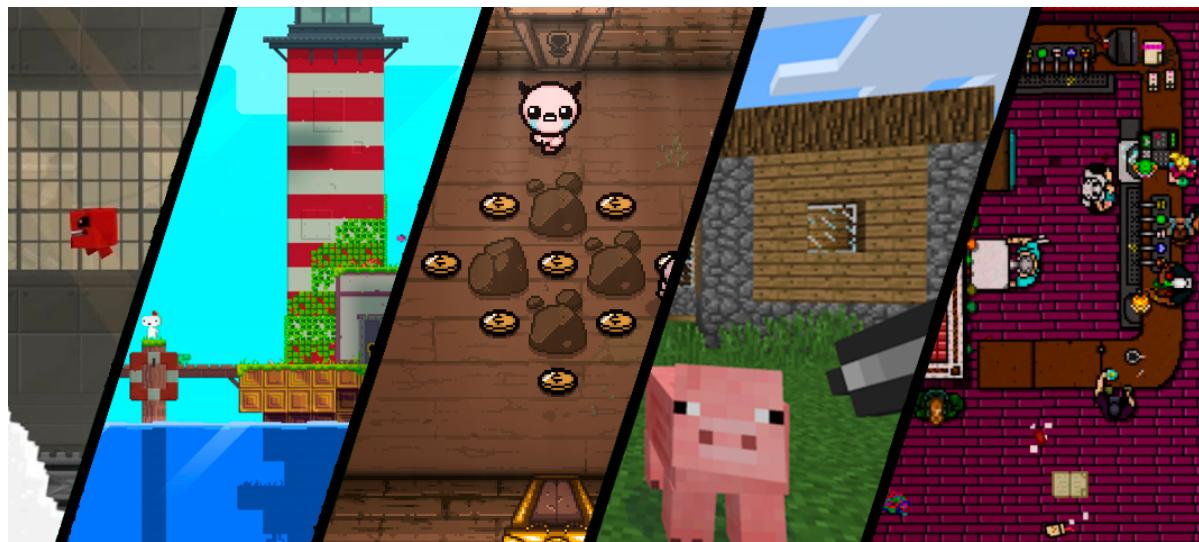


Figura 1.7: Ejemplos de videojuegos indies se hicieron un hueco en el mercado

1 Cabe remarcar que recientemente, desde Junio de 2017, el programa de «VALVe»
 2 para sus juegos independientes de «Steam» denominado «Steam Greenlight» no acepta
 3 nuevas aportaciones. Esto es porque la compañía está haciendo algunas modificaciones
 4 y el sistema pasará a denominarse «Steam Direct». El principal cambio será que los
 5 desarrolladores tendrán que abonar una fianza 100\$ por juego, en lugar de realizar un
 6 único pago como en «Greenlight» y que permitía publicar tantos juegos en el programa
 7 como se quisiese, y al recaudar 1000\$ en beneficios la fianza será devuelta. El principal
 8 motivo de esto es mejorar la calidad de los juegos que se incorporan finalmente a la
 9 plataforma al salir del programa de lanzamientos [22, 23].

10 En la actualidad, existen un gran número de indies que han saltado a la fama, el
 11 caso más conocido es el de «Minecraft» (2011), pero existen otros muchos títulos como
 12 «Super Meat Boy» (2010), «The Binding Of Isaac» (2011 y su remasterización, «Rebirth»,
 13 lanzada en 2014), «FEZ» (2012) o «Hotline Miami» (2012).

14 En conclusión, estamos en un momento en el que existe un público dentro de la
 15 comunidad que valora desde hace años el soplo de aire fresco que supone el contenido
 16 independiente en la industria, que permanecen atentos al panorama indie y no sólo
 17 a los lanzamientos triple A (o AAA), y en muchos casos valoran este tipo de conteni-
 18 do por encima de una superproducción que va dirigida a un público mucho menos
 19 específico.

1.1.3 La irrupción de los trofeos / logros y su impacto

Aunque en la década de los 90 ya aparecieron los primeros videojuegos con algún atisbo de sistema de desafíos no fue hasta hace muy poco, con la aparición de la consola «Xbox 360» y su sistema de logros (asociados a una «Gamertag»), que no se redefinieron hasta como los conocemos hoy en día: los logros son diseñados e implementados en cada juego por los desarrolladores, pero una vez que obtienen se quedan registrados en el perfil del usuario. Esto permite al jugador exhibir los logros que más le gusten, los más raros, mostrar el número total de ellos, el número de juegos con todos los logros desbloqueados, el porcentaje de obtención de logros medio, etc. En resumen, con esta renovación pasaron a tener un aspecto mucho más social.

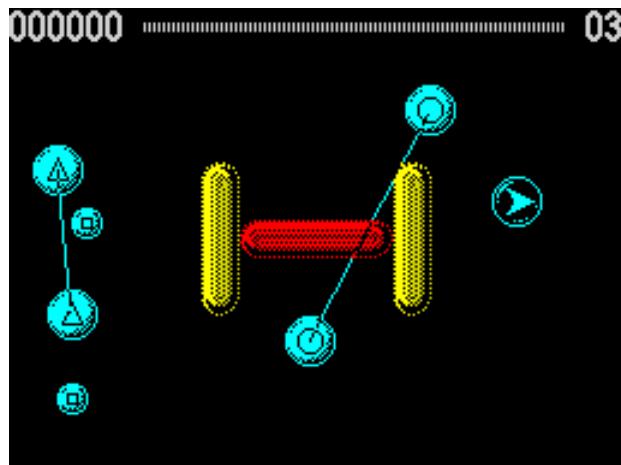


Figura 1.8: Captura del juego «E-Motion» (1990), para la videoconsola «Amiga», uno de los primeros juegos en implementar algún tipo de logro

Cabe destacar que su propósito inicial se ha ido desvirtualizando con el tiempo: un logro, como si nombre indica, originariamente era otorgado al jugador cuando este realizaba algún tipo de proeza en el juego o, por ejemplo, cuando completaba el mismo. Actualmente también se usan en muchos casos para mostrar la progresión del jugador pero por otra parte existen cada vez más títulos, conocidos como «achievement machines» (que podría traducirse como «máquinas de logros»), que recompensan excepcionalmente al jugador (por ejemplo, cada vez que realiza una acción básica como saltar, disparar, o bien múltiples logros por completar la misma acción) [20] y además permiten completar el juego en cuestión de minutos. Aunque en primera instancia pudiera parecer lo contrario, este tipo de juegos, por el mero hecho de permitirte obtener un gran número de logros en poco tiempo o logros que te permitan decorar el perfil, tiene un público y va creciendo [12].

A día de hoy existen numerosas plataformas destinadas a los «achievement hunters» o «cazadores de logros» (como pueden ser «AStats» o «AchievementStats» en «Steam»), que elaboran numerosas estadísticas con los logros y confeccionan un ranking de jugadores atendiendo a los mismos, y otras muchas comunidades con guías o tutoriales para obtener logros específicos (como podrían ser «XBoxAchievements» en «Xbox» o «PlayStationTrophies» en «PlayStation»).



Figura 1.9: Imagen del sistema de logros interno de «Minecraft» (2011), que se mantuvo hasta la versión 1.11

En definitiva, en la actualidad los logros tienen un peso importante para un sector considerable de los jugadores, de tal manera que algunos incluso dejan de comprar un título por carecer de logros o retrasan la compra hasta que incorporen los mismos.

1.2 ESTADO DEL ARTE

La industria de los videojuegos, aunque es difícil determinar su origen exacto (los primeros videojuegos surgieron tras la Segunda Guerra Mundial, pero a un ritmo lento y no son más que los albores del sector que conocemos hoy en día), es una industria joven. A pesar de esta juventud, y poniéndonos en antecedentes, es un sector que ya ha pasado por una grave crisis a principios de los ochenta y que lo llevó al borde de su extinción. Esta crisis fue motivada, principalmente, por el descontrol que originó la coexistencia en el mercado de demasiadas consolas y, sobre todo, el lanzamiento al mercado de un gran número de videojuegos de baja calidad, sin ningún tipo de control, mermando la confianza del usuario y haciendo descender las cifras de ventas.



Figura 1.10: Imagen del juego «E.T.» para la «Atari 2600», uno de los máximos detonantes de la crisis del videojuego [21]

La crisis se extendió hasta mediado de los ochenta y, desde entonces, la industria goza de salud y ha ido en crecimiento sin atravesar de nuevo ningú valle, por el contrario, es un sector que desde aquel momento ha ido en una clara tendencia positiva y que actualmente sigue en pleno auge.

Además, debemos hacer mención que en España justo después de la crisis se entró en un periodo denominado «la edad de oro del software español», que se extendió entre 1983 y 1992, en la que el país se convirtió en uno de los principales creadores de videojuegos de Europa.



Figura 1.11: Fotograma de «*La abadía del crimen*» (1987), uno de los máximos estandartes de «la edad de oro del software español»

1 Volviendo a nuestros días y centrándonos en lo que la industria supone en nuestro
 2 país, cada vez las universidades y academias ofertan más titulaciones o cursos rela-
 3 cionados con los videojuegos, y esto ayuda a que poco a poco el tejido empresarial
 4 relacionado con la industria del videojuego vaya creciendo [26]. La existencia de poco
 5 tejido empresarial no ha impedido que en el pasado, durante la corta historia de la in-
 6 dustria, en nuestro país hayan aparecido títulos de primer orden a nivel internacional:
 7 el caso más conocido es el de la saga «Commandos», que hizo su aparición a finales de
 8 los noventa y, más recientemente, un título que podríamos catalogar de súper produc-
 9 ción, «Castlevania: Lord of the Shadows» (2010).

10 Aunque hay excepciones, la notable la mayoría de los juegos actuales son realizados
 11 a partir de un motor gráfico comercial ya definido. Estos motores suelen ser propiedad
 12 de la compañía, o bien de terceros bajo acuerdo, pero en los últimos años han surgi-
 13 do varios motores de uso gratuito que han ganado enorme popularidad y que hacen
 14 más accesible el desarrollo de videojuegos, entre los que encontramos Unreal Engine
 15 4, CryEngine 3 o Unity.

Tomando como ejemplo Unreal Engine, que es el motor que se usará para llevar a cabo este proyecto, es una potente herramienta que más allá de satisfacer las necesidades básicas del desarrollador (renderizado 2D/3D, detección de colisiones, texturizado de objetos, sistema de luces, etc.) incorpora importantes herramientas en su haber tales como «Persona», que permite al desarrollador crear o realizar cambios en animaciones o «UMG», que permite crear una interfaz de usuario de una manera sencilla dentro del motor. Además, permite utilizar tanto C++ como programación gráfica (a lo que se suele hacer referencia como «Blueprints») siendo la segunda forma unas 10 veces peor (de media) en rendimiento que la primera, según la propia Epic Games, desarrolladores de Unreal Engine [6].

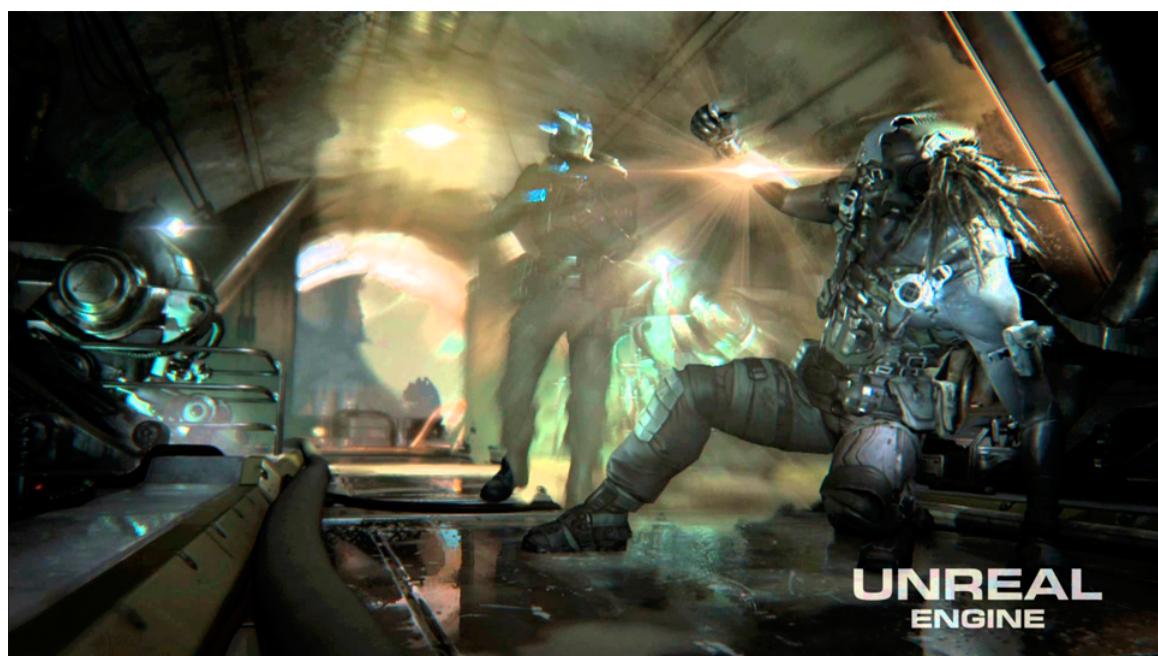


Figura 1.12: Imagen de la primera demostración técnica en tiempo real de Unreal Engine 4, titulada «Infiltrator»

Podemos concluir que la industria pasa por un buen momento, está en continuo crecimiento y no hay ningún indicio que nos haga pensar que se pueda volver a repetir una crisis como la que se originó en los ochenta, y mucho menos por la misma razón: la desinformación, premisa que es impensable que se pueda volver a dar en nuestros días. Por otro lado, los motores gráficos actuales son un avanzado conjunto de herramientas que permiten a los desarrolladores crear sus productos de una manera más sencilla y rápida que antaño, lo que repercute en los tiempos de ejecución y permite abordar cada vez proyectos más extensos o ambiciosos influyendo, por lo tanto, de manera directa en el acabado final del producto conforme van evolucionando.

OBJETIVOS DEL PROYECTO

1

2 **L**a siguiente sección presenta más detalladamente el proyecto, la razón de ser del mismo
3 y un listado de objetivos, tanto del producto como individuales, que se pretenden
4 alcanzar en el transcurso de su realización.

2.1 MOTIVACIÓN

Dada la saturación del mercado con productos similares o, directamente, clónicos se pretende crear un videojuego que, siguiendo la filosofía indie que se describía previamente, consiga ser innovador mezclando géneros, que se salga del esquema predominante y rete al jugador siendo desafiante con sus mecánicas y haciéndole pensar, que haga uso de la principalmente primera persona para tener la posibilidad de aprovechar el auge de la tecnología de realidad virtual y que, premeditadamente, deje bastante de lado la historia, personajes o desarrollo de los mismos y, por tanto, se centre plenamente en la jugabilidad. El jugador deberá ir haciendo uso de todas las mecánicas de las que dispone para sortear obstáculos, esquivar trampas, solucionar puzzles y hallar el camino correcto para ir avanzando en el juego.



Figura 2.1: Fotograma de «Bioshock» (2007), uno de los pocos «First Person Shooter» («FPS») que se supieron diferenciar dentro de la saturación del género

El gameplay se basaría en la repetición y memorización de niveles, esquive y timing, así como en la gestión de adrenalina y, en menor medida, en la resolución de pequeños puzzles y exploración:

Repetición y memorización de niveles: Los peligros del entorno (balas, flechas, láseres, etc. y, por supuesto, caídas al vacío) matarían instantáneamente al jugador, lo que le obligaría a empezar el nivel desde el último punto de control o, en el caso de no disponer de ninguno, desde cero.

¹ **Esquive y timing:** No se le daría la posibilidad al jugador de defenderse o eliminar peligros, por lo que el jugador deberá ir avanzando por el escenario haciendo uso de las mecánicas para sortear peligros en el momento justo sin morir en el intento.

⁵ **Gestión de adrenalina:** Cada acción que realice el jugador tendrá un peso asociado, por tanto, en todo momento el jugador deberá vigilar la barra de adrenalina y, así pues, pensar antes de realizar una acción.

⁸ **Resolución de puzzles:** A menudo el jugador deberá encontrar un botón / palanca / llave para avanzar o bien tendrá que encontrar un orbe dorado para acceder a una determinada zona que antes era inaccesible.

¹¹ **Exploración:** No sólo deberá encontrar el camino correcto, o la forma de desbloquear o pasar por un camino, sino que por el mundo habrá atajos que interconecten el mapa con antiguos puntos de control, así como colecciónables repartidos por el mismo.

¹⁵ Podríamos concluir que, principalmente, mezcla los géneros de primera persona, ¹⁶ aventura / exploración, plataformas, bullet hell y puzzle.

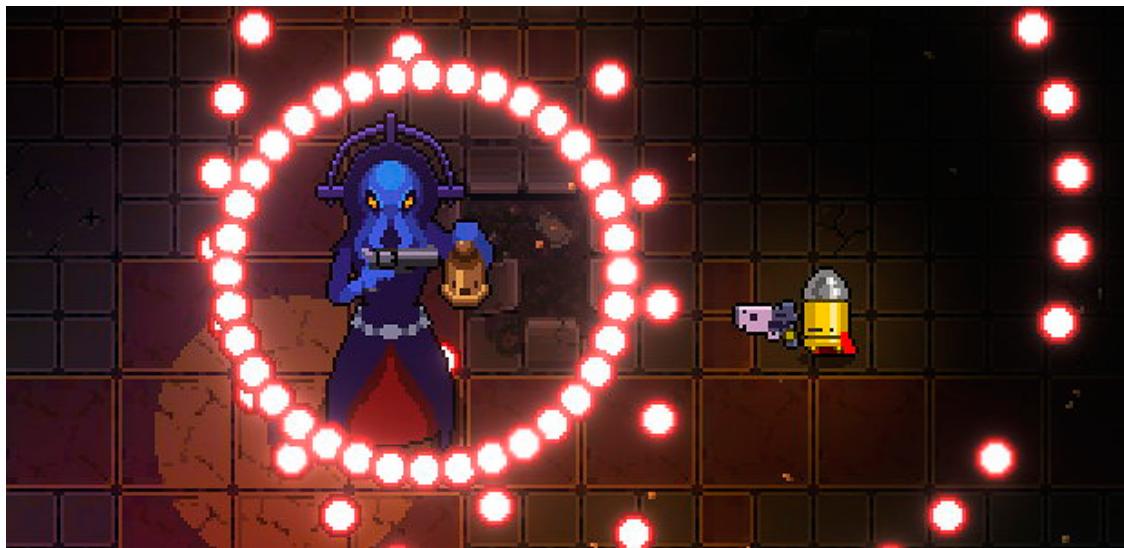


Figura 2.2: Captura de «Enter the gungeon» (2016), ejemplo de la forma más tradicional de bullet hell en dos dimensiones

¹⁷ Teniendo en cuenta lo descrito anteriormente sería un producto que no va dirigido al gran público, sino que el público objetivo sería más bien un público de nicho, que ¹⁹ busque algo diferente e intente poner a prueba sus capacidades.

2.2 LISTADO DE OBJETIVOS

Para una mayor claridad se ha decidido dividir esta sección en dos partes, los objetivos que se pretenden alcanzar individualmente y los objetivos que se pretende que alcance, una vez finalizado, el producto:

2.2.1 Objetivos del producto

Sistema de plataformeo: Realización de un sistema de mecánicas básicas de plataformeo como saltar, agacharse, correr, manejar al personaje en el aire, etc. teniendo en cuenta que, sobre todo ésta última, sean amigables con el usuario y permitan llevarlas al límite sin frustración por parte del usuario.

Sistema de escalada: Elaboración de un sistema de escalada que maximice la interacción del personaje con el entorno y permita acciones tales como escalar paredes, desplazarse verticalmente o dejarse caer entre cornisas, moverse por ellas, saltar de una a otra, etc.

Sistema de poderes: Producción de un sistema de poderes coherente, en el que el personaje pueda ganar poderes tales como el de la teletransportación sin que entre en conflicto con las mecánicas, más básicas, de plataformeo y escalada, es decir, que sólo los pueda usar durante un área determinada para poder avanzar en el juego pero que al abandonarla vuelva a tener que hacer uso de las mecánicas de plataformeo y escalada.

Elaboración del tutorial: Elaborar un nivel de tutorial en el que se le presenten al jugador las mecánicas básicas.

Elaboración de niveles prototipo: Realización uno o una serie de niveles prototipo (dependiendo de la longitud de los mismos) de lo que sería un escenario del juego real.

Sistema de animaciones: Creación de una máquina de estados que permita pasar de una animación a otra sin bugs y de una manera sutil, sin saltos entre animaciones.

Bullet hell en 3D: Conseguir adaptar el género del bullet hell a un entorno 3D de manera adecuada, ya que este género es típico de entornos 2D donde, al eliminar toda una dimensión, es mucho más sencillo de implementar porque se reducen drásticamente los puntos en los que los peligros y el jugador pueden coincidir.

1 En otras palabras, llenar el entorno de peligros en un escenario más amplio es
2 una tarea mucho más difícil de diseñar.

3 **Realidad virtual:** Diseñar el juego de manera que fuese amigable con la innovadora
4 tecnología de realidad virtual, es decir, enfocarlo hacia ella y que pudiese ser
5 utilizado con ésta sin problemas (sin prescindir del teclado / ratón o mando).

6 **Sistema de guardado / cargado:** El producto debe ser capaz de guardar y cargar los
7 avances del jugador cuando así se requiera. En concreto se hará uso del autoguardado
8 cuando el jugador entre en contacto con la superficie asociada a un punto
9 de control.

10 **Sistema interno de desafíos / logros:** Se guardará también el progreso del jugador en
11 materia de desafíos en su archivo de guardado. Esto es independiente de cual-
12 quier sistema de logros que se pudiesen implementar en el futuro en el juego
13 (logros de Steam, trofeos de PlayStation, logros de Xbox, etc.), pero serían los
14 mismos desafíos por lo que sólo faltaría la comunicación con el sistema de logros
15 externo pertinente. Un sistema como éste es el que usan actualmente muchos
16 títulos del mercado, como «The Binding of Isaac: Rebirth» o «Payday: The Heist»:
17 cuando el usuario alcanza un requisito para un logro se marca el desafío como
18 completado internamente (archivo de guardado) y a la vez lo comunican al ser-
19 vor de logros pero, si esto por algún motivo falla, lo comunican de nuevo al
20 iniciar otra vez el juego.

21 **Optimización:** Optimizar lo máximo posible realizando todo el contenido posible en
22 C++ ya que como explicaba anteriormente (tomando la propia Epic Games como
23 fuente, desarrolladora de Unreal Engine), los tiempos de ejecución de ese código
24 se reducen en torno a 10 veces usando el lenguaje de programación en C++ sobre
25 la implementación en programación gráfica [6].

2.2.2 Objetivos individuales

- Uno de los objetivos principales que me he marcado es simplemente aprender y disfrutar, conocer tanto los aspectos organizativos como los de desarrollo que envuelven al mundo de la creación de videojuegos, puesto que es algo que siempre me ha parecido muy interesante. Es un objetivo muy fácil de satisfacer, y no podría decir que cumpliendo sólo este ya estuviese satisfecho, pero sin duda me es indispensable.
- Aprender a utilizar de una forma fluida el entorno y el conjunto de herramientas que conforman Unreal Engine.
- Ampliar mis conocimientos en el lenguaje de programación C++, aprovechando que es el lenguaje en el cual se programa Unreal Engine.
- Adquirir conocimientos en programación visual, ya que es un aspecto que en la titulación se ha abordado de manera muy escueta y habiendo elegido Unreal Engine para la ejecución del proyecto es algo que voy a necesitar en algún momento del desarrollo.
- Formarme en diversas aptitudes que no entran dentro del ámbito de la titulación de Ingeniería del Software, como pueden ser la creación de entornos 3D, la texturización e iluminación de los mismos, creación de landscapes, la animación de personajes, la creación de interfaces, diseño de niveles, etc.

PARTE II

ORGANIZACIÓN

DEL PROYECTO

METODOLOGÍA

1

2 *continuación se presenta la estructura organizacional del proyecto, la metodología
3 que se ha elegido seguir para el desarrollo de este software, un breve resumen de la
4 misma y la forma en la que se va a adaptar a nuestro problema específico.*

A

3.1 ESTRUCTURA ORGANIZACIONAL DEL PROYECTO

Dado que el proyecto se realiza de forma individual no se podría decir que existe una estructura organizacional interna propiamente dicha, o al menos carecería de sentido establecer una estructura organizacional para una sola persona, puesto que todas las responsabilidades y roles presentes en el proyecto recaerían sobre una única persona, salvando la figura del tutor.

A raíz de lo descrito surge una consecuencia directa, y es que algunos roles o actividades de la metodología desaparecerían del proyecto, todos los que tengan que ver con la interacción entre dos o más personas, al carecer de sentido en un proyecto con un único miembro en él. Ejemplo de esto podrían ser todas aquellas actividades que tengan que ver con la sincronización / puesta en común con el resto de miembros del equipo o cualquier rol relacionado con la resolución de conflictos dentro del equipo del proyecto.

Sin embargo, como se verá en las posteriores secciones del capítulo, se desempeñarán a la vez tres roles: «Jefe de proyecto», «Diseñador» y «Programador», ya que aunque el equipo de proyecto esté formado por una única persona se intentarán simular estos tres roles.

Teniendo en cuenta lo citado, el organigrama resultante y que se usará en la realización del proyecto es el siguiente:



Figura 3.1: Organigrama del proyecto

3.2 METODOLOGÍA DE DESARROLLO

3.2.1 Presentación de la metodología

Para la realización de este proyecto se hará uso de la metodología «Feature-Driven Development» (FDD), metodología denominada «Desarrollo basado en funcionalidades» en español.

Fue creada por Jeff De Luca y Peter Coad a finales del siglo XX para salvar un importante proyecto que había sido declarado como irrealizable y tiene su razón de ser en la calidad y el monitoreo constante del proyecto [7, 24].

3.2.2 Resumen de la metodología

FDD es una metodología englobada en el grupo de las denominadas ágiles, constituye un proceso iterativo e incremental de desarrollo software y consta de 5 partes [1, 2]:

1. **Desarrollo del modelo global:** La realización de un proyecto siguiendo el esquema dictado por FDD empieza por la realización de un modelo global de alto nivel que tiene en cuenta el contexto y el alcance del sistema. El modelo se subdivide en partes más pequeñas que se van completando y se confecciona un diagrama de clases por cada una. Finalmente cada una de las partes en las que se subdividió se van agrupando para formar el modelo global final.
2. **Elaboración de la lista de funcionalidades:** Seguidamente, utilizando el conocimiento obtenido durante el desarrollo del modelo global, se confecciona una lista de funcionalidades que posteriormente se dividirán en funcionalidades más específicas.
3. **Planificación por funcionalidad:** Nuevamente partiendo del punto anterior, tomamos la lista de funcionalidades y esta vez las ordenamos según su prioridad y teniendo en cuenta su dependencia.
4. **Diseño por funcionalidad:** Se elige un conjunto de funcionalidades de la lista y se procede a diseñarlas y desarrollarlas mediante un proceso iterativo.
5. **Construcción por funcionalidad:** Despues de una fase de diseño satisfactoria se procede a la construcción total del proyecto.



Figura 3.2: Esquema del desarrollo basado en funcionalidades

Además, la metodología FDD consta de 6 **roles clave**, los cuales se describen a continuación [2, 25]:

1. **Jefe de proyecto:** Es el responsable de gestionar el presupuesto, el tiempo, el espacio y los recursos del proyecto. También es el responsable de transmitir el progreso del proyecto a los altos cargos de la empresa.
2. **Arquitecto jefe:** Es el responsable del diseño global del sistema. Tiene la última palabra en todas las cuestiones de diseño.
3. **Jefe de desarrollo:** Es el responsable del desarrollo en el día a día, de evitar situaciones de bloqueo y de solucionar conflictos en el proyecto.
4. **Programadores jefes:** Son desarrolladores experimentados que se encargan de diseñar los requisitos a alto nivel y trabajan junto a otros programadores jefes para resolver las dificultades a las que se enfrenta el proyecto día a día.
5. **Encargados de clases:** Desarrolladores que junto a su pequeño equipo de trabajo y bajo la supervisión de un programador jefe diseñan, programan, prueban y documentan las funcionalidades que se implementan en el sistema.
6. **Expertos de dominio:** Son los encargados, haciendo uso de su conocimiento del negocio, de detallarle minuciosamente a los desarrolladores las características que debe tener el producto.

1 3.2.3 Adaptación de la metodología

2 Teniendo en consideración que FDD es una metodología pensada para grupos de
3 personas de un tamaño considerable, grupos en los que incluso otras metodologías
4 ágiles como SCRUM no tendrían cabida al ser imposible realizar una autogestión [1],
5 y enfocada al desarrollo de un software más convencional, donde se podría diseñar
6 un diagrama de clases con facilidad (como podrían ser las aplicaciones de escritorio,
7 móvil, web, etc.) y por contrapartida este proyecto será llevado a cabo por una única
8 persona y el producto generado en su desarrollo será un videojuego, es necesario hacer
9 algunas modificaciones en nuestra metodología:

- 10 1. **Fase de diseño del videojuego:** La principal diferencia radica aquí puesto que, dada la naturaleza del proyecto, al no poder organizar un diagrama de clases tenemos que buscar alternativas. En este caso se ha optado por elaborar una documentación con las decisiones de diseño del videojuego, esto vendría a coincidir con el conjunto de las mecánicas que va a contener, las características que incluiría el videojuego y las distintas singularidades que conformarían el mismo. En la industria se denomina comúnmente «Game Design Document» («Documento de diseño del videojuego» en español) y se podrá consultar en el anexo de este documento.
- 19 2. **Elaboración de la lista de funcionalidades:** Al igual que en la metodología FDD tradicional, se elabora una lista de funcionalidades a partir del paso anterior y éstas se desglosan a su vez en funcionalidades más específicas.
- 22 3. **Planificación por funcionalidad:** Nuevamente como en la metodología FDD tradicional, ordenamos las funcionalidades resultantes según su prioridad y teniendo en cuenta sus dependencias.
- 25 4. **Diseño por funcionalidad:** Otra gran diferencia sería que, puesto que el grupo de desarrollo está formado por una sola persona, se irían eligiendo funcionalidades de la lista de una en una y se diseñaría e implementaría.
- 28 5. **Construcción por funcionalidad:** Igual que en la metodología FDD se procedería a la construcción total del proyecto.



Figura 3.3: Esquema del desarrollo basado en funcionalidades adaptado para el proyecto

La **documentación** se generaría de forma constante durante todo el transcurso del proyecto y lo haría de forma iterativa, tanto cada vez que se empezara a diseñar una funcionalidad como cuando se terminase el diseño de la misma.

Durante el trascurso del proyecto se utilizarán dos **repositorios de código**, uno para gestionar la documentación (en LaTeX) y otro para almacenar el código del producto en sí.

Para finalizar, los **roles** también necesitan adaptarse al proyecto, ya que muchos de ellos dejan de tener sentido por las mismas razones que se citaban anteriormente. En este caso es necesario una simplificación, que llevaría a que los roles de programador se aglutinaran en uno solo y lo mismo pasaría con los roles de diseño, lo que daría lugar a:

- **Jefe de proyecto:** Es el responsable de gestionar el presupuesto, el tiempo, el espacio y los recursos del proyecto. En el caso que nos ocupa sería el encargado de la elaboración de la memoria en términos generales.
- **Diseñador:** Es el responsable del diseño global del sistema y de la elicitation de requisitos tanto a alto nivel como de posteriormente detallarlos. En resumidas cuentas, sería un analista pero esta vez centrado en el mundo del videojuego.
- **Programador:** Diseña, programa, prueba y documenta las funcionalidades que se implementan en el sistema.

PLANIFICACIÓN

1

2 *n la siguiente sección se expone la planificación del proyecto y el informe de tiem-
3 pos generado durante el transcurso del mismo que, comparado con lo anterior, nos
4 permitirá ver la desviación en cuanto a tiempo del proyecto.*

4.1 RESUMEN TEMPORAL DEL PROYECTO

1

Resumen del proyecto	
Fecha de inicio	20/02/2017
Fecha de fin	01/09/2017
Periodicidad de las revisiones (promedio)	2 semanas
Carga de trabajo semanal (promedio)	16,31 horas
Horas totales previstas	310 horas
Horas finales	341,5 horas

Cuadro 4.1: Tabla resumen de tiempos y planificación

4.2 PLANIFICACIÓN INICIAL

2

Para su planificación, se ha decidido dividir el proyecto en siguientes fases o iteraciones, las cuales se describen a continuación junto con los roles que participan principalmente:

3

4

5

- **Fase de estudio:** En esta primera etapa interviene únicamente el rol **jefe de proyecto** (aunque en un contexto real intervendrían más personas, como por ejemplo el diseñador jefe), que será el encargado de estudiar las diferentes tecnologías que tiene a su disposición y de poner los cimientos más básicos de la ruta que seguirá el proyecto.
- **Fase de inicio del proyecto (o iteración 0):** Es la etapa en la que se realizan las siguientes tareas:
 - Las primeras fases de la metodología y buena parte de la elaboración de la memoria, salvando los apartados finales (por ejemplo, el manual o las conclusiones) y las secciones centradas en el desarrollo, principalmente.
 - El diseño general del videojuego y, por tanto, el documento de diseño del videojuego.
 - La inicialización del proyecto, preparación del repositorio, etc.

6

7

8

9

10

11

12

13

14

15

16

17

18

Teniendo en cuenta lo descrito, en esta fase intervienen el **jefe de proyecto** (documentación del proyecto), el **diseñador** (realización del «Documento de diseño del juego» o «GDD») y el **programador** (inicialización del proyecto).

19

20

21

1 ■ **Iteraciones 1-6:** Son las etapas más puramente de desarrollo. Intervienen tanto el
2 **diseñador**, que será el encargado de preparar las iteraciones, como el **programa-**
3 **dor**, que será quien codifique y genere documentación de las iteraciones (como ya
4 se explicó en la sección §3.2.3), además del **jefe de proyecto** que tendrá una labor
5 de supervisión (principalmente gestionar tiempos empleados y retrasos). A con-
6 tinuación, se describen muy escuetamente las iteraciones (para más información,
7 consultar el documento de diseño §B):

- 8 • **Iteración 1. Mecánicas básicas y poderes:** La primera iteración se imple-
9 mentarán las mecánicas más básicas, como saltar, agacharse, correr, etc. y,
10 además, las mecánicas de las que podrá hacer uso el personaje cuando esté
11 potenciado.
- 12 • **Iteración 2. Mecánicas de escalada:** En la segunda iteración se implemen-
13 tarán las mecánicas de «plataformeo» avanzadas.
- 14 • **Iteración 3. HUD y menú:** Durante la tercera iteración se creará el menú
15 principal y el HUD del que dispondrá el usuario.
- 16 • **Iteración 4. Peligros del entorno:** La cuarta iteración tratará sobre la imple-
17 mentación los peligros del entorno que deberá esquivar el personaje.
- 18 • **Iteración 5. Inteligencia artificial y sistema de guardado:** Durante la quinta
19 iteración se implementará la IA de los enemigos y el sistema de guardado.
- 20 • **Iteración 6. Diseño de escenarios:** En la sexta y última iteración se llevarán a
21 cabo la creación de los escenarios de los que dispondrá, a modo de demo-
22 stración, el videojuego a entregar. Esto incluye el diseño de un tutorial.

23 ■ **Fase de cierre del proyecto:** En esta iteración intervienen el **diseñador**, encargado
24 de elaborar el manual de la aplicación, y el **jefe de proyecto**, que será quien fina-
25 lice los últimos apartados de la memoria, haga las últimas modificaciones sobre
26 la misma y posteriormente realice la presentación mediante la cual se expondrán
27 los resultados del proyecto.

A tenor de lo descrito y de la metodología establecida en el capítulo anterior, podemos establecer la siguiente relación:

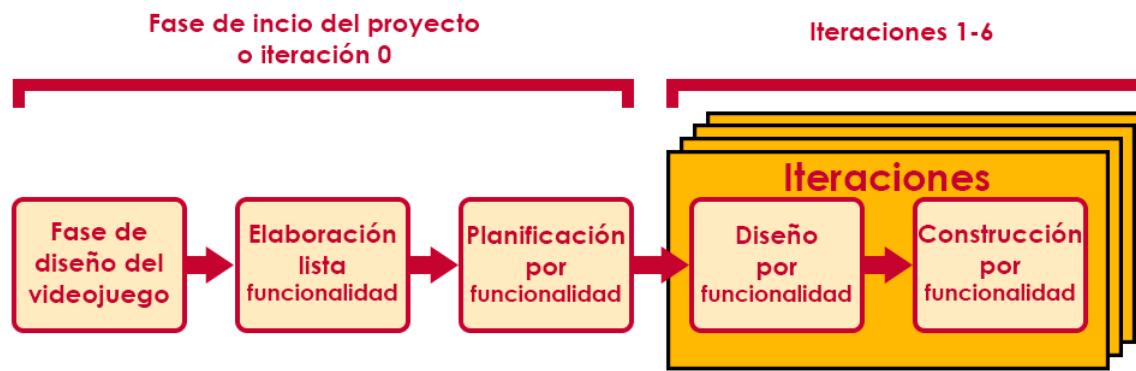


Figura 4.1: Equivalencias fases-metodología

Además, analizando la diferentes fases **desde el punto de vista de los roles** podemos concluir que:

- **Jefe de proyecto:** Realiza la mayor parte del esfuerzo en las primeras etapas del proyecto (fase de estudio y fase de inicio) y, en menor medida, al final del mismo (fase de cierre). Entre dichas fases, en las iteraciones, tiene un papel mucho más testimonial en el proyecto que nos ocupa.
- **Diseñador:** La mayor parte del esfuerzo en este caso recae en la fase de inicio del proyecto, donde se tienen que sentar las bases del juego y diseñar el documento de diseño del juego. Durante las iteraciones tiene un papel mucho menos relevante que el programador pero más que el jefe de proyecto y, al igual que pasaba con éste último, también crece su esfuerzo al final del proyecto (fase de cierre).
- **Programador:** Únicamente concentra su esfuerzo durante las iteraciones, durante el resto del proyecto no está presente en el escenario planteado.

¹ La siguiente tabla muestra un resumen de la planificación proyectada para las ite-
² raciones junto con su fecha de comienzo y fin:

Resumen de fases e iteraciones (planificación)	
Fase de estudio	20/02/17 a 12/03/17 (3 semanas)
Fase de inicio del proyecto	13/03/17 a 02/04/17 (3 semanas)
Iteración 1	03/04/17 a 16/04/17 (2 semanas)
Iteración 2	24/04/17 a 14/05/17 (3 semanas)
Iteración 3	15/05/17 a 28/05/17 (2 semanas)
Iteración 4	03/07/17 a 09/07/17 (1 semana)
Iteración 5	10/07/17 a 23/07/17 (2 semanas)
Iteración 6	24/07/17 a 06/08/17 (2 semanas)
Fase de cierre del proyecto	07/08/17 a 13/08/17 (1 semana)
Red de seguridad	14/08/17 a 31/08/17 (2,5 semanas)

Cuadro 4.2: Planificación temporal de fases e iteraciones (planificación)

³ Como podemos extraer de la tabla, durante la ejecución del proyecto hay plani-
⁴ ficadas dos paradas, la primera es de sólo una semana y la segunda de un mes. Se
⁵ describen a continuación:

- ⁶ **■ Semana del 17 al 23 de Abril:** Esta semana es coincidente con los primeros par-
⁷ ciales.
- ⁸ **■ Mes de Junio (semanas entre el 29 de Mayo y el 02 de Julio):** Coincidente con los
⁹ segundos parciales, exámenes finales y entrega y posterior defensa de proyectos
¹⁰ de diversas asignaturas.

¹¹ Además, como también figura en la tabla, se ha decidido establecer un periodo
¹² denominado «**Red de seguridad**» a modo de medida de contingencia para paliar cual-
¹³quier retraso que surja durante la ejecución del proyecto. Esto es una medida reco-
¹⁴mendable en todos los proyectos pero más antes el que nos encontramos, ya que es
¹⁵el primer proyecto de este tipo / tecnología. En concreto tendremos un colchón de
¹⁶seguridad de dos semanas y media.

A continuación se muestra una tabla detallada de las iteraciones, mostrando qué rol interviene, el número de horas de trabajo que participa y el número de horas totales de cada iteración:

Planificación de fases e iteraciones detalladas			
Iteración	Roles implicados	Horas est. / rol	Horas estimadas
Fase de estudio	Jefe de proyecto	30 horas	30 horas
	Jefe de proyecto	30 horas	
Fase inicio proyecto	Diseñador	20 horas	60 horas
	Programador	10 horas	
Iteración 1	Jefe de proyecto	4 horas	
	Diseñador	6 horas	34 horas
	Programador	24 horas	
Iteración 2	Jefe de proyecto	6 horas	
	Diseñador	9 horas	51 horas
	Programador	36 horas	
Iteración 3	Jefe de proyecto	4 horas	
	Diseñador	6 horas	34 horas
	Programador	24 horas	
Iteración 4	Jefe de proyecto	2 horas	
	Diseñador	4 horas	18 horas
	Programador	12 horas	
Iteración 5	Jefe de proyecto	4 horas	
	Diseñador	6 horas	34 horas
	Programador	24 horas	
Iteración 6	Jefe de proyecto	4 horas	
	Diseñador	6 horas	34 horas
	Programador	24 horas	
Fase cierre proyecto	Jefe de proyecto	10 horas	
	Diseñador	5 horas	15 horas
TOTAL			310 horas

Cuadro 4.3: Planificación temporal detallada de fases e iteraciones

Recordemos que es un proyecto realizado con una tecnología, que genera un producto nuevo y del que, por tanto, no se tienen referencias previas por lo que estimar las horas es especialmente difícil. Respecto a esto no hay mucho que podamos hacer, tan solo se han podido calcular las estimaciones sobre las horas dedicadas de cada rol durante las **fases de iteración** en base a las siguientes normas, para conseguir que sean homogéneas:

Calculo de horas por rol en iteraciones	
Rol	Horas estimadas
Jefe de proyecto	2 horas por semana
Diseñador	3 horas por semana
Programador	12 horas por semana

Cuadro 4.4: Tabla del calculo de horas por rol en iteraciones

Por último, de la tabla de planificación detallada podemos extraer el número total de horas de trabajo de cada rol, lo cuál nos será útil en el próximo capítulo cuando elaboraremos la documentación relativa a los costes de personal (ver sección §5.2):

Resumen de horas de trabajo por rol	
Jefe de proyecto	94 horas
Diseñador	62 horas
Programador	154 horas

Cuadro 4.5: Tabla resumen de horas de trabajo por rol

4.3 INFORME DE TIEMPOS DEL PROYECTO

Durante el desarrollo del proyecto se han obtenido los siguientes resultados respecto a tiempo:

Tiempos empleados en fases e iteraciones detallados			
Iteración	Roles implicados	Horas reales / rol	Horas reales
Fase de estudio	Jefe de proyecto	23,2 horas	23,2 horas
	Jefe de proyecto	48,2 horas	
Fase inicio proyecto	Diseñador	21,6 horas	75,3 horas
	Programador	5,5 horas	
Iteración 1	Jefe de proyecto	2,8 horas	
	Diseñador	6,7 horas	26,8 horas
	Programador	17,3 horas	
Iteración 2	Jefe de proyecto	4,3 horas	
	Diseñador	9,8 horas	58,8 horas
	Programador	44,7 horas	
Iteración 3	Jefe de proyecto	3,7 horas	
	Diseñador	7,1 horas	32,9 horas
	Programador	22,1 horas	
Iteración 4	Jefe de proyecto	1,9 horas	
	Diseñador	5,4 horas	25,3 horas
	Programador	18 horas	
Iteración 5	Jefe de proyecto	2,6 horas	
	Diseñador	6,6 horas	31,1 horas
	Programador	21,9 horas	
Iteración 6	Jefe de proyecto	2,2 horas	
	Diseñador	12,2 horas	46,2 horas
	Programador	31,8 horas	
Fase cierre proyecto	Jefe de proyecto	18,1 horas	
	Diseñador	3,8 horas	21,9 horas
TOTAL			341,5 horas

Cuadro 4.6: Tabla de tiempos empleados en fases e iteraciones detallados

4.3. INFORME DE TIEMPOS DEL PROYECTO

La tabla que se muestra a continuación muestra la **desviación de tiempo** por iteración que se ha obtenido (en la cual el símbolo «+» representará una desviación de tiempo negativa, es decir, que la fase ha llevado más tiempo del planeado y por contrapartida el símbolo «-» representará que se ha realizado en menos tiempo del previsto):

Desviación de tiempo por iteración		
Fase	Desviación de la iteración	Desviación acumulada
Fase de estudio	-6,8 horas	-6,8 horas
Fase de inicio del proyecto	+15,3 horas	+8,5 horas
Iteración 1	-7,2 horas	+1,3 horas
Iteración 2	+7,8 horas	+9,1 horas
Iteración 3	-1,1 horas	+8 horas
Iteración 4	+7,3 horas	+15,3 horas
Iteración 5	-2,9 horas	+12,4 horas
Iteración 6	+12,2 horas	+24,6 horas
Fase de cierre del proyecto	+6,9 horas	+31,5 horas
Desviación final		+31,5 horas

Cuadro 4.7: Tabla de desviación de tiempo por iteración

Asimismo, vamos a recopilar la desviación de tiempo por rol, que nos será útil para sacar conclusiones una vez finalizado el proyecto.

Desviación de tiempo por rol			
Rol	Horas planeadas	Horas reales	Desviación final
Jefe de proyecto	94 horas	107 horas	+13 horas
Diseñador	62 horas	73,2 horas	+11,2 horas
Programador	154 horas	161,3 horas	+7,3 horas

Cuadro 4.8: Tabla de desviación de tiempo por rol

La siguiente tabla muestra, por períodos, el **retraso acumulado** del proyecto:

Retrasos del proyecto		
Período	Retraso / adelanto acumulado	Justificación
20/02/17 a 20/05/17	Periodo sin retrasos	-
21/05/17 a 13/08/17	1 semana de retraso	Mala planificación de la segunda parada. El retraso además se extiende más tiempo del deseado porque ocurre antes de la citada parada.
13/08/17 a Final	2 semanas de retraso	Esta vez fue por decisión propia: Como aún había suficiente tiempo, se prefirió dotar a los escenarios que en principio iban a tener un acabado de «prototipos» de mejoras visuales para que todos los niveles del producto estuviesen a un nivel equiparable.

Cuadro 4.9: Tabla de retrasos del proyecto

Debemos tener presente que, respecto a la planificación inicial, disponemos de **2,5 semanas de red de seguridad** y, aunque no sea lo más idóneo, podemos usar éstas a lo largo del proyecto.

TODO: Conclusiones, por ejemplo, el rol de jefe de proyecto durante las iteraciones trabajó más horas de las proyectadas, etc.

COSTES

1

2 *n este capítulo abordaremos una de las partes más importantes en la planificación de
3 un proyecto: los costes, y para ello los dividiremos en costes de personal, de material e
4 indirectos.*

5.1 RESUMEN DE COSTES DEL PROYECTO

Uno de los aspectos más importantes para que un proyecto finalice con éxito es que esté bien presupuestado desde su inicio. No obstante, es un objetivo difícil de alcanzar, por tanto lo que se pretende con la realización de este capítulo es, primeramente, intentar que se ajuste lo máximo a la realidad y, en caso de que se desvíe demasiado de la misma, aprender de los errores cometidos para futuros proyectos.

La tabla que se muestra a continuación anticipa los costes a los que tendremos que hacer frente durante la ejecución del proyecto:

Resumen del proyecto	
Costes de personal	14.242,46 €
Sueldo neto	9.466,9 €
Impuestos	4.283,63 €
Costes sociales	492,9 €
Costes materiales	357,78 €
Costes indirectos	4.127,34 €
TOTAL	18.727,58 €

Cuadro 5.1: Tabla resumen de costes

En las siguientes secciones veremos en detalle cómo están calculados cada uno de los resultados mostrados.

5.2 COSTES DE PERSONAL

Los costes de personal suponen una gran parte del coste total de un proyecto en el marco de las empresas del sector TIC.

En este caso, el proyecto tendría un único trabajador pero debemos tener en cuenta que desarrollaría diversas tareas dentro del mismo, por tanto el salario no puede ser idéntico para una labor u otra. Tal y como se ha descrito previamente se han establecido 3 roles: «jefe de proyecto», «diseñador» y «programador» (para más detalles, revisar sección §3.2.3).

1 Con el fin de calcular el coste que supondrían los honorarios nos apoyamos en los
 2 estudios realizados por «Gamasutra», en este caso en la encuesta salarial de 2014 [11]
 3 (la última realizada hasta la fecha). Este estudio nos muestra el salario medio para
 4 varios puestos o roles del sector, pero a continuación se muestran los roles de nuestro
 5 proyecto junto con los roles que nos interesan de los descritos en el estudio:

Salarios medios brutos según la encuesta salarial de «Gamasutra»		
Categoría	Categoría «Gamasutra»	Salario medio anual
Jefe de proyecto	«Business and management» <i>(Negocios y gestión)</i>	101.572 \$
Diseñador	«Game designers» <i>(Diseñadores de juego)</i>	73.864 \$
Programador	«Programmers and engineers» <i>(Programadores e ingenieros)</i>	93.251 \$

Cuadro 5.2: Salarios medios brutos según la encuesta salarial de «Gamasutra» (2014) [11]

6 Elegir esta encuesta plantea una serie de problemas, y es algo que se ha pretendido
 7 evitar intentando obtener otra fuente para elaborar este capítulo, pero los salarios de la
 8 industria del videojuego en nuestro entorno actualmente no trascienden frecuentemen-
 9 te y si lo hacen es de una manera mucho más escueta de la que nos ofrece la encuesta
 10 salarial de «Gamasutra», sin reparar siquiera en diferenciar salarios por categorías.

11 Eligiendo esta encuesta se nos plantean dos problemas principalmente:

- 12 1. **Diferencia de nivel de vida:** Debemos tener presente que el nivel de vida esta-
 13 dounidense y el español están lejos de ser parejos, por lo que no podemos com-
 14 pararlos directamente como si se tratase de una encuesta salarial de un país de
 15 nuestro entorno en ámbitos económicos.
- 16 2. **Diferencia de moneda:** La encuesta salarial nos ofrece los datos en dólares, mien-
 17 tras que necesitamos los datos en euros.

18 Nótese además que nomenclatura de los roles de nuestro proyecto se adaptan a le
 19 perfección a las usadas por «Gamasutra», salvo en el caso del «Jefe de proyecto» que se
 20 ha decidido relacionarlo con «Negocios y gestión». Para justificar esta relación tenemos
 21 que volver a la encuesta salarial y buscar la definición que hace «Gamasutra» de esta
 22 categoría, que se muestra a continuación:

Esta categoría incluye a las personas cuyo trabajo es tener a la compañía organizada y, en los mejores casos, financieramente saludables. Esto incluye a las personas que son ejecutivos, directores ejecutivos, personal legal, recursos humanos, personal de tecnología de la información, personal encargado de la administración de contenido y personal de administración general [11].

Como podemos apreciar en el texto extraído de la encuesta salarial el rol descrito coincide en un amplio porcentaje con el rol de «jefe de proyecto» que describíamos anteriormente (como se decía previamente, para más detalles sobre los roles de este proyecto, revisar la sección §3.2.3), por lo que queda justificada su relación.

Ahora vamos a pasar a solucionar los problemas que nos planteaba la elección de la encuesta salarial:

Para adaptar, en la medida de lo posible, un sueldo estadounidense a su equivalente español vamos a comparar el nivel de vida de ambos países usando el «índice de coste de vida» y, para ser más concretos, vamos a comparar el coste de vida de San Francisco (puesto que es el lugar en el que más empleos del sector videojuegos se concentran en EE.UU. [10] y, por ende, del lugar que presumiblemente provienen más datos de la encuesta salarial) con el de Sevilla.

Comparativa respecto al «índice de coste de vida»	
Ciudad	Indice de coste de vida
San Francisco	101,94
Sevilla	56,49

Cuadro 5.3: Tabla comparativa costes de vida [19]

Como observamos en la tabla, a San Francisco se le asigna un índice de **101,94** mientras que a Sevilla de **56,49** [19], por lo que aplicando una sencilla de operación de división entre ambos obtenemos el coeficiente por el que multiplicaremos los datos ofrecidos por «Gamasutra»:

$$\text{Coeficiente}_{\text{CosteVida}} = \frac{\text{IndiceCosteVida}_{\text{SanFrancisco}}}{\text{IndiceCosteVida}_{\text{Sevilla}}} = \frac{56,49}{101,94} = 0,55$$

Con la obtención de este coeficiente ya tenemos paliado el primer problema, pero no podemos aplicar directamente este coeficiente a los salarios hasta que no solucionemos el restante.

Como se anticipaba previamente, puesto que los datos que nos ofrece la encuesta no están en nuestra moneda es necesario hacer un cambio de la misma. Esto es un problema, ya que el valor de una divisa respecto a la otra fluctúa en el tiempo.

Para paliar este problema vamos a realizar un pequeño estudio sobre la fluctuación euro-dólar en los dos últimos años consultando el histórico de diferencias [13]:

Fecha	Último	Apertura	Máximo	Mínimo	% var.
Jul 2017	1,1341	1,1421	1,1427	1,1313	-0,74%
Jun 2017	1,1426	1,1240	1,1448	1,1117	1,63%
May 2017	1,1243	1,0905	1,1269	1,0838	3,18%
Abr 2017	1,0897	1,0658	1,0952	1,0568	2,30%
Mar 2017	1,0652	1,0573	1,0907	1,0492	0,71%
Feb 2017	1,0577	1,0796	1,0830	1,0492	-2,05%
Ene 2017	1,0798	1,0529	1,0814	1,0339	2,68%
Dic 2016	1,0516	1,0588	1,0875	1,0350	-0,68%
Nov 2016	1,0588	1,0979	1,1302	1,0515	-3,58%
Oct 2016	1,0981	1,1233	1,1245	1,0848	-2,31%
Sep 2016	1,1241	1,1155	1,1329	1,1119	0,74%
Ago 2016	1,1158	1,1173	1,1367	1,1043	-0,14%
Jul 2016	1,1174	1,1103	1,1200	1,0950	0,62%
Jun 2016	1,1105	1,1130	1,1434	1,0909	-0,24%
May 2016	1,1132	1,1444	1,1617	1,1096	-2,83%
Abr 2016	1,1456	1,1378	1,1466	1,1213	0,67%
Mar 2016	1,1380	1,0870	1,1413	1,0820	4,66%
Feb 2016	1,0873	1,0831	1,1377	1,0812	0,33%
Ene 2016	1,0837	1,0860	1,0986	1,0709	-0,21%
Dic 2015	1,0860	1,0566	1,1059	1,0538	2,80%
Nov 2015	1,0564	1,1013	1,1053	1,0556	-4,01%
Oct 2015	1,1005	1,1175	1,1496	1,0894	-1,54%
Sep 2015	1,1177	1,1209	1,1460	1,1086	-0,34%
Ago 2015	1,1215	1,0968	1,1715	1,0847	2,07%
Máximo: 1,1715		Mínimo: 1,0339	Diferencia: 0,1376	Promedio: 1,1008	% var.: 3,2126

Figura 5.1: Diferencias euro-dólar entre Agosto de 2015 y Julio de 2017 [13]

Podemos observar que se encuentran relativamente estabilizados, para el propósito que nos ocupa, y que el promedio se sitúa en 1,1008 dólares por cada euro, que se encuentra muy cerca del valor actual.

Es por ello usaremos la siguiente equivalencia:

$$1 \text{ euro} = 1,1 \text{ dólares estadounidenses}$$

O lo que es lo mismo:

$$1 \text{ dólar estadounidense} = 0,9 \text{ euros}$$

El resultado de aplicar dicha conversión y el coeficiente referente al coste de vida que calculamos previamente es el siguiente:

IMPORTANTE: Los datos están desactualizados de aquí en adelante, faltaría multiplicar por 0.55 aquí y todos los cambios derivados de ello, pero me esperaré a cambiarlos hasta que la sección esté validada.

Salarios medios brutos (adaptados) según la encuesta salarial de «Gamasutra»		
Categoría	Categoría «Gamasutra»	Salario medio anual
Jefe de proyecto	«Business and management» (Negocios y gestión)	91.414,8 €
Diseñador	«Game designers» (Diseñadores de juego)	66.477,6 €
Programador	«Programmers and engineers» (Programadores e ingenieros)	83.925,9 €

Cuadro 5.4: Salarios medios brutos (adaptados) según la encuesta salarial de «Gamasutra» (2014) [11]

A partir de la tabla anterior, debemos calcular los honorarios por hora de cada uno de los roles. Para realizar esto debemos dividir el salario anual entre la jornada anual:

$$\text{Salario}_{\text{Hora}} = \frac{\text{Salario}_{\text{Anual}}}{\text{JornadaLaboralAnual}_{\text{Horas}}}$$

La jornada máxima anual del sector TIC está fijada en 1.800 horas en nuestro país [5], por tanto, una vez aplicado esto a cada rol, obtendríamos los siguientes resultados:

Tabla salarial (sueldos brutos) del proyecto por hora

Categoría	Salario bruto por hora
Jefe de proyecto	50,78 €
Diseñador	36,93 €
Programador	46,62 €

Cuadro 5.5: Tabla salarial (sueldos brutos) del proyecto por hora

Aunque no aparezca de manera explícita, debemos remarcar que el salario medio al que hace referencia «Gamasutra» es el **salario bruto**. No tiene sentido que una encuesta salarial se realice con salarios netos, puesto que la cantidad final que el trabajador recibe varía según sus características o condiciones personales.

Para calcular el sueldo bruto de cada uno de los roles, debemos tomar la planificación del proyecto que hemos visto previamente (sección §4.1) y multiplicar el total de horas de cada rol por su coste a la hora que acabamos de calcular, así pues:

$$SueldoBruto = SalarioBrutoHora * HorasProyecto$$

Y, a su vez, podemos concluir que:

$$TotalSueldosBruto = \sum_{i=1}^{n^{\text{empleados}}} SueldoBruto_i$$

Por tanto, aplicando la primera fórmula a cada uno de los roles y la segunda fórmula para calcular el total, obtenemos:

Costes sueldos brutos			
Rol	Total de horas	Sueldo bruto por hora	Sueldo bruto
Jefe de proyecto	94 horas	50,78 €	4.773,32 €
Costes materiales	62 horas	36,93 €	2.289,66 €
Costes indirectos	154 horas	46,62 €	7.179,48 €
TOTAL			14.242,46 €

Cuadro 5.6: Tabla costes sueldos brutos

Por tanto, el coste del proyecto en cuanto a sueldos brutos asciende a **14.242,46 €**.

Dentro de los costes de personal debemos diferenciar entre el **sueldo neto del trabajador**, los **impuestos** y los **costes sociales**.

A continuación (diferenciando el sueldo neto, los impuestos y los costes sociales dentro de cada uno), se muestra una tabla resumen por rol:

Costes sueldo rol «jefe de proyecto»			
Concepto	Anual	Hora (Anual / 1800)	Proyecto (Hora * 94)
Sueldo neto	59.760,7 €	33,2 €	3.120,8 €
Impuestos	28.795,7 €	16,0 €	1.504,0 €
Costes sociales	2.858,4 €	1,59 €	149,46 €
Sueldo bruto	91.414,8 €	50,78 €	4.773,32 €

Cuadro 5.7: Tablas costes sueldo rol «jefe de proyecto»

Costes sueldo rol «diseñador»			
Concepto	Anual	Hora (Anual / 1800)	Proyecto (Hora * 62)
Sueldo neto	46.049,2 €	25,58 €	1.585,96 €
Impuestos	17.570,0 €	9,76 €	605,15 €
Costes sociales	2.858,4 €	1,59 €	98,58 €
Sueldo bruto	66.477,6 €	36,93 €	2.289,66 €

Cuadro 5.8: Tablas costes sueldo rol «diseñador»

Costes sueldo rol «programador»			
Concepto	Anual	Hora (Anual / 1800)	Proyecto (Hora * 154)
Sueldo neto	55.646,3 €	30,91 €	4.760,14 €
Impuestos	25.421,2 €	14,12 €	2.174,48 €
Costes sociales	2.858,4 €	1,59 €	244,86 €
Sueldo bruto	83.925,9 €	46,62 €	7.179,48 €

Cuadro 5.9: Tablas costes sueldo rol «programador»

Como podemos comprobar, si sumamos el sueldo neto, los impuestos y los costes sociales obtenemos el sueldo bruto (salvando el margen de error proveniente del uso de dos decimales).

- ¹ Si sumamos los sueldos netos, impuestos y costes sociales de cada uno de los roles,
² obtenemos los referentes al proyecto:

$$Costes_{SueldosNetos} = \sum_{i=1}^{n^{\text{empleados}}} SueldoNeto_i$$

$$Costes_{Impuestos} = \sum_{i=1}^{n^{\text{empleados}}} Impuesto_i$$

$$Costes_{CostesSociales} = \sum_{i=1}^{n^{\text{empleados}}} CosteSocial_i$$

- ³ Por lo que concluimos:

- ⁴ ■ **Costes sueldo neto del proyecto:** 9.466,9 €
⁵ ■ **Costes impuestos del proyecto:** 4.283,63 €
⁶ ■ **Costes sociales del proyecto:** 492,9 €

- ⁷ **TODO: Añadir gráfica del porcentaje de costes de personal que representan el
⁸ sueldo neto, impuestos y sociales del proyecto.**

5.3 COSTES MATERIALES

Los costes materiales son costes independientes de un proyecto en concreto, por lo que el coste que supone para este proyecto no se corresponde directamente su valor de adquisición, sino que es necesario realizar una amortización teniendo en cuenta la duración de este proyecto respecto a la vida útil de cada elemento.

La amortización se realizaría de la siguiente manera:

$$Coste_{Mes} = \frac{CosteCompra}{VidaUtil_{Meses}}$$

Para la ejecución del proyecto se cuenta con:

- Dos equipos, un sobremesa y un portátil (más detalles sobre los equipos a continuación) ambos con una vida útil estimada de 3 años ya que el sobremesa, aún siendo más antiguo, ha sido actualizado recientemente.
- Una mesa de trabajo, con una vida útil de 4 años.
- Una pequeña mesa de reuniones que se utilizaría en caso de tener que mantener una reunión con una persona interesada en el proyecto y que contaría con una vida útil de 6 años.
- Cinco sillas: una de escritorio utilizada en la mesa de trabajo y cuatro, más simples, utilizadas en la mesa de reuniones. La utilizada en la mesa de trabajo costaría con una vida útil de 4 años y el resto contaría con un vida útil de 6 años.
- Dos repisas, ambas con una vida útil de 6 años.
- Una estantería, con una vida útil de 6 años.
- Dos papeleras, con una vida útil de 10 años.

- ¹ Las características de los citados equipos son:

Características sobremesa	
Sistema/s operativo/s	Windows 10
Procesador	Intel Core i7 930 2.80Ghz
Placa base	Asus P6X58D-E
Memoria RAM	12GB DDR3 1600Mhz PC3-12800 CL6 (2x4GB + 2x2GB)
Disp. almacenamiento	SSD 120GB + Disco duro 3TB SATA3 7200rpm
Tarjeta gráfica	Sapphire Radeon HD 7950 OC 3GB GDDR5

Cuadro 5.10: Tabla de características del dispositivo sobremesa

Características portátil	
Sistema/s operativo/s	Windows 10
Procesador	Intel Core i7 4712MQ 2.3 GHz
Placa base	(Dato no proporcionado)
Memoria RAM	8GB DDR3 SODIMM (1x8GB)
Disp. almacenamiento	Disco duro 1TB SATA 5400rpm
Tarjeta gráfica	Nvidia GeForce GT820M 2GB GDDR3

Cuadro 5.11: Tabla de características del dispositivo portátil

La tabla que se muestra a continuación muestra los materiales que van a cumplir una utilidad en la realización del proyecto, su vida útil, su valor y el coste real al mes que supone en el proyecto:

Amortización de materiales			
Material	Valor	Vida útil	Coste real mes
Sobremesa	1100 €	3 años (36 meses)	30,55 €
Portátil	700 €	3 años (36 meses)	19,44 €
Mesa trabajo	200 €	4 años (48 meses)	4,16 €
Mesa reuniones	150 €	6 años (72 meses)	2,08 €
Silla trabajo	60 €	4 años (48 meses)	1,25 €
Silla reuniones x 4	40 €	6 años (72 meses)	0,55 * 4 = 2,22 €
Repisa x 2	30 €	6 años (72 meses)	0,41 * 2 = 0,83 €
Estantería	80 €	6 años (72 meses)	1,11 €
Papelera x 2	5 €	10 años (120 meses)	0,04 * 2 = 0,08 €
TOTAL (mes)			59,63 €

Cuadro 5.12: Tabla de amortización de materiales

Una vez realizado esto, sólo faltaría multiplicar el coste material al mes por la duración del proyecto en meses, de esta forma:

$$\text{CosteMaterial}_{\text{Proyecto}} = \text{CosteAmortizado} * \text{MesesProyecto}$$

Teniendo en cuenta el número de meses del proyecto, en este caso **6 meses**, el coste material total es de **357,78 €**.

5.4 COSTES INDIRECTOS

Los costes indirectos son, sin lugar a dudas, los más complicados de calcular a la hora de elaborar un presupuesto.

Siguiendo su definición, un coste indirecto es aquel afecta al proceso productivo en general de uno o más productos, en este caso proyectos, y que por tanto es complicado asignar una parte de ese coste a un proyecto en concreto sin elaborar algún criterio, sistemático y estable en el tiempo, que nos permita hacerlo.

Existen gran variedad de costes indirectos, como podrían ser licencias, seguros, bancos, contratación de servicios profesionales (como podrían ser abogados para posibles temas legales o gestoras para, por ejemplo, llevar a cabo la contabilidad o las declaraciones a hacienda), alquileres, etc.

En el caso de este proyecto, al mes, son los siguientes:

Costes indirectos	
Motivo del coste	Coste mes
Licencias	0 €*
Alquiler	400 €
Internet (fibra óptica)	37,89 €
Luz	60 €
Agua	20 €
Material fungible	20 €
Servicios de limpieza	150 €
TOTAL (mes)	687,89 €

Cuadro 5.13: Tabla de costes indirectos

[*]: En cuanto a licencias debemos tener en presente que Unreal Engine es un motor de uso gratuito pero, sin embargo, la compañía propietaria del motor, en este caso Epic Games, establece en su EULA una cláusula por la que habrá que abonar cierta cantidad, un loyalty, si se superan los 3.000\$ de beneficio por cuatrimestre. Esta cantidad en gira en torno a un 5% de los beneficios que pasen de 3.000\$: por ejemplo, si se generan 5000\$ de beneficio habrá que remitirle a Epic Games aproximadamente 100\$ (el 5% de 2000\$, resultado de la resta de 5.000\$ - 3.000\$).

En este caso, el coste indirecto al mes no sería más que el sumatorio de todos los

costes indirectos al mes:

1

$$\text{CosteMaterial}_{\text{Mes}} = \sum_{i=1}^{\text{n}^{\circ}\text{costes}} \text{CosteIndirecto}_i$$

Por consiguiente, el coste indirecto total del proyecto sería el coste indirecto por mes multiplicado por el número de meses del proyecto:

2

3

$$\text{CosteMaterial}_{\text{Proyecto}} = \text{CosteAmortizado} * \text{MesesProyecto}$$

Por último, teniendo presente el número de meses del proyecto, en el caso que nos ocupa **6 meses**, el coste indirecto total del proyecto asciende a **4.127,34 €.**

4

5

PARTE III

DESARROLLO DEL PROYECTO

ARRANQUE

1

² The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck
³ of his whole damn life — and one is as good as the other.

6

7 Durante este capítulo de la memoria se presentará la lista de características que con-
8 forman el proyecto, así como el diseño arquitectónico de Unreal Engine, el motor
9 gráfico elegido para llevar a cabo el proyecto.

6.1 LISTA DE CARACTERÍSTICAS

La lista de características está confencionada a partir del documento de diseño, incorporado como anexo a este documento (Ver apéndice §B), por tanto, para encontrar una descripción más detallada tendremos que remitirnos a dicho anexo.	2 3 4
1. Mecánica: Movimiento del personaje.	5
■ Implementación del código de la mecánica.	6
■ Asignación de teclas y botones vinculados la mecánica.	7
■ Desarrollo de la animación del personaje relacionada con la mecánica.	8
2. Mecánica: Rotación del personaje.	9
■ Implementación del código de la mecánica.	10
■ Asignación de teclas y botones vinculados la mecánica.	11
■ Desarrollo de la animación del personaje relacionada con la mecánica.	12
3. Mecánica: Sprint.	13
■ Implementación del código de la mecánica.	14
■ Asignación de teclas y botones vinculados la mecánica.	15
■ Desarrollo de la animación del personaje relacionada con la mecánica.	16
4. Mecánica: Salto.	17
■ Implementación del código de la mecánica.	18
■ Asignación de teclas y botones vinculados la mecánica.	19
■ Desarrollo de la animación del personaje relacionada con la mecánica.	20
5. Mecánica: Agachado.	21
■ Implementación del código de la mecánica.	22
■ Asignación de teclas y botones vinculados la mecánica.	23
■ Desarrollo de la animación del personaje relacionada con la mecánica.	24
6. Mecánica: Tiempo bala (o «Time dilation»).	25
■ Implementación del código de la mecánica.	26

- 1 ■ Asignación de teclas y botones vinculados la mecánica.
- 2 ■ Desarrollo de la animación del personaje relacionada con la mecánica.
- 3 ■ Gestión de adrenalina relacionada con la mecánica.

4 7. Mecánica: Parpadeo (o «Blink»).

- 5 ■ Implementación del código de la mecánica.
- 6 ■ Asignación de teclas y botones vinculados la mecánica.
- 7 ■ Desarrollo de la animación del personaje relacionada con la mecánica.
- 8 ■ Gestión de adrenalina relacionada con la mecánica.
- 9 ■ Implementación del código del indicador de blink disponible.

10 8. .

11 9. .

12 10. .

13 11. .

14 12. .

15 13. .

16 14. .

17 15. .

18 16. .

19 17. .

20 18. .

21 19. .

22 20. .

23 21. .

24 22. .

25 23. .

CAPÍTULO 6. ARRANQUE

24. .	1
25. .	2
26. .	3
27. .	4
28. .	5
29. .	6
30. .	7

6.2 DISEÑO ARQUITECTÓNICO

En el momento que hablamos de diseño arquitectónico en el marco de este proyecto estamos refiriéndonos al diseño arquitectónico del motor que estamos utilizando, «Unreal Engine 4». La información que se muestra a continuación está extraída de fuentes oficiales de Epic Games, propietaria de «Unreal Engine», en concreto de la documentación (en inglés) del motor [9].

Cuando programamos elementos del gameplay (elementos del juego) usando código C++, cada módulo («gameplay module») puede contener muchas clases en C++.

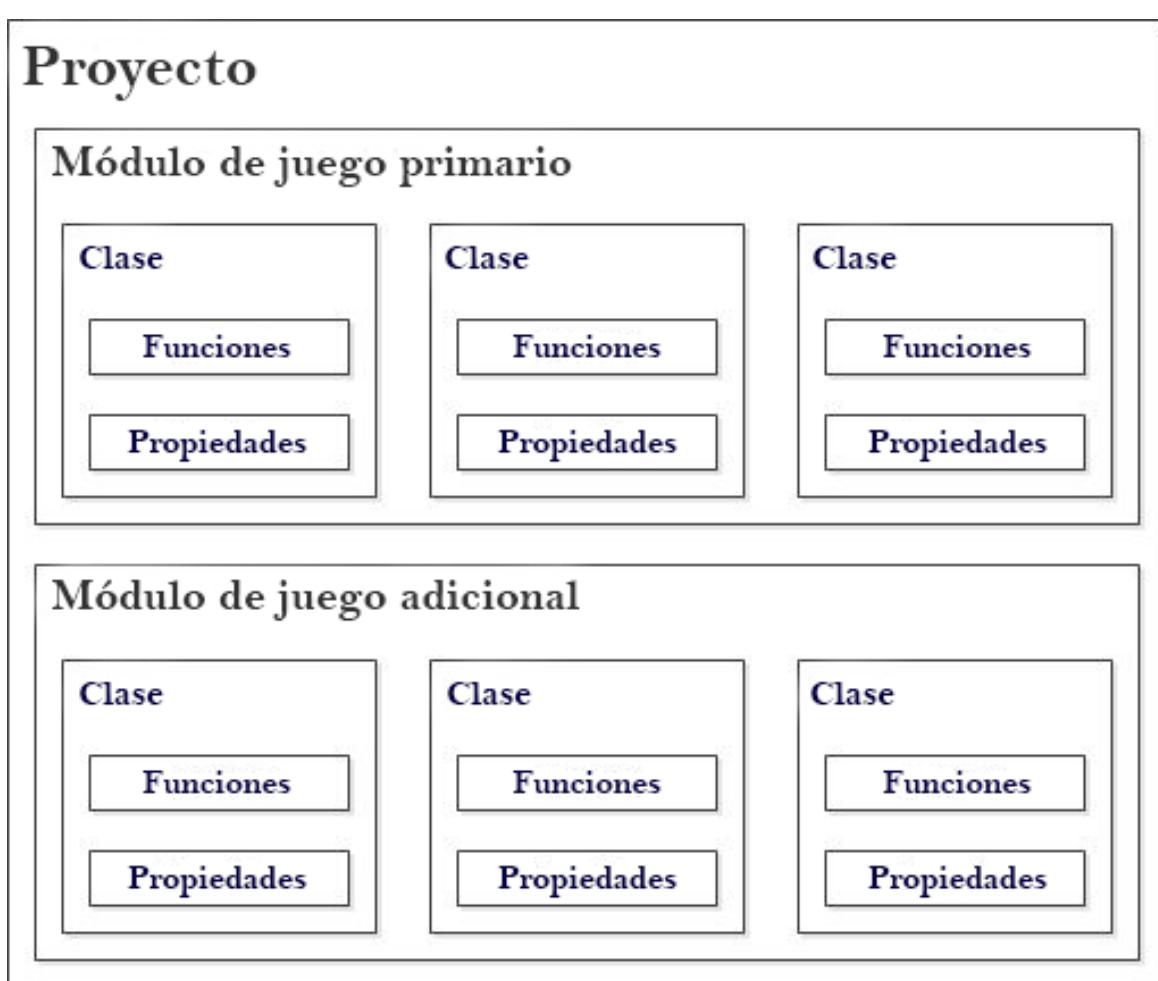


Figura 6.1: Arquitectura Unreal Engine 4

Cada clase determina una plantilla para un nuevo actor u objeto. En el archivo de encabezado de clase («.h») se declaran la clase y cualquier función o propiedades de la clase. Las clases también pueden contener «structs», estructuras de datos que ayudan en la organización y manipulación de las propiedades relacionadas. Las estructuras

también pueden ser definidas en su propia interfaz, permitiendo que puedan ser implementadas en diferentes clases.

Cuando programamos en Unreal Engine, es posible tener clases, funciones y variables estándar del lenguaje C++, usando la sintaxis estándar de C++. Sin embargo, también podemos usar las instrucciones «UCLASS()», «UFUNCTION()» y «UPROPERTY()» para que Unreal Engine conozca nuevas clases, funciones y propiedades, respectivamente. Por ejemplo, una variable cuya declaración está precedida por «UPROPERTY()» puede ser reconocida por el motor y ser mostrada y editada en el editor de Unreal Engine. También existen «UINTERFACE()» y «USTRUCT()» y palabras claves para cada una que pueden ser usadas para especificar el comportamiento de una clase, función, propiedad, interfaz o estructura en Unreal Engine.

6.2.1 Módulos de juego

De la misma forma en la que el propio motor está compuesto por una colección de módulos, cada juego está compuesto por uno o más módulos de juego. Estos son similares a los paquetes de las versiones anteriores del motor, en las cuales estos eran contenedores de una colección de clases relacionadas. En Unreal Engine 4, ya que está basado en C++, **los módulos son en realidad DLLs**, en lugar de paquetes.

Los módulos de juego deben contener, como mínimo, un archivo de cabecera («.h»), un archivo C++ («.cpp») y un archivo de compilación («*.Build.cs»).

El archivo de cabecera debe estar localizado en la carpeta «Public» de la localización del módulo, por ejemplo: «[NombreDelJuego]/Source[NombreDelMódulo]/Public». Este archivo contiene cualquier archivo de cabecera, incluyendo las cabeceras auto-generadas del módulo, necesario para compilar las clases que incluye el módulo.

El archivo «C++», localizado en la carpeta «Privada» de la localización del módulo («[NombreDelJuego]/Source[NombreDelMódulo]/Private»), implementa el módulo.

El archivo de compilación («build») es está localizado en la raíz del directorio del módulo («[NombreDelJuego]/Source[NombreDelMódulo]») y contiene información usada por la herramienta de compilación de Unreal Engine («UnrealBuildTool») para compilar el módulo.

1 Módulos de juego múltiples

2 Hay diferentes puntos de vista respecto a la separación en DLL. Separar el juego en
 3 un montón de archivos DLL puede tener más inconvenientes que beneficios, pero es
 4 una decisión que debe ser hecha por cada equipo de desarrollo basada en sus necesi-
 5 dades. Usar múltiples módulos de juego puede conducir en mejores tiempos de enlace
 6 e iteración de código más rápida, pero con más módulos será necesario lidiar con ex-
 7 portaciones de DLL e interfaces más a menudo. Esta concesión es la correcta para el
 8 motor y el editor, pero es cuestionable para el gameplay.

9 También se puede crear un módulo de juego primario y un número indeterminado
 10 de módulos de juego adicionales, para ello es necesario crear los archivos «*.Build.cs»
 11 para los citados módulos adicionales, además de añadir las referencias pertinentes
 12 en el archivo «Target.cs». Además, el módulo principal deberá contener la etiqueta
 13 «IMPLEMENT_PRIMARY_GAME_MODULE» y el resto de módulos secundarios la eti-
 14 queta «IMPLEMENT_GAME_MODULE». La herramienta de compilación de Unreal
 15 Engine deberá entonces descubrir automáticamente los módulos y compilar los DLL
 16 adicionales.

17 Limitaciones

18 Unreal Engine soporta la creación de módulos que son dependientes entre sí (es de-
 19 cir, que ambos contienen importaciones y exportaciones que hacen referencia al otro),
 20 pero no es ideal para los tiempos de compilación. Los módulos sin dependencias cru-
 21 zadas son difíciles de diseñar y mantener, pero a cambio el código es más limpio por
 22 ello.

23 6.2.2 Plugins

24 Muchos subsistemas de Unreal Engine fueron diseñados para ser extensibles, per-
 25 mitiéndole al usuario añadir nuevas funcionalidades y modificar la funcionalidad exis-
 26 tente sin modificar el código del propio motor directamente. Se pueden crear nuevos
 27 tipos de archivos, agregar nuevos elementos en el menú, nuevos comandos a la barra
 28 de herramientas o incluso agregar nuevas funciones y sub-modos de editor.

29 Anatomía

30 Los plugins con código tendrán una carpeta fuente («Source»). Esta carpeta con-
 31 tendrá un o más directorios con el código fuente del plugin. Nótese que
 32 los plugins a menudo contienen código, pero no tienen por qué contenerlo.

Para plugins con módulos de código, el plugin tendrá su propia carpeta de binarios («Binaries») que contendrá el código compilado para ese plugin. También se guardarán archivos de compilación temporales en la carpeta «Intermediate» dentro del directorio del plugin.

Los plugins pueden tener su propia carpeta de contenido («Content») que contendrá assets específicos para ese plugin.

6.2.3 Clases de gameplay

Cada clase de gameplay en Unreal Engine está compuesta por una clase cabecera («.h») y una clase de archivo fuente («.cpp»). La clase cabecera contiene las declaraciones de la clase y sus miembros, como variables y funciones, mientras que la clase de archivo fuente es donde la funcionalidad de la clase se define implementando las funciones que pertenecen a la clase.

Las clases en Unreal Engine tienen una estructura de nombres estandarizadas, por la que se podrá conocer instantáneamente qué tipo de clase es simplemente mirando a la primera letra, o prefijo. Los prefijos de las clases de gameplay son:

Prefijos de clases de gameplay en Unreal Engine

Prefijo	Significado
A	Extiende desde la clase base «spawneable». Son los denominados «actores», y pueden ser emplazados directamente en el mundo o escenario del juego.
U	Extiende desde la clase base de todos los objetos de juego. No pueden ser instanciados directamente en el mundo, deben pertenecer a un «actor». Son generalmente objetos como «componentes».

- A** Extiende desde la clase base «spawneable». Son los denominados «actores», y pueden ser emplazados directamente en el mundo o escenario del juego.
- U** Extiende desde la clase base de todos los objetos de juego. No pueden ser instanciados directamente en el mundo, deben pertenecer a un «actor». Son generalmente objetos como «componentes».

Cuadro 6.1: Prefijos de clases de gameplay en Unreal Engine

Clases de cabecera

Las clases de juego o «gameplay» en Unreal Engine generalmente tienen archivos de cabecera por separado y únicos. Estos archivos suelen ser nombrados para coincidir con la clase que se está definiendo, menos por el prefijo «A» o «U», y usando la extensión «.h». Por tanto, la clase de cabecera para la supuesta clase «AActor» sería «Actor.h». Nótese que aunque se recomienda hacer uso de esta nomenclatura no se establece ninguna relación formal entre los archivos por definirlos de ese modo (al menos

¹ en la versión actual del motor).

² Las clases de cabecera usan la sintaxis estándar de «C++», que se pueden combinar
³ con instrucciones especializadas como ya se vio anteriormente al inicio de la sección
⁴ (§6.2).

⁵ Al principio de cada clase de cabecera de una clase de gameplay, el archivo de
⁶ cabecera generado (creado automáticamente) necesita incluirse. Por tanto, al principio
⁷ de la clase «ClasePrueba.h» deberá aparecer la siguiente línea:

⁸ #include "ClasePrueba.generated.h"

⁹ Declaración de clases

¹⁰ La declaración de clase indica el nombre de la clase, de qué clase hereda y, además,
¹¹ cualquier función o variable que herede.

```
12 UCLASS([ especificador , especificador , ... ] ,  

13           [meta( clave=valor , clave=valor , ... )])  

14 class NombreDeLaClase : public NombreDeLaClasePadre  

15 {  

16     GENERATED_BODY()  

17 }
```

¹⁸ La declaración consiste en una declaración estándar de «C++» para una clase, pero
¹⁹ además de la declaración estándar, como se ve en la pieza de código, se usa la ins-
²⁰ trucción «UCLASS()» para añadir especificadores de clase y metadatos. Así mismo, la
²¹ etiqueta «GENERATED_BODY()» debe ser colocada justo al principio de la clase.

²² Especificadores de clases

²³ Cuando declaramos una clase podemos añadir especificadores a la declaración para
²⁴ controlar cómo es el comportamiento de la clase en varios aspectos del motor y del
²⁵ editor. La siguiente lista muestra estos especificadores:

- Abstract
- BlueprintType
- AdvancedClassDisplay
- ClassGroup
- AutoCollapseCategories
- CollapseCategories
- AutoExpandCategories
- Config
- Blueprintable
- Const

- ConversionRoot
- CustomConstructor
- DefaultToInstanced
- DependsOn
- Deprecated
- DontAutoCollapseCategories
- DontCollapseCategories
- EditInlineNew
- HideCategories
- HideDropdown
- HideFunctions
- Intrinsic
- MinimalAPI
- NoExport
- NonTransient
- NotBlueprintable
- NotPlaceable
- PerObjectConfig
- Placeable
- ShowCategories
- ShowFunctions
- Transient
- Within

Implementación de clases

1

Todas las clases de juego (o «gameplay») para poder ser implementadas correctamente deben hacer uso de la etiqueta «GENERATED_BODY». Esto se hace, como acabamos de ver, en la clase de cabecera («.h») que define la clase y todas sus funciones y variables.

2

3

4

5

Los archivos de código fuente («.cpp») deben incluir los archivos de cabecera («.h») que contiene la declaración C++ de la clase, que normalmente es generada automáticamente pero puede ser creada manualmente si así se desea.

6

7

8

Constructores de clases

9

Los objetos en Unreal Engine usan constructores para asignar valores por defecto a propiedades así como realizar otras inicializaciones necesarias. El constructor de clase está emplazado en la clase de implementación y, por ejemplo, para la clase de implementación «AActor.cpp» el método del constructor será «AActor::AActor».

10

11

12

13

ITERACIÓN CERO

1

2 *The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck
3 of his whole damn life – and one is as good as the other.*

4 Ernest Hemingway (1899–1961),
5 Novelist

6

7 T ODO TODO TODO

7.1 INICIALIZACIÓN DEL PROYECTO

El motor gráfico sobre el que vamos a trabajar, como se decía al inicio de esta memoria, es gratuito y para obtenerlo solamente tenemos que acceder a su página oficial, <https://www.unrealengine.com/>.

Con esto habremos obtenido el «launcher» de Epic Games / Unreal Engine por lo que una vez descargado e instalado, queda un paso más: descargar una versión concreta del motor, en nuestro caso elegiremos la versión «4.16».

Una vez descargada e instalada la versión deseada del motor, procedemos con el siguiente paso: para la confección del trabajo vamos a crear un nuevo proyecto de Unreal Engine en «C++» y sin contenido adicional, es decir, vamos a crear un proyecto limpio en «C++». Para ello, en el Launcher de Unreal Engine, hacemos click en el botón «Iniciar» y se nos abrirá una ventana de selección de proyecto. Como queremos crear uno nuevo, hacemos click en la pestaña «New Project» o «Nuevo Proyecto» y una vez en esa pestaña hacemos click en la pestaña «C++», seleccionamos «Basic Code» o «Código básico» y configuramos el proyecto de la siguiente manera:

- «Desktop / Console» (o «Escritorio / Consola»): Puesto que nuestro proyecto está destinado a dispositivos sobremesa, y no a móviles o tablets, es la opción que debemos marcar cuando creamos nuestro proyecto.
- «Maximum quality» (o «Máxima calidad»): La calidad, como se detalla en el documento de diseño anexo a esta memoria, no es una prioridad para el proyecto, pero intentaremos que el resultado final luzca lo máximo posible.
- «No starter content» (sin contenido inicial): Como se detallaba previamente, vamos a crear un proyecto en «C++» con el mínimo contenido, para conocer nuestro proyecto el máximo posible, y si en algún momento necesitamos material del contenido inicial lo añadiremos manualmente.

Una vez configurado, hacemos click en «Create Project» (o «Crear Proyecto»).

Con esto, habremos terminado la creación de nuestro proyecto y tendremos casi lista la base sobre la que empezar a trabajar.

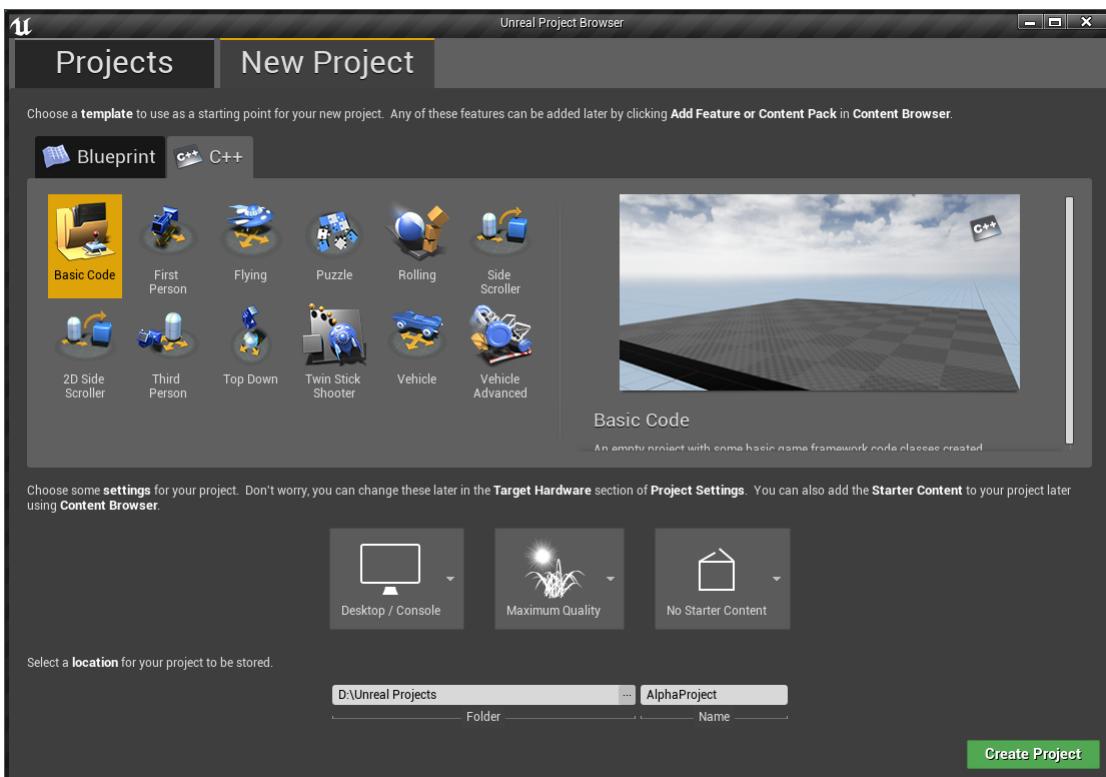


Figura 7.1: Ventana del creador de proyectos de Unreal Engine

1 7.2 PUESTA A PUNTO Y PRIMEROS PASOS

- 2 Antes de empezar a trabajar tenemos que realizar una serie de modificaciones:
- 3 TODO.

4 7.3 DIRETRICES GENERALES Y RESEÑAS

- 5 Durante la ejecución del proyecto se seguirán una serie de pautas, métodos y en definitiva trabajos que no se van a ver reflejados en su totalidad en la documentación de las siguientes iteraciones. Es por ello que esta sección pretende explicarlos y que ese trabajo, difícil de documentar ya que en muchos casos es paralelo al trabajo principal de cada iteración o engloba más de una iteración, esté documentado.

10 7.3.1 Uso de animaciones profesionales «Mixamo»

- Mixamoquoting 1
- «Mixamo» es una empresa filial de la conocida «Abobe» dedicada al mundo de la animación 3D y que se centraba en generar estas animaciones para su posterior comercialización. Actualmente, todas sus animaciones son de libre uso, por lo que supone una gran oportunidad para un proyecto indie como este. 2
3
4
5
- Las animaciones son de máxima calidad, tanto es así que se usan en juegos con alto presupuesto o ingresos: el juego más vendido del momento en PC, «Playerunknown's Battlegrounds» [29], utiliza varias de las animaciones de «Mixamo» y las sigue incorporando junto con nuevas funcionalidades en sus actualizaciones mensuales. 6
7
8
9
- Sin embargo, hay un gran problema derivado del uso de estas animaciones en concreto: el personaje que se usa en este proyecto utiliza el esqueleto estándar de Unreal Engine, y hace ya muchas versiones del motor gráfico «Mixamo» retiró el soporte para personajes que usen el esqueleto «por defecto» de Unreal Engine. 10
11
12
13
- La manera de solucionar esto es la incorporación de un esqueleto auxiliar, que no se utiliza en el proyecto salvo para este cometido, que nos servirá para realizar un «retargeting» a la animación. El proceso, una vez que tenemos listo el esqueleto destino y el esqueleto auxiliar es el siguiente: 14
15
16
17
1. Se 18
- Hay que repetir todos los pasos del 2 al X para cada una de las animaciones que se quieren incorporar al proyecto. TODO. 19
20
- TODO: Añadir al glosario retargeting. TODO: Capturas Retargeting. 21
- ### 7.3.2 Implementación del sistema de animación del personaje 22
- Para la ... se utilizará una máquina de estados... 23
- TODO. 24
- ### 7.3.3 Control de que el personaje no abandone el mapa 25
- TODO. 26

PRIMERA ITERACIÓN

1

2 *Durante la primera iteración se abordará el desarrollo de las características más bási-*
3 *cas que acompañan a este tipo de software: movimiento del personaje, rotación de su*
4 *cámara, etc., la implementación de mecánicas básicas como sprint, salto, etc. y las*
5 *mecánicas (poderes) que el personaje puede utilizar cuando esté potenciado.*

8.1 CARACTERÍSTICAS A DESARROLLAR

1. Creación del personaje.	2
2. Creación de una cámara «True first person camera».	3
3. Creación de una cámara en tercera persona.	4
4. Mecánica: Movimiento del personaje.	5
■ Implementación del código de la mecánica.	6
■ Asignación de teclas y botones vinculados la mecánica.	7
■ Desarrollo de la animación del personaje relacionada con la mecánica.	8
5. Mecánica: Rotación del personaje.	9
■ Implementación del código de la mecánica.	10
■ Asignación de teclas y botones vinculados la mecánica.	11
■ Desarrollo de la animación del personaje relacionada con la mecánica.	12
6. Mecánica: Rotación del personaje vinculada a un rate.	13
■ Implementación del código de la mecánica.	14
■ Asignación de teclas y botones vinculados la mecánica.	15
7. Mecánica: Sprint.	16
■ Implementación del código de la mecánica.	17
■ Asignación de teclas y botones vinculados la mecánica.	18
■ Desarrollo de la animación del personaje relacionada con la mecánica.	19
8. Mecánica: Salto.	20
■ Implementación del código de la mecánica.	21
■ Asignación de teclas y botones vinculados la mecánica.	22
■ Desarrollo de la animación del personaje relacionada con la mecánica.	23
9. Mecánica: Agachar.	24
■ Implementación del código de la mecánica.	25

- 1 ■ Asignación de teclas y botones vinculados la mecánica.
- 2 ■ Desarrollo de la animación del personaje relacionada con la mecánica.

3 10. Mecánica: Tiempo bala (o «Time dilation»).

- 4 ■ Implementación del código de la mecánica.
- 5 ■ Asignación de teclas y botones vinculados la mecánica.
- 6 ■ Desarrollo de la animación del personaje relacionada con la mecánica.
- 7 ■ Gestión de adrenalina relacionada con la mecánica.

8 11. Mecánica: Teletransporte (o «Blink»).

- 9 ■ Implementación del código de la mecánica.
- 10 ■ Asignación de teclas y botones vinculados la mecánica.
- 11 ■ Desarrollo de la animación del personaje relacionada con la mecánica.
- 12 ■ Gestión de adrenalina relacionada con la mecánica.
- 13 ■ Implementación del código del indicador de blink disponible.

14 12. Control de límites del mapa (en otras palabras, que el usuario no pueda abandonar el mapa y si lo hace, muera).

- 16 ■ Implementación del código de la mecánica.

17 13. Control de máxima distancia de caída (en otras palabras, que el usuario al caer en una superficie situada en un foso, muera).

- 19 ■ Implementación del código de la mecánica.

20 14. Movimiento de piernas del personaje al girarse.

- 21 ■ Implementación del código de la mecánica.
- 22 ■ Asignación de teclas y botones vinculados la mecánica.
- 23 ■ Desarrollo de la animación del personaje relacionada con la mecánica.

8.2 DISEÑO

1

Memorando técnico 1001: Creación del personaje

Asunto	Creación del personaje
Resumen	Consistiría en la creación de una capsula con la que interaccionarían los diferentes actores del entorno al, por ejemplo, colisionar con nuestro personaje (en vez de hacerlo con él directamente, ya que es mucho más costoso de calcular) y que alojaría a un modelo de personaje en su interior, adjunto a la capsula.
Factores causantes	-
Solución	Creamos nuestro personaje extendiendo la clase ACharacter y usamos la propiedad «Mesh», heredada de la clase padre y del tipo «USkeletalMeshComponent», para guardar nuestro modelo de personaje. Dependiendo del modelo de personaje debemos adelantar o retrasar un poco el mismo respecto al centro de la cápsula, rotarlo, etc para que la cámara que le añadimos, posteriormente, a nuestro modelo de personaje no traspase con las paredes ni genere ningún otro inconveniente.
Motivación	Esta solución se propone por algo que ya se ha comentado previamente: al utilizar una cápsula de colisión (o un cubo, esfera, etc) estamos reduciendo el número de cálculos que se tienen que hacer en muchas circunstancias, como la citada de que el personaje choque con una pared.
Cuestiones abiertas	Como se comentaba, hay que estar al tanto de si tenemos que mover nuestro personaje respecto a la cápsula, algo que no sabremos hasta que estemos interaccionando ya con el entorno.
Alternativas	-

Cuadro 8.1: Memorando técnico 1001: Creación del personaje

Nota: Al tratarse de la primera iteración (por tanto, no existían soluciones anteriores que se pudiesen ver afectadas en esta iteración) y teniendo en cuenta que las características a implementar no originan conflictos entre sí, no existen modificaciones que afecten al diseño.

2

3

4

5

Memorando técnico 1002: Creación de la cámara TFP

Asunto	Creación y anclaje de la cámara «true first person camera»
Resumen	El modo «true first person camera» (cámara en primera persona real) implica que tengamos un modelo de personaje completo de personaje con una cámara anexa a la cabeza del modelo, por contraposición a la típica cámara en primera persona que se utiliza normalmente.
Factores causantes	-
Solución	Creamos un «camera arm» (o brazo de cámara) para unir la cabeza del personaje con nuestra cámara, que también crearemos. Ajustamos la distancia del brazo de cámara para que la cámara se quede justo a la altura de los ojos del personaje y por delante de ellos y modificamos la propiedad «ProbeSize» del brazo de cámara para que cuando la cámara colisione con un obstáculo retroceda.
Motivación	Esta decisión genera muchos problemas, como que la cámara pueda traspasar paredes o introducirse dentro del personaje si no está correctamente calibrado, pero a la vez le da una mayor sensación de realismo, mejorando la experiencia del usuario (especialmente en un título encaminado a una adaptación a la realidad virtual).
Cuestiones abiertas	El valor de «ProbeSize» es idóneo, pero con las mecánicas que abordaremos en iteraciones posteriores en las que el personaje debe estar más pegado a obstáculos (por ejemplo cuando esté colgando de una cornisa) el valor actual de dicha variable puede ocasionar que la cámara se introduzca dentro del modelo del personaje.
Alternativas	Las alternativas principales son dos: usar cámara en primera persona sin modelo o usar una cámara en primera persona con un modelo de brazos y manos. La primera, aunque es por mucho la solución más fácil de implementar, queda descartada rápidamente en este género ya que, por ejemplo, debemos saber cuándo el personaje está agarrado a un saliente, cuándo está escalando, etc. La segunda sí podría ser una solución más viable pero tiene el inconveniente de que no generaría una sombra real del personaje y que dichos modelos de personaje, al ser muy específicos, son mucho más complicados de encontrar con una calidad aceptable.

Cuadro 8.2: Memorando técnico 1002: Creación de la cámara TFP

Memorando técnico 1003: Creación de la cámara TP

Asunto	Creación de cámara en 3ra persona «al hombro»
Resumen	Se persigue la creación de una cámara más alejada al personaje que la anterior y que nos permita ver su cuerpo.
Factores causantes	-
Solución	Creamos un «camera arm» (o brazo de cámara) para unir en un extremo la cabeza del personaje dicho brazo. Ahora creamos otro «camera arm», pero esta vez cambiando su rotación 90% respecto al primero y lo unimos al extremo que queda libre del primer brazo de cámara. El final de este último brazo de cámara será el lugar en el que se emplace finalmente la cámara (con la rotación oportuna para que apunte al personaje).
Motivación	Esta elección es mucho más profesional que utilizar un simple brazo de cámara, que en nos dejaría el personaje demasiado en medio de la pantalla, tapando gran parte de ella.
Cuestiones abiertas	-
Alternativas	La alternativa es usar sólo un brazo de cámara, pero si optásemos por esa decisión, al usar un solo brazo de cámara que saldría directamente de la cabeza del personaje el personaje estaría demasiado en medio de la pantalla y su cabeza estorbaría para ver correctamente.

Cuadro 8.3: Memorando técnico 1003: Creación de la cámara TP

Memorando técnico 1004: Movimiento del personaje	
Asunto	Movimiento del personaje.
Resumen	En función de input proporcionado por el jugador se le añade al personaje movimiento en esa dirección.
Factores causantes	-
Solución	Al detectar un input por parte del jugador relacionado con una las cuatro acciones de movimiento (delante, atrás, izquierda o derecha) se añade movimiento al personaje en la dirección especificada. Se diferencia movimiento delante-atrás e izquierda-derecha, siendo el resultado final la combinación de ambos movimientos.
Motivación	Solución propuesta por seguir el estándar.
Cuestiones abiertas	Debemos tener en cuenta que la solución generada deberá ser modificada en las siguientes iteraciones ya que no queremos que nuestro personaje pueda moverse en todo momento (por ejemplo, no queremos que el personaje se mueva hacia según qué sitios cuando esté agarrado a un saliente).
Alternativas	-

Cuadro 8.4: Memorando técnico 1004: Movimiento del personaje

Memorando técnico 1005: Rotación de cámara y personaje

Asunto	Rotación de cámara y personaje.
Resumen	Debemos permitir al usuario modificar la rotación de la cámara y, a su vez, alterar la rotación horizontal del personaje para que siempre esté mirando en la dirección de la cámara.
Factores causantes	-
Solución	La solución consta de dos partes muy diferenciadas. Por un lado cuando se detecta un cambio en los ejes asignados al movimiento de la cámara se añade rotación a la misma (se añade por separado respecto al eje X e Y, es decir, si el usuario está moviendo el eje X de la cámara con un valor de «10» hacia arriba y el eje Y de la misma con un valor de «5» hacia la izquierda se añadirá por una parte rotación hacia arriba de «10» puntos y por otra rotación hacia la derecha de «-5» puntos) y, por otro lado, marcamos como verdadera la variable booleana «bUsePawnControlRotation» de la cámara primera persona del jugador para que controle la rotación del personaje.
Motivación	Solución propuesta por seguir el estándar.
Cuestiones abiertas	Debemos tener en cuenta que la solución generada deberá ser modificada en las siguientes iteraciones ya que no queremos que nuestra cámara pueda moverse con total libertad en según qué circunstancias (por ejemplo, no queremos que se mueva cuando el personaje esté agarrando un saliente, ya que alteraría la posición del personaje).
Alternativas	-

Cuadro 8.5: Memorando técnico 1005: Rotación de cámara y personaje

Memorando técnico 1006: Rotación de cámara y personaje (para controladores)

Asunto	Señal de rotación proveniente de stick analógico.
Resumen	Existen dos métodos típicos de movimiento de cámara: el producido al desplazar un ratón y el que se origina al mover un stick analógico de un mando. En el primero no hay problema, puesto que el usuario es capaz de regular en todo momento la entrada que quiere dar dependiendo de la velocidad a la que mueva el ratón, y lo puede hacer prácticamente sin límites (o sin que esto resulte un problema). Al utilizar un stick analógico de un mando eso no es del todo así, puesto que el usuario puede elegir entre A) no mover el stick ($\text{input} = 0$), B) moverlo al máximo ($\text{input} = 1$) o C) moverlo en un estado intermedio (input mayor que 0 y menor que 1), por lo que debemos ponderar esa entrada proporcionada por el usuario, lo que sería equivalente a establecerle una sensibilidad.
Factores causantes	El factor causante en este caso no es una característica del producto, sino que se debe a la naturaleza de los sticks analógicos.
Solución	El remedio en este caso es ayudarnos de la solución obtenida para mover la cámara con el ratón y multiplicarle un «rate» (sensibilidad).
Motivación	Solución propuesta por seguir el estándar.
Cuestiones abiertas	-
Alternativas	-

Cuadro 8.6: Memorando técnico 1006: Rotación de cámara y personaje (para controladores)

Memorando técnico 1007: Sprint

Asunto	Sprint del personaje.
Resumen	Para implementar esta mecánica cambiaríamos el valor de andar del personaje a un número más elevado, para que se pudiera desplazar más rápido siempre que tuviésemos el botón indicado pulsado.
Factores causantes	-
Solución	Cuando la tecla de «Sprint» se accione cambiamos el parámetro «MaxWalkSpeed» del movimiento del personaje a 800. Cuando el jugador suelta dicha tecla el valor de la variable vuelve a su estado original.
Motivación	Solución propuesta por seguir el estándar.
Cuestiones abiertas	-
Alternativas	-

Cuadro 8.7: Memorando técnico 1007: Sprint

Memorando técnico 1008: Salto

Asunto	Salto del personaje.
Resumen	Una de las mecánicas básicas más fundamentales en el juego será la de salto: el personaje deberá saltar con precisión distintos obstáculos para llegar al final del nivel.
Factores causantes	-
Solución	Al accionar el botón asignado al salto y, si procede, el personaje empieza a saltar hasta que llegue al límite del salto o se suelte la tecla asignada.
Motivación	Solución propuesta por seguir el estándar.
Cuestiones abiertas	Muchas mecánicas futuras habrán uso de las mismas teclas destinadas para el salto, por lo que tendremos que estar atentos con ello.-
Alternativas	-

Cuadro 8.8: Memorando técnico 1009: Agachar

Memorando técnico 1009: Agachar	
Asunto	Mecánica de agachado del personaje.
Resumen	Para evitar distintos obstáculos el personaje podrá agacharse pulsando el botón indicado.
Factores causantes	-
Solución	Cuando la tecla «Agachar» se accione el personaje estéticamente se agachará y cambiaremos la altura de su capsula de colisión para que pueda acceder a zonas en las que agachado no podía. Al soltar el botón, el personaje volverá a ponerse de pie.
Motivación	Solución propuesta por seguir el estándar.
Cuestiones abiertas	La tecla usada para esta acción se volverá a usar con otras mecánicas, por lo que, de nuevo, tendremos que estar atentos en interacciones posteriores.
Alternativas	-

Cuadro 8.9: Memorando técnico 1009: Agachar

Memorando técnico 1010: Tiempo bala

Asunto	Tiempo bala.
Resumen	Nuestro personaje, cuando está potenciado (es decir, cuando encuentra un orbe dorado) podrá realizar una serie de acciones entre las que se encuentra el tiempo bala.
Factores causantes	Estar potenciado.
Solución	Cuando la tecla de tiempo bala se acciona, si el personaje está potenciado y aún le queda adrenalina podrá parar el tiempo hasta que la adrenalina se le agote, parando el tiempo del entorno en un 50% y el suyo propio en un 25%. Cuando la adrenalina se agote automáticamente se detendrá el tiempo bala.
Motivación	Solución propuesta por su sencillez.
Cuestiones abiertas	-
Alternativas	-

Cuadro 8.10: Memorando técnico 0010: Tiempo bala

Memorando técnico 1011: Teletransporte o «blink»	
Asunto	Teletransporte, «parpadeo» o «blink».
Resumen	Nuestro personaje, cuando está potenciado (de nuevo, cuando encuentra un orbe dorado) podrá realizar un teletransporte dentro del rango que se le permite.
Factores causantes	Estar potenciado.
Solución	En cada tick del juego, lanzamos una línea desde el vector dirección de la cámara activa del jugador y con una longitud predefinida. Cuando pulsamos la tecla de «blink», si tenemos suficiente adrenalina y la mencionada línea choca, se nos teletransportará al lugar de impacto.
Motivación	Solución propuesta por su sencillez.
Cuestiones abiertas	-
Alternativas	En vez de usar glstick podríamos calcular el punto de impacto cuando el usuario solicite hacer un teletransporte, pero esto no nos permitiría indicarle cuando es posible realizar un teletransporte (por lo tanto, tenemos que estar checando el entorno en todo momento).

Cuadro 8.11: Memorando técnico 0011: Teletransporte o «blink»

Memorando técnico 1012: Control de límites del mapa

Asunto	Caja de colisión para controlar los límites del mapa.
Resumen	Cuando el personaje abandone los límites del mapa, morirá.
Factores causantes	Abandonar los límites del mapa
Solución	Implementar una «trigger box» o caja de colisión que, en el evento «ActorEndOverlap» compruebe si el actor que ha salido de la caja de colisión es un personaje y lo mate.
Motivación	Solución propuesta por simplicidad.
Cuestiones abiertas	-
Alternativas	-

Cuadro 8.12: Memorando técnico 0012: Control de límites del mapa

Memorando técnico 1013: Control de máxima distancia de caída

Asunto	Caja de colisión para controlar que el personaje no sobreviva si cae desde demasiado alto.
Resumen	Cuando el personaje caiga a un sitio desde demasiado alto, morirá.
Factores causantes	Caer a un sitio profundo
Solución	Implementar una «trigger box» o caja de colisión que, en el evento «ActorBeginOverlap» compruebe si el actor que ha entrado en la caja de colisión es un personaje y lo mate.
Motivación	Solución propuesta por simplicidad.
Cuestiones abiertas	-
Alternativas	-

Cuadro 8.13: Memorando técnico 0013: Control de máxima distancia de caída

Memorando técnico 1014: Movimiento de piernas del personaje al girarse.

Asunto	Queremos que cuando el personaje se gire no «flete» sin animación, sino que actúe de una forma menos artificial.
Resumen	Cuando el sistema detecte algún input horizontal relacionado con la cámara (ya sea controlador o ratón) avisará al blueprint de animación del personaje que debe reproducir esa animación si se dan las circunstancias adecuadas.
Factores causantes	-
Solución	Crear un nuevo bind (asignación), ficticio que se active cuando se reciba alguno de los parámetros que necesitamos, independientemente de si se producen mediante controlador o ratón.
Motivación	Solución propuesta por simplicidad.
Cuestiones abiertas	-
Alternativas	-

Cuadro 8.14: Memorando técnico 0014: Movimiento de piernas del personaje al girarse.

8.3 IMPLEMENTACIÓN

1

ID	Descripción de la acción de alto nivel					
1004	Añade movimiento en la dirección del personaje o en la opuesta					
Métodos de alto nivel						
[void] MoveFoward (Value:float)						
Pasos						
1. Cuando se hace uso del enlace asignado al movimiento frontal / trasero del personaje (llamado «MoveFoward»), comprueba que el valor de 'Value' sea distinto de 0.						
1.1. Si es distinto de 0, añade movimiento equivalente al valor de 'Value' en la dirección del vector dirección del actor. Si el valor de 'Value' es positivo el personaje avanzará en la dirección indicada y si es negativo retrocederá.						
1.2. Si es igual a 0, no se hace nada.						
Métodos de bajo nivel necesarios						
#	Clase	Método	Mem. Téc.			
1.1	APawn	[void] AddMovementInput (Direction:FVector, Value:float)	N/A			

2

ID	Descripción de la acción de alto nivel					
1004	Añade movimiento a izquierda o derecha de la dirección del personaje					
Métodos de alto nivel						
[void] MoveRight (Value:float)						
Pasos						
1. Cuando se hace uso del enlace asignado al movimiento lateral del personaje (llamado «MoveRight»), comprueba que el valor de 'Value' sea distinto de 0.						
1.1. Si es distinto de 0, añade movimiento equivalente al valor de 'Value' en la dirección del vector derecha del actor. Por tanto, si el valor de 'Value' es positivo el personaje avanzará hacia la derecha y si es negativo hacia la izquierda.						
1.2. Si es igual a 0, no se hace nada.						
Métodos de bajo nivel necesarios						
#	Clase	Método	Mem. Téc.			
1.1	APawn	[void] AddMovementInput (Direction:FVector, Value:float)	N/A			

3

ID	Descripción de la acción de alto nivel					
1005	Añade rotación horizontal a la cámara					
Métodos de alto nivel						
[void] Turn (Value:float)						
Pasos						
1. Cuando se hace uso del enlace asignado al movimiento horizontal de la cámara (llamado «Turn»), se delega en el método «AddControllerYawInput» de la clase APawn, puesto que no es necesario definir una función en la clase de nuestro personaje para modificar o controlar alguna faceta del resultado.						
Métodos de bajo nivel necesarios						
#	Clase	Método	Mem. Téc.			
1	APawn	[void] AddControllerYawInput (Value:float)	N/A			

ID	Descripción de la acción de alto nivel					
1005	Añade rotación vertical a la cámara					
Métodos de alto nivel						
[void] LookUp (Value:float)						
Pasos						
1. Cuando se hace uso del enlace asignado al movimiento vertical de la cámara (llamado «LookUp»), se delega en el método «AddControllerPitchInput» de la clase APawn, puesto que no es necesario definir una función en la clase de nuestro personaje para modificar o controlar alguna faceta del resultado.						
Métodos de bajo nivel necesarios						
#	Clase	Método	Mem. Téc.			
1	APawn	[void] AddControllerPitchInput (Value:float)	N/A			

CAPÍTULO 8. PRIMERA ITERACIÓN

ID	Descripción de la acción de alto nivel					
1006	Añade rotación horizontal a la cámara para controladores					
Métodos de alto nivel						
[void] TurnAtRate (Rate:float)						
Pasos						
<p>1. Cuando se hace uso del enlace asignado al movimiento horizontal de la cámara proveniente de un mando o similar (llamado «TurnRate»), se delega en el método «AddControllerYawInput» de la clase APawn, pero esta vez le mandamos la multiplicación del (1) Rate que envía el usuario (porcentaje, entre «-1.0» y «1.0» que mueve, por ejemplo, el eje del joystick del mando asignado), (2) de la sensibilidad horizontal (definida como «BaseTurnRate») y (3) por el tiempo en el que se renderiza cada frame.</p>						
Métodos de bajo nivel necesarios						
#	Clase	Método	Mem. Téc.			
1	APawn	[void] AddControllerYawInput (Value:float)	N/A			
1	UWorld	[Float] GetDeltaSeconds ()	N/A			

ID	Descripción de la acción de alto nivel					
1006	Añade rotación vertical a la cámara para controladores					
Métodos de alto nivel						
[void] LookUpAtRate (Rate:float)						
Pasos						
<p>1. Cuando se hace uso del enlace asignado al movimiento vertical de la cámara proveniente de un mando o similar (llamado «LookUpRate»), se hace eso del método «AddControllerPitchInput» de la clase APawn, pero esta vez le mandamos la multiplicación del (1) Rate que envía el usuario (porcentaje, entre «-1.0» y «1.0» que mueve, por ejemplo, el eje del joystick del mando asignado), (2) de la sensibilidad vertical (definida como «BaseLookUpRate») y (3) por el tiempo en el que se renderiza cada frame.</p>						
Métodos de bajo nivel necesarios						
#	Clase	Método	Mem. Téc.			
1	APawn	[void] AddControllerPitchInput (Value:float)	N/A			
1	UWorld	[Float] GetDeltaSeconds ()	N/A			

ID	Descripción de la acción de alto nivel
1007	El personaje entra en modo sprint
Métodos de alto nivel	
[void] OnSprintPressed ()	
Pasos	
1. Mientras se hace uso del enlace «Sprint» se cambia la velocidad a la que se mueve el personaje («MaxWalkSpeed») al valor de la velocidad de correr (guardada en la constante «SprintSpeed»).	

ID	Descripción de la acción de alto nivel
1007	El personaje entra en modo caminar
Métodos de alto nivel	
[void] OnSprintReleased ()	
Pasos	
1. Cuando se deja de hacer uso del enlace «Sprint» se cambia la velocidad a la que se mueve el personaje («MaxWalkSpeed») al valor de la velocidad de andar (guardada en la constante «WalkSpeed»).	

ID	Descripción de la acción de alto nivel		
1008	El personaje salta		
Métodos de alto nivel			
[void] OnJump ()			
Pasos			
1. Cuando se hace uso del enlace «Jump» se llama a la función «Jump()» de la clase «ACharacter»			
Métodos de bajo nivel necesarios			
#	Clase	Método	Mem. Téc.
1	ACharacter	[void] Jump ()	N/A

CAPÍTULO 8. PRIMERA ITERACIÓN

ID	Descripción de la acción de alto nivel		
1008	El personaje deja de saltar		
	Métodos de alto nivel		
	[void] StopJumping ()		
	Pasos		
	1. Cuando se deja de hacer uso del enlace «Jump» se llama a la función «StopJumping()» de la clase «ACharacter»		
	Métodos de bajo nivel necesarios		
#	Clase	Método	Mem. Téc.
1	ACharacter	[void] StopJumping ()	N/A

ID	Descripción de la acción de alto nivel		
1009	El personaje se agacha		
	Métodos de alto nivel		
	[void] OnCrouchPressed ()		
	Pasos		
	1. Cuando se hace uso del enlace «Crouch» se llama a la función «Crouch()» de la clase «ACharacter»		
	Métodos de bajo nivel necesarios		
#	Clase	Método	Mem. Téc.
1	ACharacter	[void] Crouch ()	N/A

ID	Descripción de la acción de alto nivel		
1009	El personaje deja de agacharse		
	Métodos de alto nivel		
	[void] OnCrouchReleased ()		
	Pasos		
	1. Cuando se deja de hacer uso del enlace «Crouch» se llama a la función «UnCrouch()» de la clase «ACharacter».		
	Métodos de bajo nivel necesarios		
#	Clase	Método	Mem. Téc.
1	ACharacter	[void] UnCrouch ()	N/A

ID	Descripción de la acción de alto nivel		
1010	El personaje entra en tiempo bala o sale de él		
Métodos de alto nivel			
[void] OnTimeDilationToggle ()			
Pasos			
<p>1. Cuando se hace uso del enlace «TimeDilation» se llama a la función «TimeDilationToggle()» de «AMainCharacter» y se comprueba si el jugador está potenciado</p> <p>1.1.Si el tiempo bala está activo, procede a llamar a EndTimeDilation().</p> <p>1.2.Si, por el contrario, el tiempo bala está inactivo, procede a llamar a StartTimeDilation().</p>			
Métodos de bajo nivel necesarios			
#	Clase	Método	Mem. Téc.
1.1	AMainCharacter	[void] StartTimeDilation ()	1010
1.2	AMainCharacter	[void] EndTimeDilation ()	1010

ID	Descripción de la acción de alto nivel
1011	El personaje se teletransporta a un lugar cercano
Métodos de alto nivel	
[void] OnBlink ()	
Pasos	
<p>1. Cuando se hace uso del enlace «Blink» se llama a la función «OnBlink()» de la clase «AMainCharacter».</p> <p>2. Se comprueba si el personaje está potenciado y no está mirando a un punto demasiado lejano (fuera de rango):</p> <p>2.1. Si se cumple la colisión el jugador salta, se ejecuta un sonido y a continuación el teletransporte.</p> <p>2.2. Si no, no ocurre nada.</p>	

ID	Descripción de la acción de alto nivel
1012	El jugador abandona el área de juego
Métodos de alto nivel	
[void] ActorEndOverlap ()	
Pasos	
<p>1. Cuando se comprueba que algún actor ha abandonado el área de juego (ha dejado de estar dentro del área de juego):</p> <p>2.1. Si el actor es un jugador, lo mata, pasando a la pantalla de muerte.</p> <p>2.2. Si no es un jugador, no ocurre nada.</p>	
1	
ID	Descripción de la acción de alto nivel
1013	El jugador abandona el área de juego
Métodos de alto nivel	
[void] ActorBeginOverlap ()	
Pasos	
<p>1. Cuando se comprueba que algún actor ha impactado en el suelo después de una gran caída):</p> <p>2.1. Si el actor es un jugador, lo mata, pasando a la pantalla de muerte.</p> <p>2.2. Si no es un jugador, no ocurre nada.</p>	
2	
ID	Descripción de la acción de alto nivel
1014	Comunica a la animación que el personaje está rotando
Métodos de alto nivel	
[void] RotationTrick (Value:float)	
Pasos	
<p>1. Cuando se detecta rotación horizontal de la cámara de cualquier tipo (ya sea proveniente de una fuente que no necesita un «rateo», como un ratón, o de una que sí lo necesita, como un mando) se cambia el valor de «RotationInput» de la instancia de animación del protagonista.</p>	
3	

8.4 PRUEBAS

² TODO: Descripción de las pruebas realizadas al software

8.5 DESPLIEGUE

1

TODO: Breve resumen de cómo se han desplegado los cambios en el sistema de producción.

2

3

SEGUNDA ITERACIÓN

1

2 *D*urante esta iteración nos centraremos en el sistema de escalada, uno de los ejes prin-
3 *cipales del proyecto, e implementaremos acciones como agarrar salientes, saltar de*
4 *uno a otro o carreras por las paredes.*

9.1 CARACTERÍSTICAS A DESARROLLAR

1. Base del sistema de escalada (*).	2
■ Implementación del código de la base del sistema de escalada.	3
2. Mecánica: Agarrar un saliente (*).	4
■ Implementación del código de la mecánica.	5
■ Desarrollo de la animación del personaje relacionada con la mecánica.	6
3. Mecánica: Agarrar un saliente pequeño y escalarlo (*).	7
■ Implementación del código de la mecánica.	8
■ Desarrollo de la animación del personaje relacionada con la mecánica.	9
4. Mecánica: Escalar un saliente al que estemos agarrados (*).	10
■ Implementación del código de la mecánica.	11
■ Asignación de teclas y botones vinculados la mecánica.	12
5. Mecánica: Soltar un saliente (*).	13
■ Implementación del código de la mecánica.	14
■ Desarrollo de la animación del personaje relacionada con la mecánica.	15
6. Mecánica: Movernos por un saliente.	16
■ Implementación del código de la mecánica.	17
■ Desarrollo de la animación del personaje relacionada con la mecánica.	18
7. Mecánica: Saltar desde saliente lateralmente.	19
■ Implementación del código de la mecánica.	20
■ Desarrollo de la animación del personaje relacionada con la mecánica.	21
8. Mecánica: Mirar hacia atrás desde saliente.	22
■ Implementación del código de la mecánica.	23
■ Desarrollo de la animación del personaje relacionada con la mecánica.	24
9. Mecánica: Saltar desde saliente mirando hacia atrás	25

- 1 ■ Implementación del código de la mecánica.
- 2 ■ Desarrollo de la animación del personaje relacionada con la mecánica.

3 10. Mecánica: Escalada en pared (*).

- 4 ■ Implementación del código de la mecánica.
- 5 ■ Desarrollo de la animación del personaje relacionada con la mecánica.

6 11. Mecánica: Giro y salto durante escalada en pared.

- 7 ■ Implementación del código de la mecánica.
- 8 ■ Desarrollo de la animación del personaje relacionada con la mecánica.

9 Para la realización (o parte de la realización) de las mecánicas marcadas con un
10 asterisco (*) se siguieron las siguientes referencias:

- 11 ■ **Agarrar un saliente:** <https://youtu.be/4yjcwZLQq1E>
- 12 ■ **Escalar un saliente:** <https://youtu.be/H2xqW71Kkyw>
- 13 ■ **Soltar un saliente y escalada por pared:** <https://youtu.be/2vDjzr9EvUc>
- 14 ■ **Agarrar un saliente pequeño y escalarlo:** <https://youtu.be/fLLKgc0LDqc>

15 No obstante, estando basada buena parte de esta iteración del proyecto en dichas
16 referencias tenemos que tener en cuenta lo siguiente:

- 17 1. **Están implementadas en diferentes lenguajes de programación:** esta ya es una
18 diferencia de enorme de por sí, pero además no debemos olvidarnos que al tener
19 que lidiar con los blueprints de animaciones desde C++ todo se hace de manera
20 diferente, tanto es así que se necesitan instrucciones adicionales en el blueprint
21 de animaciones del personaje para que el sistema funcione correctamente.
- 22 2. **Cada una está realizada en diferentes versiones del motor:** lo que provocó que
23 instrucciones no funcionaran instrucciones y hubiese que buscar alternativas.
- 24 3. **Es una versión depurada en la que se han corregido numerosos fallos:** algunos
25 de ellos de enorme importancia.
- 26 4. Se han modificado, añadido y eliminado funcionalidades, así como que se ha
27 ampliado el sistema base con numerosas mejoras.

9.2 DISEÑO

Antes de empezar la fase de diseño, vamos a intentar centrarnos en la parte más importante del sistema de escalada: la base.

La base del sistema de escalada nos ofrece información en tiempo real del escenario y que usarán cada una de las funciones de escalada que se implementen para funcionar, por tanto, es el germen del que emanan todas ellas.

Para obtener información del escenario, lo que hacemos es trazar líneas por determinados canales desde el personaje o alguna de sus partes, con una determinada longitud, y ver si impactan con algún elemento del escenario y con cuál.

Esto se hace en cada tick del juego, por lo que está implementado totalmente en «C++» para optimizar su funcionamiento.

El sistema traza en todo momento, mientras que estemos cayendo y no estemos agarrados a nada (o bien estemos realizando una escalada en pared):

Trazadores de la base del sistema de escalada 1/2			
Nombre	Descripción	Tipo	Canal
Trazador de pared	Intenta buscar si tenemos una pared adecuada enfrente	Esfera	Especial
Trazador de techo	Intenta saber si hemos chocado con un techo al correr por la pared	Esfera	Visibilidad
Trazador de altura izquierdo	Trata de saber si el personaje podría emplazar la mano ahí izquierdo una vez agarrado	Esfera	Especial
Trazador de altura derecho	Igual que el anterior, pero referido a la otra mano	Esfera	Especial
Trazador de suelo	Trata de conocer si estamos demasiado cerca del suelo para escalar	Esfera	Visibilidad

Cuadro 9.1: Trazadores de la base del sistema de escalada 1/2

- ¹ Y cuando el personaje está agarrado algún saliente, se trazan:

Trazadores de la base del sistema de escalada 2/2			
Nombre	Descripción	Tipo	Canal
Trazador pie izquierdo	Intenta buscar si nuestro personaje puede emplazar el pie izquierdo	Esfera	Visibilidad
Trazador pie derecho	Intenta buscar si nuestro personaje puede emplazar el pie derecho	Esfera	Visibilidad

Cuadro 9.2: Trazadores de la base del sistema de escalada 2/2

- ² Además de estos, hay varios trazadores más que se implementan dentro de las propias funciones de escalada.

Memorando técnico 2002: Agarrar un saliente	
Asunto	Agarrar un saliente
Resumen	Se requiere que nuestro personaje se agarre a determinados salientes, previamente indicados dichos salientes.
Factores causantes	-
Solución	Cuando el personaje esté en una posición en la que pueda agarrar una de estas superficies especiales se quedará agarrado a ellas, sin necesidad de que el usuario interaccione más con el juego.
Motivación	Es la solución más simple.
Cuestiones abiertas	-
Alternativas	-

Cuadro 9.3: Memorando técnico 2002: Agarrar un saliente

Memorando técnico 2003: Agarrar un saliente pequeño para escalarlo

Asunto	Agarrar un pequeño saliente y escalarlo
Resumen	Se requiere que nuestro personaje cuando nuestro personaje se agarre con suficiente holgura a un objeto escalable, lo escale sin más, sin tener que caer y soportarse sobre sus manos.
Factores causantes	-
Solución	Cuando el personaje esté en una posición en la que pueda agarrar y saltar directamente una de las superficies especiales se agarrará a ella y la escalará, sin necesidad de que el usuario interaccione más con el juego.
Motivación	Es la solución más simple.
Cuestiones abiertas	-
Alternativas	-

Cuadro 9.4: Memorando técnico 2003: Agarrar un saliente pequeño para escalarlo

Memorando técnico 2004: Escalar un saliente al que estemos agarrados

Asunto	Escalar un saliente al que estemos agarrados
Resumen	Se requiere que nuestro personaje, estando agarrado de un saliente, puede saltarlo y subir al piso superior si puede.
Factores causantes	-
Solución	Cuando el personaje esté agarrado a un saliente, si pulsa el botón «saltar», puede hacer fuerzas con los pies y no hay ningún impedimento que permita al personaje emplazarse en la posición de destino, podrá saltar el saliente e incorporarse al siguiente nivel.
Motivación	Es la solución más simple.
Cuestiones abiertas	-
Alternativas	-

Cuadro 9.5: Memorando técnico 2004: Escalar un saliente al que estemos agarrados

Memorando técnico 2005: Soltar un saliente

Asunto	Soltar un saliente
Resumen	Se quiere que un personaje que esté agarrado a un saliente pueda soltarse.
Factores causantes	-
Solución	Cuando el personaje esté en un saliente y pulse el botón «agachar», se soltará del saliente.
Motivación	Es la solución más simple.
Cuestiones abiertas	-
Alternativas	-

Cuadro 9.6: Memorando técnico 2005: Soltar un saliente

Memorando técnico 2006: Movernos por un saliente

Asunto	Movernos por un saliente
Resumen	Se requiere que nuestro personaje pueda moverse libremente por los salientes si no hay nada que lo impida.
Factores causantes	-
Solución	Cuando no haya nada que lo impida el personaje podrá moverse por el saliente pulsando las teclas de movimiento izquierda y derecha.
Motivación	Es la solución más simple.
Cuestiones abiertas	-
Alternativas	-

Cuadro 9.7: Memorando técnico 2006: Movernos por un saliente

Memorando técnico 2007: Saltar desde saliente lateralmente

Asunto	Saltar desde saliente lateralmente
Resumen	Se requiere que nuestro personaje pueda saltar lateralmente cuando llegue a un extremo del saliente para agarrarse a otros.
Factores causantes	-
Solución	El personaje, una vez que llegue al límite del saliente y pueda hacer fuerza con los pies, hará un gesto con la mano indicando que puede saltar y si el usuario pulsa el botón «saltar» abandonará el vértice y se propulsará. Si hubiese otro saliente y lo estuviéramos mirando podríamos agarrarnos de nuevo a él.
Motivación	Es la solución más simple.
Cuestiones abiertas	-
Alternativas	-

Cuadro 9.8: Memorando técnico 2007: Saltar desde saliente lateralmente**Memorando técnico 2008: Mirar hacia atrás desde saliente**

Asunto	Mirar hacia atrás desde saliente
Resumen	Se requiere que nuestro personaje pueda volverse y mirar hacia atrás desde un saliente.
Factores causantes	-
Solución	El usuario, en cualquier momento mientras que el personaje esté agarrado a un saliente y pueda hacer fuerza con los pies, podrá girar al personaje y mirar hacia atrás usando la tecla de movimiento «atrás».
Motivación	Es la solución más simple.
Cuestiones abiertas	-
Alternativas	-

Cuadro 9.9: Memorando técnico 2008: Mirar hacia atrás desde saliente

Memorando técnico 2009: Saltar desde saliente mirando hacia atrás

Asunto	Saltar desde saliente hacia atrás
Resumen	Se requiere que nuestro personaje pueda realizar un salto hacia atrás del saliente, si nada lo impide, para caer en un determinado lugar o agarrarse a otros salientes.
Factores causantes	-
Solución	El usuario, en cualquier momento mientras que el personaje esté agarrado a un saliente y pueda hacer fuerza con los pies, podrá girar al personaje pulsando el botón de movimiento «atrás» y, sin soltarlo, pulsar el botón «saltar» para que el personaje de una patada a la pared, se impulse, y abandone el saliente.
Motivación	Es la solución más simple.
Cuestiones abiertas	-
Alternativas	-

Cuadro 9.10: Memorando técnico 2009: Saltar desde saliente mirando hacia atrás

Memorando técnico 2010: Escalada en pared

Asunto	Escalada en pared
Resumen	Se requiere que el personaje pueda escalar zonas propicias para la escalada.
Factores causantes	-
Solución	Cuando el personaje esté en frente y mirando una superficie escalable (no importa si está quieto, en movimiento, saltando o cayendo) si el usuario pulsa el botón de «salto» lo suficientemente cerca se iniciará la escalada.
Motivación	Es la solución más simple.
Cuestiones abiertas	-
Alternativas	-

Cuadro 9.11: Memorando técnico 2010: Escalada en pared

Memorando técnico 2011: Giro y salto durante escalada en pared

Asunto	Giro y salto durante escalada en pared
Resumen	Se quiere que el personaje pueda patear la pared impulsándose mientras está realizando una escalada por pared.
Factores causantes	-
Solución	En cualquier momento mientras que el personaje sigue escalando una pared el usuario podrá apretar el botón «saltar» para hacer que el personaje se gire y se impulse.
Motivación	Es la solución más simple.
Cuestiones abiertas	-
Alternativas	-

Cuadro 9.12: Memorando técnico 2011: Giro y salto durante escalada en pared

9.3 IMPLEMENTACIÓN

ID	Descripción de la acción de alto nivel		
2002	Agarrar un saliente		
	Métodos de alto nivel		
	[void] GrabLedge ()		
	Pasos		
	1. Comprueba la variable «bIsClimbingLedge»:		
	1.1. Si es verdadero no hace nada.		
	1.2. Si es falso:		
	1.2.1. Ajusta la caja de colisión de la cámara de primera persona (para evitar que la cámara traspase elementos del escenario).		
	1.2.2. Ajusta levemente la posición del personaje dentro de la cápsula para que se ajuste a la posición de la pared a la que nos agarraremos.		
	1.2.3. Cambia el valor de «bIsHanging» del blueprint encargado de animar al personaje a verdadero.		
	1.2.4. Cambia el modo de movimiento del personaje a volando.		
	1.2.5. Cambia el valor de «bIsHanging» de la clase a verdadero.		
	1.2.6. Mueve al personaje a una posición en la que la animación quede ajustada perfectamente.		
	1.2.7. Cuando termina para el movimiento del personaje inmediatamente.		
	1.2.8. Ajusta el valor de la variable «bCanDoWallRunning» a verdadero.		
	1.2.9. Ajusta el valor de la variable «bUseControllerRotationYaw» a falso, para que no controlemos la rotación del personaje con ella sino que permanezca siempre alineado con la pared.		
	1.2.10. Aplica restricciones de cámara.		
	Métodos de bajo nivel necesarios		
#	Clase	Método	Mem. Téc.
1.2.10.	AMainCharacter	[void] RestrictView (YawMin:float, YawMax:float, PitchMin:float)	N/A

ID	Descripción de la acción de alto nivel					
2003	Agarrar un saliente pequeño para escalarlo					
Métodos de alto nivel						
[void] KneeClimbLedge ()						
Pasos						
1. Comprueba la variable «bIsClimbingLedge»:						
1.1. Si es verdadero no hace nada.						
1.2. Si es falso:						
1.2.1. Llama a la función «ClimbingKnees()» del blueprint encargado de animar al personaje.						
1.2.2. Cambia el valor de «bIsClimbingLedge» de la clase a verdadero.						
1.2.3. Cambia el modo de movimiento del personaje a «volando».						
1.2.4. Mueve al personaje a una posición en la que la animación quede ajustada perfectamente.						
1.2.5. Cuando termina para el movimiento del personaje inmediatamente.						
1.2.6. Ajusta el valor de la variable «bUseControllerRotationYaw» a falso, para que no controlemos la rotación del personaje con ella sino que permanezca siempre alineado con la pared.						
1.2.7. Aplica restricciones de cámara.						
Métodos de bajo nivel necesarios						
#	Clase	Método	Mem. Téc.			
1.2.7.	AMainCharacter	[void] RestrictView (YawMin:float, YawMax:float, PitchMin:float)	N/A			

ID	Descripción de la acción de alto nivel		
2004	Escalar un saliente al que estemos agarrados (Inicio)		
Métodos de alto nivel			
[void] ClimbLedge ()			
Pasos			
1. Comprueba la variable «bIsClimbingLedge»:			
1.1. Si es verdadera no hace nada.			
1.2. Si es falsa:			
1.2.1. Cambia el modo de movimiento del personaje a «volando».			
1.2.2. Cambia el valor de «bIsClimbingLedge» de la clase a verdadero.			
1.2.3. Cambia el valor de «bIsHanging» de la clase a falso.			
1.2.1. Llama a la función «Climbing()» del blueprint encargado de animar al personaje.			
1.2.3. Cambia el valor de «bIsHanging» del blueprint encargado de animar al personaje a falso.			
1.2.5. Cuando termina para el movimiento del personaje inmediatamente.			
1.2.6. Ajusta el valor de la variable «bUseControllerRotationYaw» a falso, para que no controlemos la rotación del personaje con ella sino que permanezca siempre alineado con la pared.			
1.2.7. Aplica restricciones de cámara.			
Métodos de bajo nivel necesarios			
#	Clase	Método	Mem. Téc.
1.2.7.	AMainCharacter	[void] RestrictView (YawMin:float, YawMax:float, PitchMin:float)	N/A

1

ID	Descripción de la acción de alto nivel		
2004	Escalar un saliente al que estemos agarrados (Final)		
Métodos de alto nivel			
	[void] CompleteClimb ()		
Pasos			
1	<ol style="list-style-type: none"> 1. Cambia el modo de movimiento del personaje a «caminando». 2. Cambia el valor de «bIsClimbingLedge» de la clase a falso. 3. Cambia el valor de «bIsHanging» de la clase a falso. 4. Cambia el valor de «bIsWallRuning» de la clase a falso. 5. Cuando termina para el movimiento del personaje inmediatamente. 6. Ajusta el valor de la variable «bUseControllerRotationYaw» a verdadero, para volver a controlar la rotación del personaje con la cámara. 7. Aplica un reseteo en las restricciones de cámara. 8. Ajusta la caja de colisión de la cámara de primera persona (para evitar que la cámara traspase elementos del escenario). 9. Devuelve al personaje a su posición inicial dentro de la caja de colisión 		
Métodos de bajo nivel necesarios			
#	Clase	Método	Mem. Téc.
7.	AMainCharacter	[void] ResetRestrictView ()	N/A

ID	Descripción de la acción de alto nivel		
2005	Soltar un saliente		
Métodos de alto nivel			
[void] LeaveLedge ()			
Pasos			
<p>1. Cambia el modo de movimiento del personaje a «caminando».</p> <p>2. Cambia el valor de «bIsHanging» del blueprint encargado de animar al personaje a falso.</p> <p>3. Cambia el valor de «bCanBraceHang» del blueprint encargado de animar al personaje a falso.</p> <p>4. Cambia el valor de «GrabRightInput» del blueprint encargado de animar al personaje a 0.</p> <p>5. Cambia el valor de «GrabLookingRightInput» del blueprint encargado de animar al personaje a 0.</p> <p>6. Cambia el valor de «bIsGrabingAndLookingRear» del blueprint encargado de animar al personaje a falso.</p> <p>7. Cambia el valor de «bIsHanging» de la clase a verdadero.</p> <p>8. Cambia el valor de «bCanBraceHang» de la clase a falso.</p> <p>9. Cambia el valor de «bIsGrabbingLookingSide» de la clase a falso.</p> <p>10. Cambia el valor de «bIsGrabbingLookingRear» de la clase a falso.</p> <p>11. Ajusta el valor de la variable «bUseControllerRotationYaw» a verdadero, para volver a controlar la rotación del personaje con la cámara.</p> <p>12. Aplica un reseteo en las restricciones de cámara.</p> <p>13. Ajusta la caja de colisión de la cámara de primera persona (para evitar que la cámara traspase elementos del escenario).</p> <p>14. Devuelve al personaje a su posición inicial dentro de la caja de colisión</p>			
Métodos de bajo nivel necesarios			
#	Clase	Método	Mem. Téc.
12.	AMainCharacter	[void] ResetRestrictView ()	N/A

ID	Descripción de la acción de alto nivel
2006	Movernos por un saliente (PRIMERA PARTE)
Métodos de alto nivel	
[void] GrabLedgeMove ()	
Pasos	
<p>1. Comprobamos si el usuario intenta moverse lateralmente o está mirando hacia atrás:</p> <p>1.1. Si es verdadero:</p> <ul style="list-style-type: none"> 1.1.1. Detenemos el movimiento del personaje inmediatamente. 1.1.2. Cambia el valor de «GrabRightInput» del blueprint encargado de animar al personaje a 0. 1.1.3. Cambia el valor de «GrabLookingRightInput» del blueprint encargado de animar al personaje a 0. <p>1.2. Si por el contrario no se cumple:</p>	

1

ID	Descripción de la acción de alto nivel
2006	Movernos por un saliente (SEGUNDA PARTE PARTE)
Métodos de alto nivel	
[void] GrabLedgeMove ()	
Pasos	
1.2.1. Si el usuario intenta moverse hacia la izquierda:	
1.2.1.1. Trazamos una esfera un poco más a la izquierda de donde nos encontramos, buscando la pared, para ver si aún hay superficie a la que moverse:	
1.2.1.1.1. Si la esfera impacta:	
1.2.1.1.1.1. Cambia el valor de «GrabRightInput» del blueprint encargado de animar al personaje al valor del input que envía el usuario.	
1.2.1.1.1.2. Cambia el valor de «GrabLookingRightInput» del blueprint encargado de animar al personaje a 0.	
1.2.1.1.1.3. Comprobamos la variable «bCanBraceHang»:	
1.2.1.1.1.3.1. Si es verdadero:	
1.2.1.1.1.3.1.1. Le añadimos movimiento al personaje hacia la izquierda, pero dividimos entre 5 el valor que el usuario le quiere dar.	
1.2.1.1.1.3.2. Si es falso:	
1.2.1.1.1.3.1.1. Le añadimos movimiento al personaje hacia la izquierda, pero dividimos entre 8 el valor que el usuario le quiere dar.	
1.2.1.1.1.4. Cambia el valor de «bIsGrabbingLookingSide» de la clase a falso.	
1.2.1.1.2. Si la esfera no impacta:	
1.2.1.1.2.1. Detenemos el movimiento inmediatamente	
1.2.1.1.2.2. Lanzamos una esfera hacia la derecha del jugador para asegurarnos de que no hay obstáculos.	
1.2.1.1.2.2.1. Si la esfera impacta:	
1.2.1.1.2.2.1.1. Cambia el valor de «GrabRightInput» del blueprint encargado de animar al personaje a 0.	
1.2.1.1.2.2.1.2. Cambia el valor de «GrabLookingRightInput» del blueprint encargado de animar al personaje a 0.	
1.2.1.1.2.2.1.3. Cambia el valor de «bIsGrabbingLookingSide» de la clase a falso.	
1.2.1.1.2.2.2. Si la esfera no impacta:	
1.2.1.1.2.2.1.1. Cambia el valor de «GrabRightInput» del blueprint encargado de animar al personaje a 0.	
1.2.1.1.2.2.1.2. Cambia el valor de «GrabLookingRightInput» del blueprint encargado de animar al personaje al valor que está solicitando el jugador.	
1.2.1.1.2.2.1.3. Cambia el valor de «bIsGrabbingLookingSide» de la clase a verdadero.	

ID	Descripción de la acción de alto nivel
2006	Movernos por un saliente (TERCERA PARTE)
Métodos de alto nivel	
[void] GrabLedgeMove ()	
Pasos	
1.2.2. Si el usuario intenta moverse hacia la izquierda:	
1.2.2.1. Trazamos una esfera un poco más a la izquierda de donde nos encontramos, para ver si aún hay superficie a la que moverse:	
1.2.2.1.1. Si la esfera impacta:	
1.2.2.1.1.1. Cambia el valor de «GrabRightInput» del blueprint encargado de animar al personaje al valor del input que envía el usuario.	
1.2.2.1.1.2. Cambia el valor de «GrabLookingRightInput» del blueprint encargado de animar al personaje a 0.	
1.2.2.1.1.3. Comprobamos la variable «bCanBraceHang»:	
1.2.2.1.1.3.1. Si es verdadero:	
1.2.2.1.1.3.1.1. Le añadimos movimiento al personaje hacia la izquierda, pero dividimos entre 5 el valor que el usuario le quiere dar.	
1.2.2.1.1.3.2. Si es falso:	
1.2.2.1.1.3.1.1. Le añadimos movimiento al personaje hacia la izquierda, pero dividimos entre 8 el valor que el usuario le quiere dar.	
1.2.2.1.1.4. Cambia el valor de «bIsGrabbingLookingSide» de la clase a falso.	
1.2.2.1.2. Si la esfera no impacta:	
1.2.2.1.2.1. Detenemos el movimiento inmediatamente	
1.2.2.1.2.2. Lanzamos una esfera hacia la izquierda del jugador para asegurarnos de que no hay obstáculos.	
1.2.2.1.2.2.1. Si la esfera impacta:	
1.2.2.1.2.2.1.1. Cambia el valor de «GrabRightInput» del blueprint encargado de animar al personaje a 0.	
1.2.2.1.2.2.1.2. Cambia el valor de «GrabLookingRightInput» del blueprint encargado de animar al personaje a 0.	
1.2.2.1.2.2.1.3. Cambia el valor de «bIsGrabbingLookingSide» de la clase a falso.	
1.2.2.1.2.2.2. Si la esfera no impacta:	
1.2.2.1.2.2.1.1. Cambia el valor de «GrabRightInput» del blueprint encargado de animar al personaje a 0.	
1.2.2.1.2.2.1.2. Cambia el valor de «GrabLookingRightInput» del blueprint encargado de animar al personaje al valor que está solicitando el jugador.	
1.2.2.1.2.2.1.3. Cambia el valor de «bIsGrabbingLookingSide» de la clase a verdadero.	

ID	Descripción de la acción de alto nivel		
2007	<p>Saltar desde saliente lateralmente</p> <p>Métodos de alto nivel</p> <p>[void] JumpGrabbingSide ()</p> <p>Pasos</p> <p>1. Usamos el método «LeaveLedge()» para abandonar el saliente primero.</p> <p>2. Comprobamos el valor del input del movimiento izquierda-derecha:</p> <p>2.1. Si el usuario está pulsando el botón para moverse a la izquierda:</p> <p>2.1.1. Usamos el método «LaunchCharacter» de la clase «ACharacter» para lanzar al jugador hacia la izquierda (y un poco hacia arriba)</p> <p>2.2. Si el usuario está pulsando el botón para moverse a la derecha:</p> <p>2.2.1. Usamos el método «LaunchCharacter» de la clase «ACharacter» para lanzar al jugador hacia la derecha (y un poco hacia arriba)</p> <p>Métodos de bajo nivel necesarios</p>		
#	Clase	Método	Mem. Téc.
1.	AMainCharacter	[void] LeaveLedge ()	2005
2.1.1.	ACharacter	[void] LaunchCharacter (LaunchVelocity:FVector, bXYOverride:bool, bZOverride:bool)	N/A
2.2.1.	ACharacter	[void] LaunchCharacter (LaunchVelocity:FVector, bXYOverride:bool, bZOverride:bool)	N/A

ID	Descripción de la acción de alto nivel		
2008	Mirar hacia atrás desde saliente		
Métodos de alto nivel			
[void] GrabLedgeRear ()			
Pasos			
1. Comprobamos el valor de la variable «bIsGrabbingLookingRear»:			
1.1. Si es verdadera no hacemos nada.			
1.2. Si es falsa:			
1.2.1. Cambia el valor de «bIsGrabbingLookingRear» de la clase a verdadera.			
1.2.2. Cambia el valor de «bIsGrabingAndLookingRear» del blueprint encargado de animar al personaje a verdadero.			
1.2.3. Detenemos el movimiento inmediatamente			
1.2.4. Ajusta el valor de la variable «bUseControllerRotationYaw» a falso, para que no controlemos la rotación del personaje con ella sino que permanezca siempre alineado con la pared.			
1.2.5. Aplica restricciones de cámara.			
Métodos de bajo nivel necesarios			
#	Clase	Método	Mem. Téc.
1.2.5.	AMainCharacter	[void] RestrictView (YawMin:float, YawMax:float, PitchMin:float)	N/A

ID	Descripción de la acción de alto nivel		
2009	Saltar desde saliente mirando hacia atrás		
Métodos de alto nivel			
[void] JumpGrabbingRear ()			
Pasos			
<ol style="list-style-type: none"> 1. Usamos el método «LeaveLedge()» para abandonar el saliente primero. 2. Usamos el método «LaunchCharacter» de la clase «ACharacter» para lanzar al jugador en la dirección del vector normal de la pared a la que nos agarramos (y un poco hacia arriba) 3. Giramos 180 grados nuestro personaje 			
Métodos de bajo nivel necesarios			
#	Clase	Método	Mem. Téc.
2.	ACharacter	[void] LaunchCharacter (LaunchVelocity:FVector, bXYOverride:bool, bZOverride:bool)	N/A

ID	Descripción de la acción de alto nivel
2010	Escalada en pared
Métodos de alto nivel	
[void] JumpOrWallRunning ()	
Pasos	
1. Comprobamos el valor de la variable «bIsWallRunning»:	
1.1. Si es verdadera:	
1.1.1. Se llama a la función «WallRunningRearJump()», para realizar un salto durante la escalada en pared.	
1.2. Si es falsa:	
1.2.1. Se comprueba si el modo de movimiento del personaje es «volando»:	
1.2.1.1. Si es verdadero:	
1.2.1.1.1. No ocurre nada.	
1.2.1.2. Si es falso:	
1.2.1.2.1. Se comprueba si el modo de movimiento del personaje es «caminando»:	
1.2.1.2.1.1. Si es verdadero:	
1.2.1.2.1.1.1. Ajusta el valor de la variable «bCanDoWallRunning» a verdadero.	
1.2.1.2.1.2. Si es falso:	
1.2.1.2.1.2.1. No hace nada.	
1.2.1.2.2. Se traza una esfera hacia delante del jugador para saber si tenemos delante una pared apropiada para escalar:	
1.2.1.2.2.1. Si impacta y además la variable «bCanDoWallRunning» es verdadera:	
1.2.1.2.2.1.1. Cambia el valor de «bIsWallRunning» del blueprint encargado de animar al personaje a verdadero.	
1.2.1.2.2.1.2. Llama a la función «WallRunning()» del blueprint encargado de animar al personaje.	
1.2.1.2.2.1.3. Cambiamos el modo de movimiento del personaje a «volando».	
1.2.1.2.2.1.4. Movemos la cápsula y nuestro personaje hacia arriba verticalmente, poco a poco.	
1.2.1.2.1.1.5. Ajusta el valor de la variable «bIsWallRunning» a verdadero.	
1.2.1.2.1.1.6. Ajusta el valor de la variable «bCanDoWallRunning» a falso.	
1.2.1.2.2.1.7. Ajusta el valor de la variable «bUseControllerRotationYaw» a falso, para que no controlemos la rotación del personaje con ella sino que permanezca siempre alineado con la pared.	
1.2.1.2.2.1.8. Aplica restricciones de cámara.	
1.2.1.2.2.2. Si no:	
1.2.1.2.2.2.1. Saltamos mediante la función «Jump()».	

ID	Descripción de la acción de alto nivel		
2010	Giro y salto durante escalada en pared		
Métodos de alto nivel			
[void] WallRunningRearJump ()			
Pasos			
<p>1. Comprobamos que «bIsWallRunning» sea verdadero y «bIsClimbingLedge» falso:</p> <p>1.1. Si ocurre:</p> <p>1.1.1. Paramos la escalada en pared.</p> <p>1.1.2. Giramos 180 grados al personaje.</p> <p>1.1.3. Usamos el método «LaunchCharacter» de la clase «ACharacter» para lanzar al jugador en la dirección del vector normal de la pared a la que nos agarramos (y un poco hacia arriba)</p> <p>1.1.4. Ajusta el valor de la variable «bCanDoWallRunning» a verdadero.</p> <p>1.2. Si no ocurre:</p> <p>1.2.1. No pasa nada.</p>			
Métodos de bajo nivel necesarios			
#	Clase	Método	Mem. Téc.
1.1.3.	ACharacter	[void] LaunchCharacter (LaunchVelocity:FVector, bXYOverride:bool, bZOverride:bool)	N/A

¹ 9.4 PRUEBAS

9.5 DESPLIEGUE

1

TERCERA ITERACIÓN

1

² The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck
³ of his whole damn life – and one is as good as the other.

6

⁷ **T** ODO TODO TODO

10.1 CARACTERÍSTICAS A DESARROLLAR

1

10.2 DISEÑO

2

10.3 IMPLEMENTACIÓN

3

10.4 PRUEBAS

4

10.5 DESPLIEGUE

5

CUARTA ITERACIÓN

1

² The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck
³ of his whole damn life – and one is as good as the other.

6

⁷ **T** *ODO TODO TODO*

11.1 CARACTERÍSTICAS A DESARROLLAR

1

11.2 DISEÑO

2

11.3 IMPLEMENTACIÓN

3

11.4 PRUEBAS

4

11.5 DESPLIEGUE

5

QUINTA ITERACIÓN

1

² The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck
³ of his whole damn life – and one is as good as the other.

6

⁷ **T** ODO TODO TODO

12.1 CARACTERÍSTICAS A DESARROLLAR

1

12.2 DISEÑO

2

12.3 IMPLEMENTACIÓN

3

12.4 PRUEBAS

4

12.5 DESPLIEGUE

5

SEXTA ITERACIÓN

1

² The good parts of a book may be only something a writer is lucky enough to overhear or it may be the wreck
³ of his whole damn life — and one is as good as the other.

6

7 **T** ODO TODO TODO

13.1 CARACTERÍSTICAS A DESARROLLAR

1

13.2 DISEÑO

2

13.3 IMPLEMENTACIÓN

3

13.4 PRUEBAS

4

13.5 DESPLIEGUE

5

PARTE IV

CIERRE DEL PROYECTO

MANUAL DE USUARIO

1

2 *Durante este capítulo se muestran los distintos dispositivos de entrada que admite el*
3 *videojuego, los controles asociados a cada uno de ellos, la explicación detallada de*
4 *las opciones de los menús y el significado de cada elemento de los HUDs.*

14.1 CONTROLES

En esta sección se definen los enlaces entre las diferentes entradas al alcance del jugador y su traducción al universo del juego.

El producto ofrece al jugador dos métodos de entrada: «teclado y ratón» y «mando». Ambos métodos no son excluyentes entre sí, y si se dispone de los dos métodos de entrada en cualquier momento se podrá pasar de uno a otro, sin ser necesario reiniciar el videojuego.

14.1.1 Controles con teclado y ratón

Esquemas

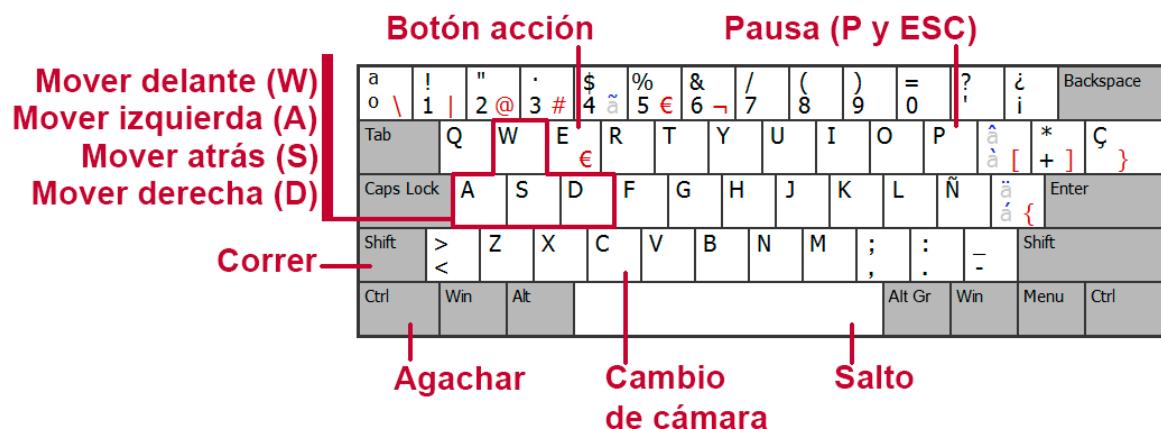


Figura 14.1: Diagrama de la asignación de acciones del teclado



Figura 14.2: Diagrama de la asignación de acciones del ratón

1 Listado detallado

- 2 ■ Tecla **W**: Desplazamiento del jugador hacia el frente.
 - 3 ■ Tecla **A**: Desplazamiento del jugador hacia atrás.
 - 4 ■ Tecla **S**: Desplazamiento del jugador hacia la izquierda.
 - 5 ■ Tecla **D**: Desplazamiento del jugador hacia la derecha.
 - 6 ■ **Eje X** del ratón: Regula el movimiento horizontal de la cámara.
 - 7 ■ **Eje Y** del ratón: Regula el movimiento vertical de la cámara.
 - 8 ■ Tecla **E**: Botón de acción.
 - 9 ■ **Barra espaciadora**: Salto / Wall running / Salto en wall running / Escalar o saltar
10 de saliente.
 - 11 ■ Tecla **Ctrl Izquierdo**: Agachado / Abandonar saliente.
 - 12 ■ Tecla **Shift Izquierdo**: Sprint.

 - 13 ■ **Botón izquierdo** del ratón: Blink (*).
 - 14 ■ **Botón derecho** del ratón: Comutador de tiempo bala (*).
- 15 **Nota:** Las acciones marcadas con (*) necesitan que el jugador tenga en su poder un
16 orbe dorado para que lleguen a producirse.

14.1.2 Controles con mando

La siguiente sección muestra los controles con mando, en concreto se usa como referencia un controlador de «Xbox 360» o «Xbox One» (puesto que las versiones de los controladores ambas consolas son muy similares). Esto se debe a que actualmente este es el estándar en PC, pero no hay motivos para que el software no pueda ser jugado con otro mando compatible diferente de los citados: simplemente los nombres de los botones, gatillos, etc. serán diferentes.

Esquema

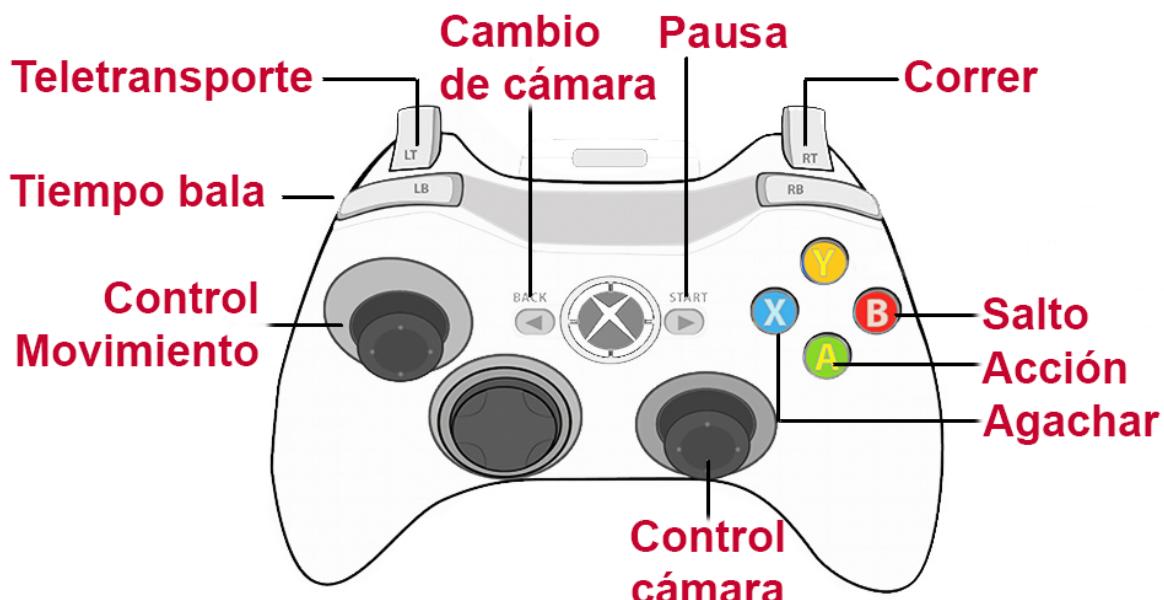


Figura 14.3: Diagrama de la asignación de acciones del controlador

Listado detallado

- Joystick izquierdo Eje Y+: Desplazamiento del jugador hacia el frente.
- Joystick izquierdo Eje Y-: Desplazamiento del jugador hacia atrás.
- Joystick izquierdo Eje X-: Desplazamiento del jugador hacia la izquierda.
- Joystick izquierdo Eje X+: Desplazamiento del jugador hacia la derecha.
- Joystick derecho Eje X: del ratón: Regula el movimiento horizontal de la cámara.
- Joystick derecho Eje Y: Regula el movimiento vertical de la cámara.
- Botón Start: Botón de pausa.

- ¹ ■ Botón **Back**: Cambio de cámara.
- ² ■ Botón **A**: Botón de acción.
- ³ ■ Botón **B**: Salto / Wall running / Salto en wall running / Escalar o saltar de saliente.
- ⁴
- ⁵ ■ Botón **X**: Agachado / Abandonar saliente.
- ⁶ ■ Disparador **RT**: Sprint.
- ⁷ ■ Disparador **LT**: Blink (*).
- ⁸ ■ Botón **LB**: Comutador de tiempo bala (*).

⁹ **Nota:** Las acciones marcadas con (*) necesitan que el jugador tenga en su poder un
¹⁰ orbe dorado para que lleguen a producirse.

14.2 MENÚ PRINCIPAL

El menú principal es nuestra pantalla de bienvenida al juego, en el que se nos brindan cuatro opciones: «Nuevo juego», «Cargar juego», «Ajustes» y «Salir a Windows».



Figura 14.4: Fotograma del menú principal del software

Como podemos apreciar las opciones que se muestran son bastante intuitivas, pero esto es lo que hace **en detalle** cada una de esas opciones al interaccionar con ellas:

- **Nuevo juego:** Inicia el juego desde el principio. **Si hubiese datos guardados anteriores, los borraría y empezaría una nueva partida.**
- **Cargar juego:** Carga la partida anterior. **Esta opción estará deshabilitada si no tenemos un punto de guardado creado.**
- **Ajustes:** Oculta el menú y muestra el menú de ajustes (Ver sección §14.4).
- **Salir a Windows:** Cierra el videojuego y vuelve a Windows.

14.3 MENÚ DE PAUSA

Este menú es prácticamente idéntico al menú principal, pero con algunas opciones extras: «Cerrar menú» y «Volver al menú principal».



Figura 14.5: Fotograma del menú de pausa del software

Las opciones vuelven a ser muy intuitivas pero, de nuevo, en detalle:

- **Cerrar menú:** Cierra el menú y reanuda el juego.
- **Volver al menú principal:** Carga el mapa del menú principal como si entrásemos nuevamente al juego.

14.4 MENÚ DE AJUSTES

Esta opción es accesible vía menú principal o menú de pausa, haciendo click en «Settings» o «Ajustes» (dependiendo del idioma en el que el software esté configurado).



Figura 14.6: Fotograma del menú de ajustes del software

Estas son las opciones del menú de ajustes, en detalle:

■ Idioma

- **Descripción:** Nos permite elegir el idioma en el que se mostrarán todos los mensajes del juego.
- **Opciones disponibles:** Inglés y español.

■ Relación de aspecto

- **Descripción:** Nos permite visualizar las opciones de resolución compatibles con la relación de aspecto que seleccionemos.
- **Opciones disponibles:** «4:3», «16:9», «16:9+» y «16:10», siendo «16:9+» las resoluciones con ese aspect ratio mayores que Full HD (1920x1080).

1 ■ Resolución

- 2 • **Descripción:** Muestra las opciones de resolución que concuerden con el ajuste seleccionado en la opción anterior. Al hacer click se cambiará a dicha resolución.
- 3 • **Opciones disponibles (según la relación de aspecto):**
- 4 ○ «**4:3**»: «640x480», «800x600», «1024x768» y «1280x1024».
- 5 ○ «**16:9**»: «1280x720», «1366x768» y «1920x1080».
- 6 ○ «**16:9+**»: «2560x1440», «3840x2160» y «7680x4320».
- 7 ○ «**16:10**»: «720x480», «1280x768», «1280x800» y «1680x1050».

10 ■ Pantalla completa

- 11 • **Descripción:** Nos permite alternar entre modo ventana y pantalla completa.
- 12 • **Opciones disponibles:** «Sí» y «No».

13 ■ Opciones gráficas

- 14 • **Descripción:** Nos permite ajustar la calidad visual del juego (así como, indirectamente, su rendimiento).
- 15 • **Opciones disponibles:** «Bajo», «Medio», «Alto» y «Ultra», siendo «Bajo» la que peor calidad visual tiene (pero mejor rendimiento) y «Ultra» la que mejor calidad ofrece (y, por ende, peor rendimiento).

14.5 HUD DEL USUARIO

1

El HUD del usuario es el que se le muestra al jugador durante el tiempo de juego.

2

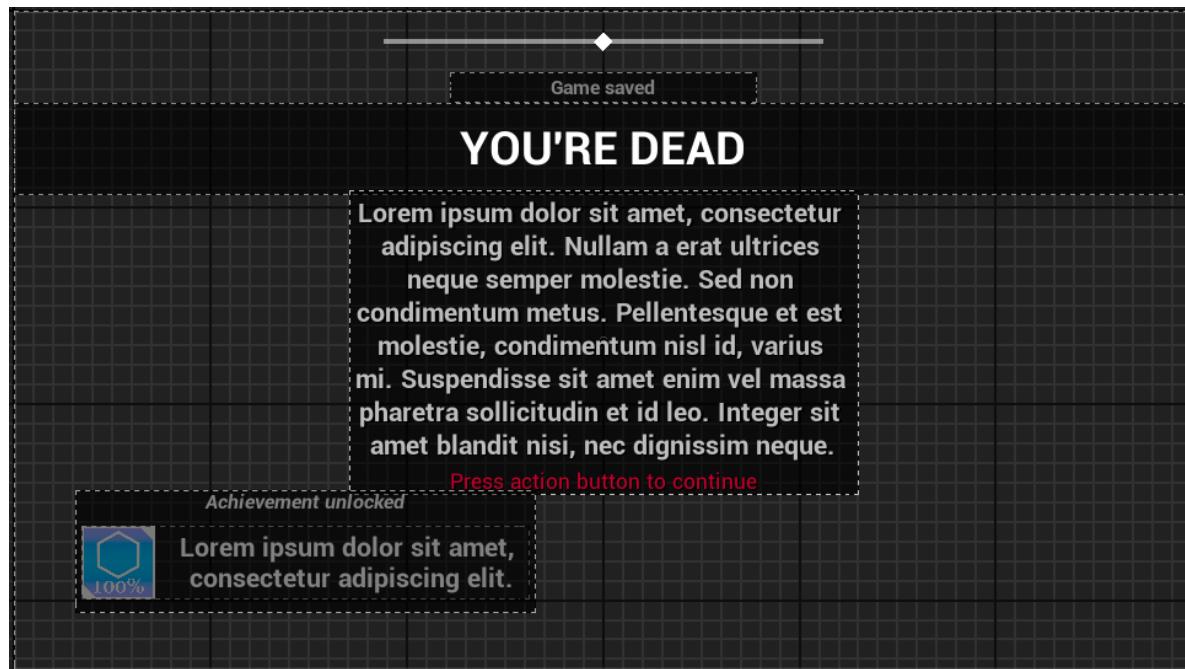


Figura 14.7: Instantánea mostrando el widget blueprint del HUD del usuario, en el que se pueden observar todos los elementos que se pueden mostrar en pantalla

Este HUD se compone de los siguientes elementos:

3

- **Punto de referencia:**

4

- **Tipo:** Imagen.
- **Funcionalidad:** Imagen (punto) de referencia que se usa principalmente para evitar el «motion sickness» o «cinetosis». En otras palabras, para suavizar mareos en jugadores con propensión a tenerlos.
- **Posición:** Equidistante de todos los lados de la pantalla.
- **Visibilidad:** Siempre.

5

6

7

8

9

10

1 ■ Indicador de habilidad «blink»:

- 2 • **Tipo:** Imagen.
- 3 • **Funcionalidad:** Indica cuando el jugador puede ejecutar la habilidad blink.
- 4 • **Posición:** Alrededor del punto de referencia.
- 5 • **Visibilidad:** Sólo cuando el personaje pueda realizar la habilidad blink y
6 esté mirando a una superficie propicia para ello.

7 ■ Indicador de adrenalina:

- 8 • **Tipo:** Barra.
- 9 • **Funcionalidad:** Muestra la cantidad de adrenalina restante.
- 10 • **Posición:** Adyacente al límite inferior de la pantalla.
- 11 • **Tamaño:** Ancho de la pantalla.
- 12 • **Visibilidad:** Se muestra si la adrenalina no está cargada por completo.



Figura 14.8: Captura del juego que muestra distintos componentes del hud: punto de referencia (centro), indicador de «blink» (alrededor de éste último), brújula (parte superior) y barra de adrenalina (parte inferior)

■ Brújula / indicador de fin de nivel:

- **Tipo:** Brújula.
- **Funcionalidad:** Muestra la dirección en la que se encuentra el cristal de telettransportación para avanzar por los diferentes mapas del juego.
- **Posición:** Parte superior de la pantalla.
- **Tamaño:** Aproximadamente el 50% de la pantalla.
- **Visibilidad:** Se muestra siempre que haya un cristal de nivel en el mapa.
- **Nota:** Si el cristal está cerca del campo de visión del personaje se mostrará un diamante en la brújula y estará en una posición u otra dentro de la propia barra de la brújula dependiendo de la posición del cristal respecto al centro del campo de visión del personaje. Si se encuentra lejos del campo de visión del jugador por la izquierda el «diamante» se situará en el extremo izquierdo de la brújula, y su forma cambiará a una flecha apuntando hacia la izquierda. Lo propio pasará cuando se encuentre en la parte derecha del campo de visión: esta vez el «diamante» se situará en el extremo derecho y su forma será una flecha apuntando a la derecha.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

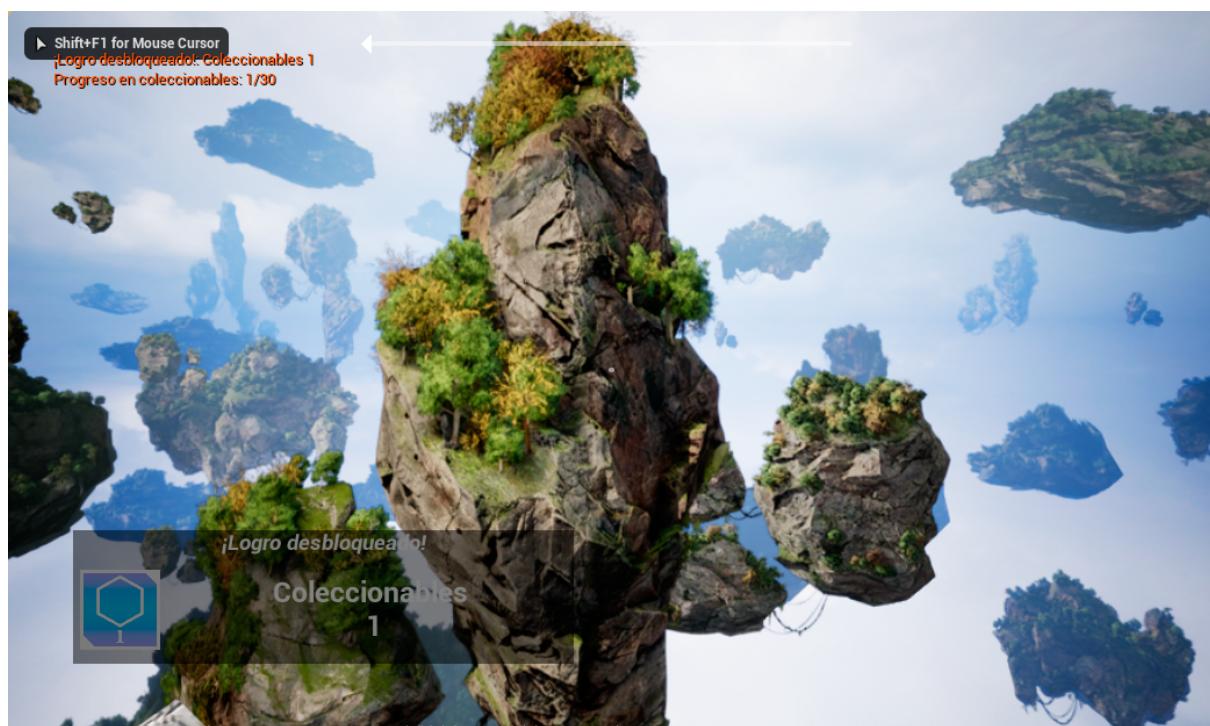


Figura 14.9: Captura del juego tomada durante la obtención de un logro

1 ■ Recuadro de logro:

- 2 • **Tipo:** Caja y texto.
- 3 • **Funcionalidad:** Caja que muestra un texto con información correspondiente
4 al logro obtenido.
- 5 • **Posición:** Parte inferior izquierda de la pantalla.
- 6 • **Visibilidad:** Se mostrará, junto a una animación, al obtener el logro y des-
7 aparecerá, con una animación de igual manera, en unos breves segundos.

8 ■ Recuadro de tutorial:

- 9 • **Tipo:** Caja y texto.
- 10 • **Funcionalidad:** Caja que muestra un texto con información correspondiente
11 al tutorial que está en ese momento activo.
- 12 • **Posición:** Centro de la pantalla.
- 13 • **Tamaño:** Alrededor de un 50% de la pantalla.
- 14 • **Visibilidad:** Se mostrará siempre que el personaje esté inmerso en un tuto-
15 rial, mostrando el texto relativo a cada tutorial.

16 ■ Mensaje de fin de juego:

- 17 • **Tipo:** Texto.
- 18 • **Funcionalidad:** Muestra el mensaje de fin de juego.
- 19 • **Posición:** Parte superior de la pantalla, centrado.
- 20 • **Tamaño:** Ocupando toda la pantalla horizontalmente.
- 21 • **Visibilidad:** Aparecerá al alcanzarse cualquiera de las condiciones de fin de
22 juego.

23 ■ Recuadro de juego guardado:

- 24 • **Tipo:** Caja y texto.
- 25 • **Funcionalidad:** Caja que muestra un texto con la advertencia de que el juego
26 ha sido guardado.
- 27 • **Posición:** Parte superior de la pantalla, centrado.
- 28 • **Visibilidad:** Se mostrará, junto a su animación, cuando el jugador llegue a
29 un punto de control, con una animación también, en unos breves segundos.

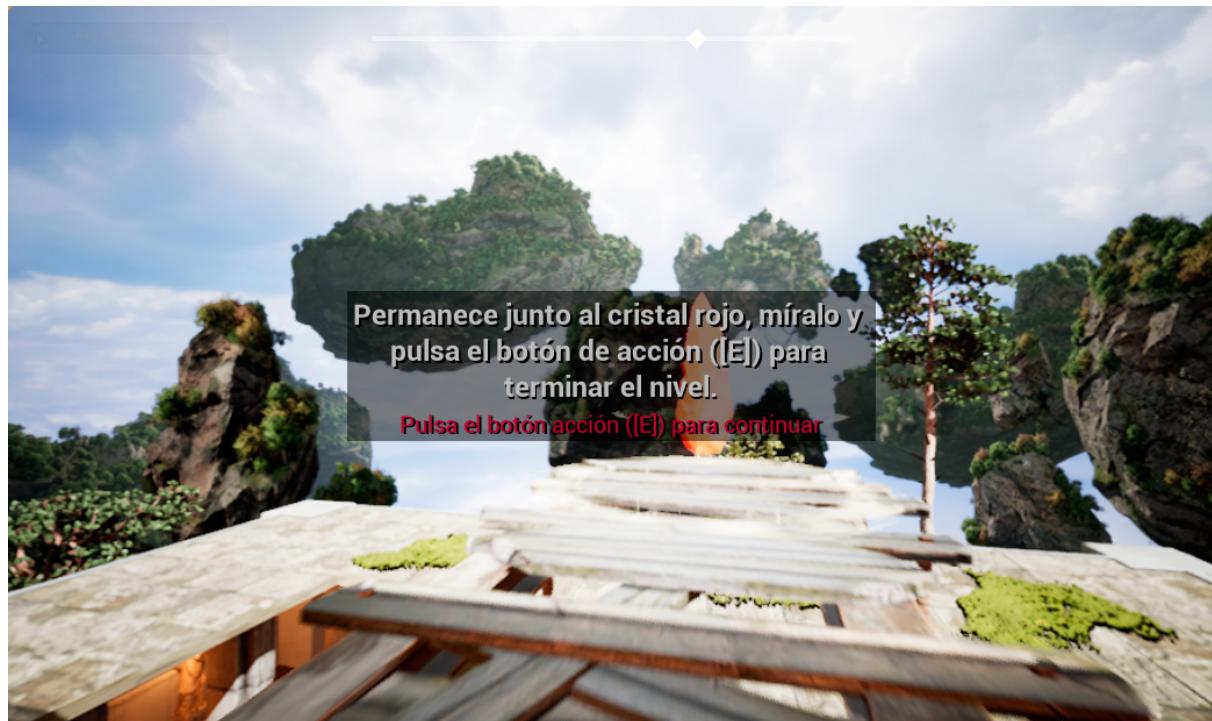


Figura 14.10: Captura del juego tomada durante un tutorial

CONCLUSIONES

1

2 *n el último capítulo del cuerpo del documento, y a modo de clausura del mismo, se*
3 *plasman a modo de memoria las conclusiones extraídas de la realización del proyecto,*
4 *una pieza clave en la ingeniería software, lo que nos será de una inmensa utilidad en*
5 *proyectos similares futuros.*

15.1 INFORME POST-MORTEM

Un informe post-mortem es aquel que se elabora a la conclusión de un proyecto y enumera los aciertos y problemas que han surgido al llevar a cabo el mismo, con el objetivo principal de aprender de dichos aciertos y errores en futuros proyectos del mismo tipo que se vayan a realizar.

Este tipo de informes son de una enorme utilidad y su realización está más que recomendada en el mundo de la ingeniería software. Sobre todo son de utilidad para personas que no estuvieron involucradas en ese proyecto (o lo estaban, pero desempeñando un rol diferente) ya que normalmente el equipo que se encargó de realizar el proyecto habrá pasado por situaciones similares a las que nos encontraremos si queremos llevar a cabo un proyecto del mismo ámbito, suponiendo un gran ahorro en tiempo: lo que por tanto se traduce directamente en un gran ahorro de dinero o recursos.

Durante las siguientes secciones detallaremos los aspectos más relevantes que han ido bien del proyecto, los que han ido mal, lo que se ha aprendido y lo que se mejoraría de cara a nuevos proyectos, así como los aspectos del proyecto que podrían quedar pendientes o son ampliables en proyectos futuros.

15.1.1 Lo que ha ido bien

Planificación y presupuesto

La importancia de una buena planificación es vital, ya que repercute directamente en el coste de un proyecto, pero lo es aún más en un proyecto de una naturaleza completamente nueva como es el caso.

En este caso, salvo en las excepciones explicadas en el propio capítulo de planificación (Ver sección §4.1) todo ha transcurrido de acuerdo a lo esperado y los retrasos se han podido subsanar perfectamente con el colchón de tiempo que se fijó durante la fase de planificación.

Además de esto, se ha conseguido evitar prácticamente por completo el denominado «síndrome del 90%» [14], aquel que aparece en proyectos de una considerable envergadura cuando se acerca su final y que nos hace creer que estamos a punto de finalizar el proyecto pero surgen nuevas tareas sin cesar, quedando el software estancado debido a esto durante un largo periodo de tiempo.

¹ **Código «C++» frente a «blueprints»**

² Una de las principales apuestas del proyecto ha sido la utilización del lenguaje de
³ programación C++ frente a la programación gráfica o visual en blueprints para las ta-
⁴ reas más complejas, es decir, principalmente aquellas tareas que hicieran uso reiterado
⁵ del tick para calcularse.

⁶ Esta decisión aportó inmediatamente un mayor rendimiento [6], frente a pequeños
⁷ prototipos que se realizaron antes del proyecto, así como una ventaja muy clara y ne-
⁸ cesaria que es el uso del control de versiones, en este caso «git», para tener un control
⁹ sobre los cambios en nuestro software.

¹⁰ Sin embargo, y aunque el balance final es claramente positivo, durante el desarrollo
¹¹ del proyecto se hicieron palpables algunas carencias que pusieron en entredicho esta
¹² opción y que son dignas de mención:

- ¹³ ■ La mayoría de conocimiento que circula por la red está en blueprints, tanto es así
¹⁴ que incluso hay diversas carencias en la documentación oficial de C++ de Epic
¹⁵ Games frente a la documentación de blueprints, que es mucho más completa.
¹⁶ Esto ha ocasionado que aprender a programar Unreal Engine en C++ fuese una
¹⁷ tarea mucho más costosa y tediosa de lo que en un principio debería.
- ¹⁸ ■ Mientras que si usamos blueprints podemos acceder a variables, funciones o cla-
¹⁹ ses definidas en C++ sin ningún tipo de problema no ocurre lo mismo al con-
²⁰ trario: para modificar una variable, función o clase de un blueprint no podemos
²¹ hacerlo directamente, sino que tenemos, por ejemplo, hacer que el blueprint her-
²² rede de una clase «cpp» y acceder o trabajar con dicho «cpp». Esto es un aspec-
²³ to muy a tener en cuenta puesto que el uso de blueprints en Unreal Engine es
²⁴ prácticamente obligatorio para tareas como las de animación de un personaje o
²⁵ la implementación de inteligencia artificial.
- ²⁶ ■ El tiempo de compilación de un blueprint se sitúa por debajo del segundo (en
²⁷ el equipo utilizado para el desarrollo de este proyecto), sin embargo, compilar
²⁸ código «C++» toma mucho más tiempo, llegando a tardar minutos en muchos
²⁹ casos, lo que laстра tanto el tiempo empleado para realizar una tarea como el
³⁰ ritmo de trabajo en sí, que se ve seriamente resentido ocasionando un desánimo
³¹ en muchas ocasiones que debería ser evitable.
- ³² ■ Los blueprints, además de ser más intuitivos si no se domina a la perfección Un-
³³ real Engine, ofrecen mucha más comodidad a la hora de trabajar con elementos

3D, puesto que disponen de un visor que permite ver la posición y rotación de dichos elementos y ajustarlos con unos simples clicks del ratón. Sin embargo, si trabajamos con elementos 3D en C++ no sólo deberemos compilar cada vez que queramos ver el resultado, sino que incluso a veces será necesario reiniciar Unreal Engine en su totalidad puesto que no siempre se actualizan los cambios.

Construcciones periódicas del proyecto durante el desarrollo

Como se fijó en la metodología, cada vez que se implementara una nueva funcionalidad se pasaría a la construcción del proyecto. Esto ha sido tremadamente útil durante el desarrollo del proyecto principalmente porque no toda la funcionalidad implementada que se prueba en el editor (y que parece funcionar correctamente) funciona realmente en la versión construida / exportada del proyecto.

Además, cabe destacar que Unreal Engine es especialmente poco explícito con sus errores en algunos casos (sobre todo, en cuanto a construcción del proyecto), siendo un auténtico quebradero de cabeza encontrar el problema.

Sin estas construcciones periódicas probablemente se hubiese llegado a una entrega de proyecto mucho más apurada, en la que muchas de las funcionalidades que parecían funcionar correctamente en el editor de Unreal Engine dejarían de funcionar sin motivo aparente, teniendo que revisar código que para más inri podría ser de hace meses.

Incorporación de animaciones profesionales

Otra de las grandes apuestas del proyecto ha sido optar en reiteradas ocasiones por las animaciones profesionales de Mixamo.

Aunque la incorporación de estas animaciones requiere de un procesamiento, como se ha visto en secciones anteriores del documento, ya que no se adaptan al esqueleto del personaje utilizado, el resultado final ha merecido la pena, puesto que están muy por encima de las animaciones que se podrían haber desarrollado desde cero.

15.1.2 Lo que ha ido mal

2 Dificultad en la detección de errores

3 En un producto complejo de estas características, en el que se toma información
 4 en tiempo real de un escenario en tres dimensiones y en el que, por ende, influyen
 5 gran cantidad de variables en cada acción hace que sea especialmente difícil detectar
 6 el origen de un problema cuando se produce uno.

7 Antes del cierre del proyecto, un grupo de personas ha podido probar el producto,
 8 en distintas etapas del desarrollo. Sin embargo, cada vez que se producía un error, y
 9 pese a que numerosas de estas personas estaban familiarizadas con el mundo de la
 10 informática o los videojuegos, en contadas ocasiones podían dar una pista sobre el
 11 error, y mucho menos dar una lista de acciones que llevaron a cabo para que el error se
 12 produjese.

13 Como se comentaba previamente esto se debe a la complejidad del producto, que
 14 dista enormemente de los fallos que puede arrojar un software más convencional, en
 15 los que si alguien encuentra un error al llenar un formulario, acceder a una deter-
 16 minada URL, etc. con total seguridad, al menos muchos de ellos, nos podrán dar al
 17 menos una pista de cómo se originó el error. Esto hace que sea especialmente difícil la
 18 reproducción de errores, algo que en muchas circunstancias es vital para solucionarlo,
 19 lo que puede lastrar seriamente el desarrollo del software aunque afortunadamente en
 20 el caso de este proyecto no ha llegado a tal extremo.

21 Cabe destacar un error en especial que persiguió al software desde la segunda ite-
 22 ración: en muy contadas ocasiones (estamos hablando de alrededor de un 1% de las
 23 veces) el personaje se agarraba a un saliente y empezaba a caer poco a poco en vez de
 24 quedarse a la misma altura del saliente, pero sin embargo la animación del persona-
 25 je lo mostraba como agarrado, la variable que controla esto estaba en orden e incluso
 26 podíamos soltarnos del saliente o realizar alguna otra acción si las comprobaciones
 27 necesarias se daban. Algunas personas completaron el videojuego sin que este error
 28 se reprodujese pero otras experimentaron este bug sin que supieran describir mínima-
 29 mente cómo llegaron a él. Tras algunas horas de testeо se llegó a la conclusión de que
 30 el error sólo se producía cuando llegábamos a la acción de agarrar el saliente prove-
 31 nientes de una escalada en pared: esa acción contiene una instrucción que fija el modo
 32 de movimiento del personaje en caída y por algún motivo seguía ejecutándose des-
 33 pués de para la escalada en pared. Una simple instrucción «if», que controlase que el
 34 personaje no estuviera agarrado a un saliente antes de fijar el modo de movimiento en

caída, bastó para solucionar el problema que podría haberse solucionado mucho antes con las indicaciones adecuadas.

El riesgo de las actualizaciones

Es bien conocido en el desarrollo software que actualizar nuestras aplicaciones de desarrollo entraña una serie de riesgos (así como, en muchos casos, de grandes ventajas).

Durante el desarrollo de este producto se ha intentado ir con pies de plomo en lo referido a actualizar nuestras herramientas:

- Antes del lanzamiento de las versiones en Unreal Engine, de igual manera que ocurre con muchos productos software, antes del lanzamiento final dichas versiones pasan por versiones «previews» o versiones candidatas al lanzamiento. De ninguna manera se pensó en actualizar a estas versiones.
- Para asegurarnos aún más, se ha evitado actualizar a las versiones 4.XX.0, para cerciorarnos de que la nueva versión del motor tenía ya cierto rodaje y evitar aún más los fallos que se hayan podido ocasionar con la actualización del motor.

Durante el desarrollo, la versión del motor se ha actualizado en dos ocasiones:

1. Como se explicará más adelante en la sección «trabajos futuros» no toda la funcionalidad inicial pudo ser implementada con la versión de Unreal Engine en la que se inició el desarrollo, la versión 4.15.1, sino que para añadir luz volumétrica para implementar una funcionalidad relacionada con un enemigo se optó por actualizar Unreal Engine a la versión 4.16.1, que actualizaba el motor añadiendo esta característica.
2. Igualmente se optó por actualizar a la nueva versión 4.17.1, esta vez por motivos de rendimiento, que preocupaban en la etapa final del desarrollo y que se han subsanado añadiendo opciones gráficas para adaptar el motor a cada equipo (ya que por defecto, al exportar el proyecto, se comprobó que el motor establecía las opciones de calidad gráfica más altas, y eso en un motor que aspira a estar en la vanguardia visual, como lo es Unreal Engine, hace que muchos dispositivos tengan enormes problemas de rendimiento).

La actualización a la versión 4.16.1 fue sin problemas, pero con la actual versión 4.17.1 se han experimentado diversos errores que, si bien no son graves, son evitables

¹ en el proyecto. Entre otros, con la actualización a esta versión se han podido detectar:
² valores por defecto de variables que se guardan pero sólo hasta que se cierra el editor
³ de Unreal Engine (siendo necesario guardar y cerrar Unreal entre 5 y 10 veces para que
⁴ los valores finalmente se guarden, algo fuera de toda lógica aparente), avisos de imposi-
⁵ sibilidad de guardar a la hora de intentar salvar escenarios pero sin embargo guardan
⁶ correctamente, etc.

⁷ De igual manera, tenemos que tener en cuenta que muchos de los assets o plugins
⁸ que estemos usando pueden dejar de funcionar. Fue el caso de «Ocean plugin», que
⁹ tras la actualización a la versión 4.17.1 dejó de funcionar y fue eliminado del proyecto.
¹⁰ Esto, ya se previó antes de la actualización y no es un contratiempo puesto que sólo se
¹¹ usaba con motivos estéticos en el menú y se prefirieron las mejoras de optimización (y
¹² aliviar el espacio que ocuparía el proyecto) antes que proseguir con este plugin, pero
¹³ igualmente es algo que hay que tener en cuenta.

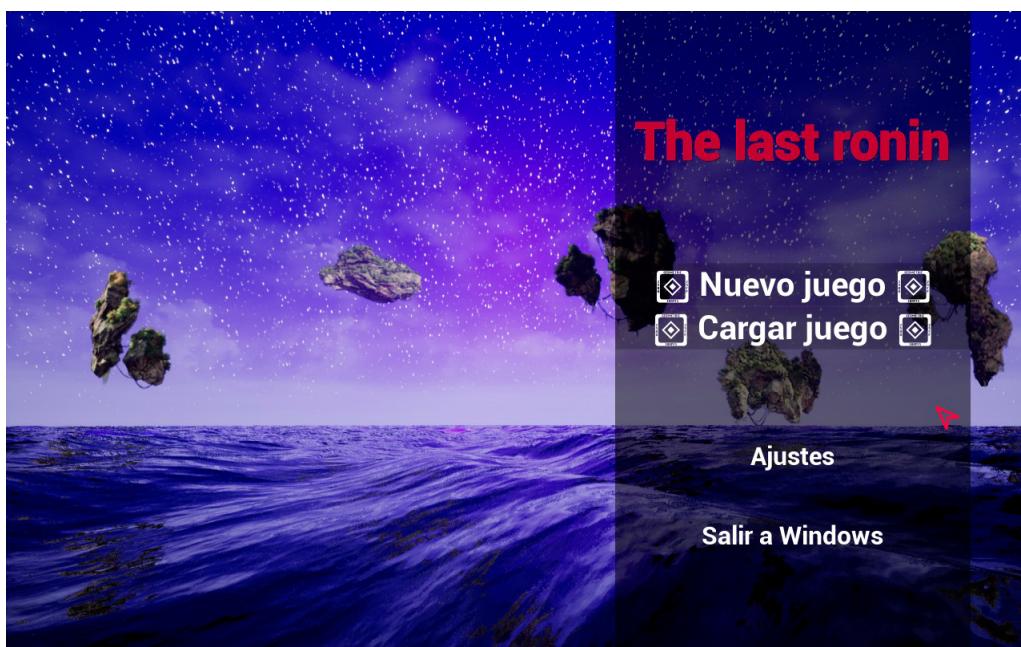


Figura 15.1: Captura del menú principal durante versiones tempranas del desarrollo

¹⁴ Afortunadamente en este caso estos fallos no fueron importantes, pero sin duda
¹⁵ podrían haber ido a más y es un punto negativo que sin duda hay que tener presente
¹⁶ de cara al futuro.

¹⁷ Un videojuego requiere aptitudes muy dispares entre sí

¹⁸ Llevar a buen puerto un producto software con las características de un videojuego
¹⁹ requiere no sólo habilidades de programación, que sin duda juegan un papel funda-

mental ya que inciden directamente en la jugabilidad, sino aptitudes muy dispares entre sí que hacen que se antoje muy difícil que recaigan en una misma persona: modelado 3D, texturizado, iluminación, animación, etc.

Esto, de nuevo, difiere mucho de las habilidades necesarias para desarrollar un producto software más tradicional, como una página web, en el que el papel artístico queda en un relativo segundo plano: sin duda forma parte del conjunto del software, pero es mucho más accesible para un ingeniero de software que realizar, por ejemplo, un modelado 3D de un personaje.

Realizar el proyecto individualmente, sin duda, tiene algunas ventajas como que no recorta un ápice de creatividad, permitiéndonos tomar las decisiones de diseño que personalmente consideremos mejor para el producto, pero por otra parte hace mucho más difícil abarcar tantos aspectos y tan diferentes. Por ello, en una empresa de desarrollo de videojuegos coexisten no sólo programadores, sino diseñadores y artistas, jugando estos últimos un papel mucho más relevante que en otro tipo de aplicaciones.

1 15.2 DISCUSIÓN

2 En función de lo anterior, si hoy empezáramos un nuevo proyecto...

- 3 ■ Se seguiría apostando por C++ frente a blueprints: pese a que ambas opciones
4 libran una ardua batalla, como se ha dejado patente en este capítulo, la ventaja
5 de tener control sobre el código (en cuanto al uso de repositorios) y la optimiza-
6 ción pesan más que los tiempos de compilación, para lo cuál se barajaría la
7 posibilidad de adquirir nuevos equipos para reducir notablemente el tiempo de
8 compilación (y, colateralmente, el tiempo de construcción de proyecto o el cálculo
9 de la iluminación, cosa que se ha comprobado ser también muy costosa en cuanto
10 a tiempo).
- 11 ■ Se apostaría por la ampliación del equipo de trabajo, pero no sólo con la incursión
12 de nuevos programadores que permitiesen llegar a objetivos más ambiciosos o
13 reducir los tiempos de desarrollo, sino definitivamente de artistas y diseñadores
14 que se encargasen de cubrir las carencias del resto del grupo con sus aptitudes de
15 diseño o artísticas, ayudando además a diferenciar mucho más el producto. En
16 definitiva, se trataría de formar un equipo polifacético.
- 17 ■ Se barajaría también la posibilidad de contar con «beta-testers» más especializa-
18 dos que supieran dar indicaciones más precisas para poder reproducir los bugs
19 que se encontrasen en el producto. Igualmente, en este tipo de productos existen
20 bugs con una muy baja probabilidad de repetición, incluso en los que influya un
21 tanto la aleatoriedad, y se necesitaría gente dedicada al testeo para paliar esta
22 carencia de la naturaleza de este tipo de software.
- 23 ■ Se haría aún más hincapié en la necesidad de hacer construcciones periódicas del
24 proyecto, por los motivos previamente explicados: los errores de construcción
25 son difíciles de determinar y una construcción del proyecto periódica definiti-
26 vamente ayuda a acotar el error y, sobre todo, porque funcionalidad que podría
27 funcionar en el editor no funciona en el proyecto construido o no lo hace de la
28 misma forma (dando incluso lugar a cierres del programa).
- 29 ■ Salvo en casos de extrema necesidad no se actualizaría Unreal Engine a la versión
30 más reciente de forma precipitada, sino que se estudiaría la migración del pro-
31 yecto a dichas versiones previamente y se buscaría información en la comunidad
32 de Unreal de los posibles fallos de las nuevas versiones.

15.3 TRABAJOS FUTUROS

Esta sección pretende enumera los puntos abiertos, aquellos que por un motivo u otro o bien no se han resuelto en el trascurso del proyecto o bien son candidatos a una revisión o mejora en el futuro. Si bien todo lo que se había planeado al inicio del proyecto ha podido realizarse de una manera satisfactoria a la conclusión del mismo hay una serie de matices o puntos de mejora que podemos comentar.

Si bien el **sistema de escalada** es algo en lo que se ha hecho especial énfasis en la realización del proyecto y es una pieza que se podría catalogar como clave, a la cual se la ha tratado con mimo y cuya funcionalidad planeada ha sido totalmente implementada existen algunas nuevas funcionalidades que podríamos añadir a la misma, además de una manera sencilla gracias al conocimiento que se ha adquirido con la realización del proyecto y gracias a que base de la que goza el sistema de escalada está implementada y depurada. Una de estas funcionalidades podrían ser los saltos verticales hacia arriba en los que nuestro personaje, estando ya agarrado a un saliente, pudiera saltar para alcanzar uno que se encontrase en un nivel superior, lo que haría ganar al diseño de niveles de una preciada verticalidad. Esta nueva funcionalidad no es más que un ejemplo puesto que, como siempre, el límite es la imaginación, lo que podría original un proyecto futuro centrado en parte en ampliar el sistema de escalada.

De igual manera, otro punto que podría sufrir una mejora notable es el **repertorio de habilidades** de las que nuestro personaje dispone cuando está potenciado. Actualmente dispone de dos habilidades: pasar a tiempo bala (habilidad que ralentiza el tiempo del entorno en un 50% y el de nuestro personaje en un 25%, haciendo más fácil solucionar algunos puzzles) y realizar un teletransporte en la dirección indicada por la cámara (también denominado «blink» o «parpadeo»), pero se podrían añadir muchas más. Una de ellas, descartada antes del desarrollo del proyecto pero que se llegó a implementar en un prototipo realizado con blueprints, podría ser el poder de resistir una serie de impactos de bala de las torretas, a cambio de cambiar la velocidad de movimiento del personaje a una considerablemente más lenta. Esto podría dar también lugar a un nuevo proyecto.

Otro punto candidato de mejora sería la incursión de nuevos tipos de **enemigos** y sus respectivas inteligencias artificiales. Estos enemigos podrían combinarse con los ya presentes en el producto dando lugar a innumerables nuevos tipos de puzzles.

1 Por supuesto, el punto de mejora más claro es el de los **niveles**: actualmente el
2 juego consta de un tutorial y dos niveles y si bien el juego está destinado a jugarse
3 haciendo énfasis en la repetición, lo cuál alarga de manera contundente la duración
4 de los niveles, este punto es un claro candidato a ampliarse en futuros proyectos que
5 tengan como objetivo trabajar a partir de la base de éste. Además, de igual manera
6 se podría realizar una revisión de los primeros niveles para añadir más motivos de
7 decoración.

8 Además, si bien el **sistema de logros** está implementado en el proyecto tal y como
9 se quería, debemos recordar que por un lado no todos los logros diseñados están im-
10 plementados en el producto (aunque esto es un mero trámite, puesto que el sistema
11 como tal ya existe) y que, sobre todo, el sistema de logros interno está pensado para
12 interaccionar con el sistema de logros de diversas plataformas en el hipotético caso de
13 que el producto se lanzara en ellas. Esta parte de la funcionalidad no está implementa-
14 da puesto que no se dispone de acceso a ninguna de las API de las citadas plataformas.

15 Otros puntos de mejora serían la inclusión de **configuraciones gráficas personali-
16 zadas** en el menú así como un **soporte total para el controlador**, ya que de momento es
17 necesario usar ratón en el menú principal, y el soporte de más **idiomas** en el producto.

18 La **adaptación a realidad virtual** no es que se pueda considerar como un punto
19 abierto en este proyecto, puesto que desde el principio se planteó como finalmente se
20 ha incluido, pero sí que podría dar lugar a un pequeño proyecto de adaptación parcial
21 a esta tecnología en el futuro. El motivo por el que no se ha realizado esta pequeña
22 adaptación es que, simplemente, no se dispone del hardware necesario para su testeo,
23 pero entre implementación y pruebas no superaría unas horas de trabajo gracias a las
24 facilidades que nos ofrece Unreal Engine 4 y a que sólo se quiere adaptar el control de
25 la cámara con los cascos / gafas de realidad virtual, siendo necesario igualmente un
26 controlador o bien teclado y ratón para jugar.

27 Por último, cabe hacer mención a una característica que finalmente ha podido ser
28 incluida en el juego, pero que en un momento se pensó que podría ser descartada por
29 limitaciones tecnológicas y que daría lugar a un nuevo proyecto: la implementación
30 del cono de visión de los enemigos denominados como «vigilantes». Estos enemigos,
31 como se detalla en este documento, entre otras particularidades poseen un cono de
32 visión que debe ser renderizado por el juego para que el jugador pueda conocer el
33 área de visión de dicho enemigo y así ocultarse en el momento oportuno, pero esto
34 es un problema difícil de solucionar en más de un escenario, entre los que se incluye
35 este juego. Unreal Engine 4.15 no soportaba la posibilidad de incluir luz volumétrica.

ca y, típicamente, para la implementación de esta característica se usaba un cono, con colisiones desactivadas y alta transparencia en cada uno de los objetos en los que quisiéramos simular luz, el ejemplo más típico es el de una farola que, en la oscuridad, emite luz desde su foco hasta el suelo (que sería la extensión del cono en este caso, la base del cono se situaría en el suelo y el vértice en el foco de la farola). En nuestro caso no era posible llevar a cabo esta solución puesto que, si bien podríamos adjuntar un cono con transparencias y sin colisiones a la cabeza de nuestro modelo de enemigo no tendríamos ninguna forma de que dicho cono no traspasase las paredes, ni mucho menos de que su longitud se adaptase a los objetos con los que colisiona en el escenario (pudiendo colisionar partes del cono con un obstáculo y partes continuar su recorrido). Sin embargo, con la aparición de Unreal Engine 4.16 se incluyeron las luces volumétricas y este problema pudo ser solucionado adjuntando un foco de luz a la cabeza del modelo del vigilante y configurándolo para que la luz que emitiese se comportase como volumétrica.

1
2
3
4
5
6
7
8
9
10
11
12
13
14



Figura 15.2: Fotograma del título «Fallout 4» (2015) que muestra una farola que emite un cono de luz

Como podemos apreciar, ninguno de estos detalles son aspectos que hayan quedado pendientes en el desarrollo sino mejoras que se podrían aplicar.

15
16

PARTE V

APPENDICES

GLOSARIO

Asset	Representación de cualquier elemento que pueda ser utilizado por el motor gráfico (música, efecto de sonido, modelado 3D, textura, etc.). Su origen puede ser externo al motor o bien crearse dentro del mismo.
Blueprint	En Unreal Engine, los blueprints son cada una de las plantillas sobre las que se implementa programación gráfica. Los blueprints permiten emplazar nodos y unirlos entre sí, creando así un flujo entre los mismos similar al que obtendríamos utilizando métodos de codificación más tradicionales, como "C++", donde cada nodo se traduciría como una línea de código.
Bullet hell	Traducido al español como "infierno de balas", bullet hell es un subgénero de videojuegos que se caracteriza por llenar el entorno ("la pantalla") de peligros (tradicionalmente balas, de ahí su nombre) que matan, dañan o provocan algún tipo de alteración al personaje.
Distribuidora	En el mundo de los videojuegos es una empresa que distribuye juegos, ya sean propios o de terceros, y que normalmente se encarga también de aspectos propios de una editorial, como de llevar a cabo la promoción de la obra.

Escenario	Escena o escenario es la manera en la que se suele denominar un mapa o escenario de un videojuego (en inglés, se suele denominar ““scenario””).
Frame	Los frames, o fotogramas en nuestro idioma, son típicamente cada una de las imágenes en las que se divide un vídeo. Cuando hablamos de frames en animaciones nos referimos a un instante de la animación (la posición, rotación y escala de cada uno de los huesos que forman el esqueleto).
Gameplay	Forma en la que se juega a un videojuego, incluyendo las reglas, los objetivos y la forma de alcanzarlos.
Gaming	Hace referencia a que el dispositivo está diseñado o enfocado para el uso relacionado con videojuegos.
Independiente	Generalmente se denomina videojuego independiente o indie a aquel realizado por un número reducido de personas y/o con poco presupuesto, sin el respaldo de una gran compañía.
Jugabilidad	Experiencia del jugador durante la interacción con las distintas mecánicas, controles y situaciones del videojuego. Se podría decir que una buena jugabilidad es aquella que transmite correctamente las acciones que el jugador le da al software (a través de los dispositivos de entrada) a su ““avatar”” (en este caso, personaje) dentro del videojuego.

Landscape	En un videojuego, un landscape es un plano, generalmente con diferentes alturas, que se utiliza principalmente para la recreación de un paisaje natural pudiendo contener montañas, valles, etc. Principalmente se usa como base para un escenario y en él se emplazarán árboles, edificios y modelados de todo tipo.
Mecánica	Son cada una de las reglas del universo del videojuego y conjunto de acciones que puede realizar el jugador. Por ejemplo: Una mecánica sería que el personaje muriese al caer desde una determinada altura o bien que el personaje, simplemente, pueda correr o saltar.
Metodología SCRUM	Metodología ágil orientada a equipos autogestionados que, muy resumidamente, se caracteriza por sus reuniones diarias de corta duración (15 minutos como máximo) para hablar de lo que se ha hecho, lo que se va a hacer y los problemas que se han encontrado y basada en sprints: la realización de un conjunto de funcionalidades (Sprint Backlog), extraídas del Product Backlog (el conjunto de funcionalidades totales).
Metodologías ágiles	Las metodologías ágiles surgieron como contraposición a las metodologías tradicionales. Son aquellas que se basan en el desarrollo iterativo y donde los requisitos y soluciones evolucionan a lo largo del tiempo según lo necesite el proyecto.
Modelado	Un modelado o malla (del inglés mesh) es la representación mediante polígonos de un determinado objeto, que normalmente tiene asociado un material o textura, pudiendo también tener asociado varios.

Motor gráfico	Es un framework diseñado para desarrollar videojuegos que le ofrece al programador al menos las características básicas para llevar a cabo esa tarea, tales como podrían ser un motor de renderizado para visualizar gráficos en pantalla, detección de colisiones entre objetos, creación de animaciones, etc.
Plataformas	Género de videojuegos en el que el personaje tendrá que hacer uso de mecánicas básicas como salto, sprint o escalada para ir sorteando obstáculos presentes en el mapa y superar el nivel.
Renderizar	Proceso de generar un espacio que normalmente consta de tres dimensiones (3D), pero también se utiliza para generar espacios de dos (2D), representando en el proceso objetos, materiales, luces, etc.
Retargeting	Técnica usada cuando se trabaja con animaciones para adaptar las animaciones creadas sobre un esqueleto a otro. Consiste en encontrar similitudes entre los huesos de uno y otro esqueleto. Además, para que la técnica funcione ambos esqueletos deben estar en la misma pose de inicio (si no lo están, se deberá modificar la pose del esqueleto destino).

Tick

Se denomina tick a cada ciclo generado en el bucle del proceso del juego. Este concepto va enormemente ligado al concepto de frame, ya que cada tick genera a su vez un fotograma del juego. En resumidas cuentas se podría decir que en cada tick del juego se elaboran los cálculos pertinentes para generar un nuevo fotograma (los ejemplos más básicos podrían ser la posición y rotación de los actores o su estado, en ese mismo momento).

Timing

Realizar una acción específica en el momento preciso. Normalmente está sujeto a la repetición de niveles, es decir, si el jugador realiza las mismas acciones en el mismo momento pasará el nivel o una parte del mismo (siempre que no haya elementos aleatorios involucrados).

Triple A (AAA)

Se denominan así a los videojuegos lanzados por grandes empresas, con un gran presupuesto detrás tanto para su producción como para su promoción. El motivo de esto es que las tres aes (AAA), tradicionalmente, significaban un diez en gráficos, un diez en sonido y un diez en jugabilidad, pero el término ha ido evolucionando hasta llegar al descrito anteriormente, sin tener que ser necesariamente un juego de diez en ningún apartado.

Volumétrico

En concepto de volumétrico en el ámbito de los gráficos generados por ordenador hace referencia a la técnica de añadir efectos de luz o niebla en un escenario de forma que interaccione con el mismo. Por ejemplo, al emplazar una luz solar volumétrica en un escenario los rayos de luz traspasarían entre las hojas de los posibles árboles e ignorarían materiales que no sean opacos como cristales.

DOCUMENTO DE DISEÑO

Un documento de diseño, conocido en la industria como GDD por sus siglas en inglés (*Game Design Document*), es un documento que recoge las características que conforman el videojuego, es decir, una síntesis del mismo.

Teniendo en cuenta las características de nuestro proyecto, no se pretende elaborar un documento de diseño al uso sino un documento cuyo principal fin sea permitir generar, a partir del mismo, los requisitos del producto software que se va a desarrollar.

Además, atendiendo de nuevo a las características específicas de nuestro proyecto, numerosas secciones del documento de diseño se verían solapadas con la documentación del proyecto ya descrita en el cuerpo de este documento. En caso de que se produzca dicho solapamiento, se citará la sección correspondiente.

B.1 VISIÓN DE CONJUNTO

La visión de conjunto describe de manera global videojuego, su género, la forma de jugarlo y los elementos que lo componen.

Este apartado se describe en la sección «Motivación», donde mediante frases cortas se describen los elementos de los que consta el videojuego cuyo desarrollo está ligado a este proyecto (Ver sección §2.1).

B.1.1 Plataformas de destino

Las plataformas de destino son aquellas para las cuales se está desarrollando el software o se tiene previsto su desarrollo.

A pesar de que nuestro motor gráfico permite la exportación de nuestro producto a diversas plataformas tales como PlayStation 4, Xbox One, Nintendo Switch, Linux o Mac (entre otras), la única plataforma de destino en la que nos centraremos durante la realización del proyecto será Microsoft Windows.

B.1.2 Requisitos mínimos objetivo

Los requisitos mínimos hacen alusión a las características que, como mínimo, deberá tener un equipo para hacer funcionar el videojuego.

El proyecto que nos ocupa no someterá a ninguno de los componentes a una carga alta de trabajo: el poligonaje de los modelos no será alto, las texturas usadas no serán de gran resolución, por norma general no se hará un uso avanzado de la iluminación (por ejemplo, uso excesivo de sombras dinámicas), etc. Asimismo, el motor gráfico usado se caracteriza por no hacer un uso agresivo de los recursos disponibles.

Los requisitos mínimos, o en este caso los requisitos mínimos objetivo, coinciden con los dispositivos descritos en la sección «Diseño arquitectónico» (Ver sección §6.2). Los requisitos mínimos como tal no podrán ser calculados hasta la finalización del proyecto.

B.1.3 Estilo gráfico

El estilo gráfico del producto se podría enmarcar como realista: tanto el personaje como el escenario que se usará como tutorial se clasificarían en este estilo. Sin embargo, debemos tener presente que los niveles prototipo que se pretenden utilizar para recrear la forma en la que se jugaría el juego se desmarcarían de este estilo, ya que usarían formas y texturas mucho más básicas al tratarse de niveles prototipo.

Dentro de este estilo realista, se podría clasificar el universo donde tiene lugar el juego como fantástico.

B.1.4 Estilo sonoro

La música del juego... TODO.

Los efectos de sonido (utilizados cuando el personaje anda, salta, cae, etc.) se podrían clasificar como realistas.

Respecto a las voces y diálogos el producto carecerá de los mismos.

B.2 COMIENZO

B.2.1 Menú principal

B.2.2 Comienzo del juego e introducción

B.2.3 In-game HUDs y menús

B.3 INTERFAZ DE USUARIO

B.3.1 Menús

Coincide con la sección «Manual» del cuerpo del documento.

B.3.2 In-game HUD

El HUD se compone de los siguientes elementos:

B.3.3 In-game Options

TODO.

B.3.4 Pantalla de fin de juego

Cuando el personaje entre en un estado de fin de juego, la pantalla pasará a negro y se mostrará el mensaje de final de juego sobre la misma hasta que el jugador presione la tecla o botón indicado para reiniciar el nivel desde el último punto de control.

B.3.5 Pantalla de selección de nivel

TODO.

Tipos de nivel

El producto consta de dos tipos de nivel, muy diferenciados:

- Nivel menú: Utilizado para mostrar el menú del juego.
- Nivel tipo tutorial: Enseñará al jugador a usar las distintas mecánicas del juego.
- Nivel tipo regular: Se recrea un escenario real de juego.

Lista completa de niveles

La etapa de desarrollo que tendrá lugar durante el desarrollo de este proyecto generará 3 niveles en total:

- Nivel tutorial.
- Nivel prototipo 1.
- Nivel prototipo 2.

El nivel del tutorial tendrá un nivel de acabado representativo del acabado final del producto (por tanto estará correctamente iluminado, texturizado, decorado, etc.), mientras que el resto de niveles serían niveles de ejemplo y tendrán un acabado más simple.

B.4 GAMEPLAY

B.4.1 Mecánicas

Las mecánicas son el conjunto de acciones que puede realizar el jugador y las reglas por las que se rige el universo del videojuego. Atendiendo a este proyecto, podemos establecer claros grupos entre las mismas:

- **Mecánicas de bajo perfil:** En este proyecto, nos referimos a mecánicas de bajo perfil cuando hablamos de mecánicas dependientes del jugador y que podrá usar en cualquier momento.
- **Mecánicas de alto perfil:** Se denominan así al conjunto de mecánicas que el jugador podrá utilizar cuando disponga de poderes específicos.
- **Mecánicas de escalada:** El conjunto de mecánicas relacionadas con la escalada. Son mecánicas dependientes del jugador y que se pueden llevar a cabo en cualquier momento, por lo que podrían englobarse en las mecánicas de bajo perfil pero se categorizan por separado debido a la importancia que tienen dentro del proyecto y al número de las mismas.
- **Mecánicas del entorno o universo:** Son mecánicas que tienen relación con cómo se relaciona el personaje con el entorno.

Mecánicas de bajo perfil

1. **Desplazamiento:** El jugador podrá desplazar el personaje por el entorno pulsando las teclas, botones o joysticks asignados para ello.
2. **Rotación:** El jugador podrá rotar al personaje y, a su vez, a la cámara utilizando el movimiento del ratón o el joystick asignado.

3. **Interaccionar:** El personaje podrá interaccionar con determinados elementos prefijados del entorno pulsando la tecla o el botón asignado para ello. Esta acción provocará por tanto cambios en el jugador o en el entorno.

4. Saltar:

- El personaje podrá saltar, ganando altura hasta una altura máxima, mientras el jugador mantiene pulsada la tecla de salto.
- Si la altura máxima de salto se ha alcanzado o el jugador cesa en su acción de presionar la tecla de salto el personaje dejará de ganar altura en el salto y comenzará a caer.

5. Agachado:

- Mientras que el jugador mantenga pulsada la tecla vinculada a la acción de agacharse el personaje se agachará, cambiando la posición de la cámara y reduciendo su caja de colisión, permitiéndole acceder a zonas por las que no podría pasar erguido o bien esquivar peligros.
- Cuando el jugador deje de presionar dicha tecla el personaje volverá a estar de pie.

6. Correr:

- Mientras que el jugador mantenga pulsada la tecla de sprint el personaje pasará a desplazarse por el entorno con una velocidad mayor, acelerando además su animación.
- Cuando el jugador deje de presionar la tecla relacionada el personaje volverá a su velocidad habitual.

Mecánicas de alto perfil

1. **Tiempo bala (o time dilation):** El jugador podrá, presionando la tecla o el botón correspondiente, ralentizar el tiempo del entorno y del personaje (manteniendo el escenario levemente más ralentizado que el personaje, para de esta manera ganar una pequeña ventaja) para así poder sortear obstáculos que de otra forma le serían complicado o imposible de salvar.
2. **Parpadeo (o blink):** El jugador podrá teletransportar al personaje a la posición a la que está mirando si no hay obstáculos de por medio.

Mecánicas de escalada

1. **Wall running:** El personaje podrá correr verticalmente por una pared, hasta una altura máxima, siempre que sea una superficie adecuada para ello.
2. **Salto en wall running:** El personaje podrá utilizar la pared que está escalando para impulsarse hacia el lado contrario, cambiando de esta forma su orientación. Si el personaje combina esta mecánica con la anterior (es decir, si tenemos dos superficies en la que hacer wall running una en frente de otra) para encadenar varios wall runnings y permitirle subir a posiciones muy elevadas.
3. **Agarrarse a un saliente:** El personaje podrá permanecer agarrado a un saliente siempre que se reúnan las condiciones adecuadas.
4. **Escalar un saliente:**
 - Si está agarrado y no hay obstáculos que no le permitan subir, pulsando el botón de saltar podrá terminar de escalar el saliente.
 - Si no está agarrado pero el saliente es de baja altura automáticamente escalará el saliente en vez de agarrarse a él.
5. **Desplazarse por un saliente:** Siempre que no haya obstáculos el personaje podrá desplazarse lateralmente por un saliente.
6. **Dejarse caer de un saliente:** Presionando el botón de agacharse el jugador puede hacer que el personaje abandone el saliente al que estaba agarrado.
7. **Salto lateral desde un saliente:** Cuando el personaje esté al final de un saliente, si no hay obstáculos de por medio, podrá saltar lateralmente desde ese saliente, permitiéndole así agarrarse a salientes que no estaban conectados con el original o llegar a zonas previamente inaccesibles.
8. **Salto reverso desde un saliente:** Cuando el personaje esté agarrado a un saliente podrá impulsarse desde el mismo y realizar un salto en la dirección opuesta a la del saliente.

Mecánicas del entorno

1. Si el personaje entra en contacto con algún peligro del escenario (como balas, láseres, etc.) morirá instantáneamente.
2. El personaje, al caer...

- ... Si la altura de caída es lo suficientemente pequeña, no pasará nada.
 - ... Si la altura de caída es lo suficientemente grande, no podrá moverse por unas décimas de segundo.
 - ... Si cae al vacío morirá instantáneamente.
3. El personaje morirá instantáneamente al abandonar el área de juego.

B.4.2 Controles

Esta sección coincide con la homónima del cuerpo del documento (Ver sección §14.1).

B.4.3 Modos de juego

Aunque el jugador sólo perciba uno, en realidad se establecerán dos modos de juego: el primero se utilizará para navegar por los menús y el segundo será la forma de jugar al juego en sí.

B.4.4 Ganando el juego

La única forma de ganar el juego es completando la fase, es decir, cuando el personaje entre en contacto con la caja de colisión deseada al final de cada nivel.

B.4.5 Logros / Trofeos

Los logros son medallas otorgadas al jugador al realizar un avance específico dentro del juego y que dan una idea del progreso dentro del mismo.

Relacionados con la historia

El jugador los obtendrá al hacer progresos en lo que se podría denominar «la historia», en este caso al finalizar un nivel.

[...]

Logro historia - Juego



Logo

Obtención Otorgado al jugador por completar el juego

Idioma	Nombre	Descripción
Español	Completado	Completa el juego
Inglés	Finished	Finish the game

Cuadro B.1: Logro historia - Juego

Logro historia - Tutorial



Logo

Obtención Otorgado al jugador por completar el tutorial

Idioma	Nombre	Descripción
Español	Puesta a punto	Completa el tutorial
Inglés	Setup	Finish the tutorial

Cuadro B.2: Logro historia - Tutorial

Logro historia - Capítulo I



Logo

Obtención Otorgado al jugador por completar el primer nivel

Idioma	Nombre	Descripción
Español	Capítulo I	Completa el capítulo I
Inglés	Chapter I	Finish the chapter I

Cuadro B.3: Logro historia - Capítulo I

Logro historia - Capítulo II



Logo

Obtención Otorgado al jugador por completar el segundo nivel

Idioma	Nombre	Descripción
Español	Capítulo II	Completa el capítulo II
Inglés	Chapter II	Finish the chapter II

Cuadro B.4: Logro historia - Capítulo II

Pruebas de tiempo

Son aquellos otorgados al jugador al superar un escenario por debajo de un determinado tiempo preestablecido.

Logro prueba de tiempo - Capítulo I		
Logo		
		
Obtención	Otorgado al jugador por completar el primer nivel por debajo del tiempo establecido	
Idioma	Nombre	Descripción
Español	Prueba de tiempo: Capítulo I	Completa el capítulo I por debajo del tiempo establecido
Inglés	Time trial: Chapter I	Finish the chapter I under the time limit

Cuadro B.5: Logro prueba de tiempo - Capítulo I

Logro prueba de tiempo - Capítulo II		
Logo		
		
Obtención	Otorgado al jugador por completar el segundo nivel por debajo del tiempo establecido	
Idioma	Nombre	Descripción
Español	Prueba de tiempo: Capítulo II	Completa el capítulo II por debajo del tiempo establecido
Inglés	Time trial: Chapter II	Finish the chapter II under the time limit

Cuadro B.6: Logro prueba de tiempo - Capítulo II

[...]

Coleccionables

El jugador obtendrá logros al recoger objetos repartidos por el universo del juego que el personaje podrá recolectar al entrar en contacto con ellos. Concretamente el jugador obtendrá un logro en los siguientes casos:

Logro colecciónables - 1 colecciónable		
Logo		
Obtención	Otorgado al jugador por encontrar un colecciónable (cualquiera de ellos)	
Idioma	Nombre	Descripción
Español	Colecciónables 1	Encuentra tu primer colecciónable
Inglés	Collectibles 1	Find your first collectible

Cuadro B.7: Logro colecciónables - 1 colecciónable

Logro colecciónables - 5 colecciónables		
Logo		
Obtención	Otorgado al jugador por encontrar cinco colecciónables cualquiera	
Idioma	Nombre	Descripción
Español	Colecciónables 5	Encuentra cinco colecciónables en total
Inglés	Collectibles 5	Find five collectibles in total

Cuadro B.8: Logro colecciónables - 5 colecciónables

Logro colecciónables - 10 colecciónables		
Logo		
Obtención	Otorgado al jugador por encontrar diez colecciónables cualquiera	
Idioma	Nombre	Descripción
Español	Colecciónables 5	Encuentra diez colecciónables en total
Inglés	Collectibles 5	Find ten collectibles in total

Cuadro B.9: Logro colecciónables - 10 colecciónables

Logro colecciónables - 50 % de colecciónables		
Logo		
Obtención	Otorgado al jugador por encontrar el 50 % de los colecciónables	
Idioma	Nombre	Descripción
Español	Colecciónables 50 %	Encuentra la mitad de los colecciónables
Inglés	Collectibles 50 %	Find a half of the collectibles

Cuadro B.10: Logro colecciónables - 50 % de colecciónables

Logro colecciónables - 100 % de colecciónables



Logo

Obtención Otorgado al jugador por encontrar todos los colecciónables

Idioma	Nombre	Descripción
Español	Colecciónables 100%	Encuentra todos los colecciónables
Inglés	Collectibles 100 %	Find all the collectibles

Cuadro B.11: Logro colecciónables - 100 % de colecciónables

Misceláneos

Además de los indicados, el jugador será recompensado con un logro al realizar las siguientes acciones satisfactoriamente:

Logro misceláneos - 100 saltos		
Logo		
Obtención	Otorgado al jugador por saltar 100 veces	
Idioma	Nombre	Descripción
Español	Salto 100	Salta 100 veces
Inglés	Jump 100	Jump 100 times

Cuadro B.12: Logro misceláneos - 100 saltos

Logro misceláneos - 1000 saltos		
Logo		
Obtención	Otorgado al jugador por saltar 1000 veces	
Idioma	Nombre	Descripción
Español	Salto 1000	Salta 1000 veces
Inglés	Jump 1000	Jump 1000 times

Cuadro B.13: Logro misceláneos - 1000 saltos

Logro misceláneos - 10000 saltos



Logo

Obtención Otorgado al jugador por saltar 10000 veces

Idioma	Nombre	Descripción
Español	Salto 10000	Salta 10000 veces
Inglés	Jump 10000	Jump 10000 times

Cuadro B.14: Logro misceláneos - 10000 saltos

100 %

Verifica que el jugador ha completado el resto de logros.

Logro 100 %		
 A blue and white graphic logo featuring the number "100%" in a large, bold, sans-serif font. The logo has a three-dimensional effect with a white drop shadow and a slight perspective.		
Logo		
Obtención	Otorgado al jugador por obtener el resto de logros	
Idioma	Nombre	Descripción
Español	100 %	Completa el juego y acaba con el resto de retos
Inglés	100 %	Finish the game and all the challenges

Cuadro B.15: Logro 100 %

B.5 ASSETS

Para la elaboración del producto se usan gran cantidad de assets y de variada clasificación, la inmensa mayoría procedente de proyectos de libre uso de Unreal Engine creados por Epic Games (como por ejemplo assets usados en demos técnicas o los pertenecientes a packs de assets liberados).

Teniendo en cuenta esto, esta sección no pretende elaborar una lista detallada de todos los assets que se han usado en la creación del videojuego, sino los assets que se salgan de esa norma: assets de uso gratuito no pertenecientes a Epic Games o assets sin licencia de uso gratuito (de los cuales se disponga de sus derechos de uso, ya sea mediante su adquisición o por el permiso explícito de su creador, o bien assets realizados por el equipo del proyecto).

En cada uno de los casos se explicará el uso específico que se ha hecho de cada asset, es decir, cómo ha repercutido al proyecto.

B.5.1 Personajes

Tratándose de un juego de un solo jugador (y sin selector de personajes ni ningún otro método de intercambio de personajes jugables) sólo se usa un asset, que es el siguiente:

Assets de personajes			
ID	Personaje	Procedencia	Beneficios aportados
#1	Protagonista	Adquisición	Estrictamente estéticos
Notas: Cambio meramente estético respecto al maniquí por defecto de Unreal Engine. Se compone de un modelado y textura (ya que usa el mismo esqueleto que el maniquí por defecto).			

Cuadro B.16: Assets de personajes

B.5.2 Enemigos

TODO

B.5.3 Animaciones

Las animaciones usadas en el proyecto tienen cuatro orígenes principalmente: animaciones extraídas del pack de animaciones gratuito de Epic Games (disponible para su descarga desde el Marketplace de Unreal Engine), animaciones de la plantilla por defecto de tercera persona de Unreal Engine 4 (al crear esta plantilla, se generan estas animaciones), animaciones propiedad de Mixamo (cuyo uso es gratuito desde hace unos años) y animaciones realizadas específicamente para el proyecto generalmente a partir de un frame de una animación.

Una buena parte de las animaciones usadas, sea de la fuente que sea, ha tenido que ser modificada de alguna u otra forma para su uso en el proyecto pero, más específicamente, las animaciones de Mixamo tienen añadido un trabajo extra: dichas animaciones no están creadas a partir de un esqueleto compatible con Unreal Engine 4, por tanto ha sido necesario realizar un **retargeting** en cada una de ellas para adaptarlo al esqueleto que del personaje del que se hace uso (que sí usa un esqueleto compatible con el esqueleto por defecto de Unreal Engine 4).

B.5.4 Equipo y mejoras

TODO: Poderes

B.5.5 Entorno

TODO

B.5.6 Audio

TODO

BIBLIOGRAFÍA

- [1] A comparison of fdd and scrum, 2011. (pages 27, 29).
- [2] Metodología fdd - feature driven development / desarrollo basado en funciones, 2012. (pages 27, 28).
- [3] AEVI. El videojuego en el mundo. Technical report, Asociación española del videojuego (AEVI), 2017. (page 4).
- [4] Buhomag. Xbox apuesta fuerte por los videojuegos indies españoles, 2015. (page 10).
- [5] CC.OO. Xvi convenio colectivo estatal de empresas consultoras de planificación, organización de empresas y contable, empresas de servicios de informática y de estudios de mercado y de la opinión pública, 2008. (page 46).
- [6] U. E. Developer. Fortnite developer fireside on unreal engine 4, 2014. (pages 16, 21, 147).
- [7] DZone. An introduction to feature-driven development, 2009. (page 27).
- [8] E. Economista. Los videojuegos facturaron más de 1.000 millones, el doble que la industria del cine, 2016. (pages XI, 6).
- [9] EpicGames. *Documentación Unreal Engine*. Epic Games, 2017. (page 61).
- [10] Fortune. The 10 most successful states for video game development, 2015. (page 44).
- [11] Gamasutra. Gamasutra salary survey 2014. Technical report, Gamasutra, 2014. (pages XV, XV, 43, 44, 46).
- [12] T. P. O. V. Games. Why do achievements, trophies, and badges work?, 2016. (page 12).
- [13] Investing. Datos históricos eur/usd, 2017. (pages XII, 45).

- [14] MejoresProyectos. El síndrome de los 90, 2007. (page 146).
- [15] E. Mundo. Playstation premia el talento de los videojuegos españoles, 2016. (page 10).
- [16] Newzoo. 2017 global mobile market report. Technical report, Newzoo, 2017. (pages xi, 8).
- [17] Newzoo. Esports revenues will reach 696 million dollars this year and grow to 1.5 billion dollars by 2020. Technical report, Newzoo, 2017. (pages xi, 7).
- [18] Newzoo. Global games market 2016 report. Technical report, Newzoo, 2017. (pages xi, 4, 8).
- [19] Numbeo. Cost of living index 2017 mid-year, 2017. (pages xv, 44).
- [20] PC-Gamer. A game with over 4,000 achievements has just launched on steam, 2017. (page 12).
- [21] E. Press. La increíble historia de et, el videojuego que hundió a atari antes de ser enterrado, 2017. (pages xi, 14).
- [22] SoftZone. Qué es steam direct y en qué se diferencia de steam greenlight, 2017. (page 11).
- [23] Steam. Tasa de steam direct y próximas actualizaciones de la tienda, 2017. (page 11).
- [24] Step-10. Fdd: History, 2014. (page 27).
- [25] Step-10. Fdd: People, 2014. (page 28).
- [26] U-Tad. La industria del videojuego en españa continúa con su imparable crecimiento, 2015. (pages 6, 15).
- [27] Xataka. Las chicas también juegan a videojuegos, y cada vez más, 2014. (page 9).
- [28] ZehnGames. El desarrollo español en steam: Una lectura desde los datos. Technical report, Zehn Games, 2017. (pages xi, 10).
- [29] ZonaRed. Playerunknown's battlegrounds continúa en el trono de las ventas semanales de steam, 2017. (page 70).