

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

septiembre 2017

Publicado en septiembre 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
03/09/2017

Dedicado a todos los que me han apoyado

Y en especial a...

A mi familia, pareja y amigos.



AGRADECIMIENTOS

Querría agradecer y dedicar este humilde trabajo especialmente a mis padres, José y Pepi, y a la persona con la que tengo la suerte de compartir mi vida desde hace ya bastantes años, Cristina, puesto que no recuerdo ni una sola vez en la que ellos no me hayan apoyado en algo que quisiera hacer, y este proyecto no ha sido una excepción.

Agradecer por igual a toda mi familia y amigos, pero me gustaría dedicarlo con especial acento a los que desafortunadamente ya no están: mis abuelos, familiares, amigos y a alguien que fue, y seguirá siendo, una hermanita para mí.

Por supuesto querría accordarme de mis compañeros, con los que tantas horas he pasado estudiando o trabajando, puesto que dicen que la unión hace la fuerza, para sacar la carrera a la que pone fin este trabajo.

No me gustaría olvidarme del profesorado, de todos los campos y niveles académicos, que a fin de cuentas son los que me han formado para que pueda realizar este proyecto. Querría hacer especialmente énfasis en aquellos que me han enseñado programación y matemáticas, puesto que prácticamente todo el proyecto se basa en esos campos docentes. Más especialmente me querría agradecer a mi tutor, Pablo, por aguantar mis numerosas dudas, por todo el apoyo que me ha dado y lo bien que me ha tratado en todo momento.

Y, por último pero no menos importante, me gustaría agradecer a todos aquellos que han sacado un rato de su tiempo para probar el prototipo del proyecto. Por el mero hecho de probarlo ya les estoy agradecido, pero muchos de ellos lo han seguido jugando, me han informado de errores y han querido completarlo.

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.

IMPORTANTE: Los enlaces que se exponen a continuación contienen un vídeo, en el que se muestra el resultado final del proyecto (una demostración de más de 20 minutos, aunque una partida al título por primera vez podrá rondar la hora u hora y media, dependiendo de la habilidad del jugador), y las versiones ejecutables para PC (32 y 64 bits) del software que ha sido generado con este proyecto.

■ **Vídeo demostrativo del proyecto:**

- **Enlace a Youtube:** <https://www.youtube.com/watch?v=i5G1jZ5wXPU>
- https://mega.nz/#!N3pwBB6L!Hd50wmsDTbDKm7v-t4PsaCmiZ1vf_rAvTsw4_GIsCQ4

■ **Descarga versión ejecutable - Windows 32 bits:**

- https://mega.nz/#!53JmjBgD!7CmT5Wfwxis2ndt9EdkPSimejg_N17TIItqj3Zx1Y1Ik

■ **Descarga versión ejecutable - Windows 64 bits (recomendado):**

- https://mega.nz/#!FuwwBTyZ!n_uz1AywgEgPJL9ARe8ZaLd_1wWQUrHjj9YaduCQpbM

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.	Introducción	26
3.2.	Estructura organizacional del proyecto	27
3.3.	Metodología de desarrollo	28

3.3.1. Presentación de la metodología	28
3.3.2. Resumen de la metodología	28
3.3.3. Adaptación de la metodología	30
3.4. Conclusiones	32
4. Planificación	33
4.1. Introducción	34
4.2. Resumen temporal del proyecto	35
4.3. Planificación inicial	35
4.4. Informe de tiempos del proyecto	41
4.5. Conclusiones	44
5. Costes	47
5.1. Resumen de costes del proyecto	48
5.2. Costes de personal	48
5.3. Costes materiales	55
5.4. Costes indirectos	58
5.5. Costes directos e inmateriales	60
5.6. Conclusiones	61
5.6.1. Desviación de costes	61
III Desarrollo del proyecto	65
6. Arranque	67
6.1. Lista de características	68
6.2. Diseño arquitectónico	71

6.2.1. Módulos de juego	71
6.2.2. Clases de juego	73
7. Iteración cero	75
7.1. Inicialización del proyecto	76
7.2. Directrices generales	78
7.2.1. Definición de controles	78
7.2.2. Adaptación de animaciones no compatibles	79
7.2.3. Procesado de animaciones carentes de nodo raíz	82
7.2.4. Sistema de animación del personaje	84
8. Primera iteración	89
8.1. Introducción	90
8.2. Características a desarrollar	92
8.3. Diseño	94
8.4. Implementación	107
8.5. Pruebas	115
8.6. Despliegue	122
9. Segunda iteración	125
9.1. Introducción	126
9.2. Características a desarrollar	130
9.3. Diseño	132
9.3.1. Modificaciones	139
9.4. Implementación	140
9.5. Pruebas	155

ÍNDICE GENERAL

9.6. Despliegue	163
10. Tercera iteración	165
10.1. Resumen	166
10.2. Características a desarrollar	168
10.3. Diseño	170
10.4. Implementación	175
10.5. Pruebas	182
10.6. Despliegue	186
11. Cuarta iteración	187
11.1. Introducción	188
11.1.1. Sistema de puzzles	188
11.1.2. Peligros a implementar	192
11.2. Características a desarrollar	193
11.3. Diseño	194
11.4. Implementación	195
11.5. Pruebas	196
11.6. Despliegue	197
12. Quinta iteración	199
12.1. Características a desarrollar	200
12.2. Diseño	200
12.3. Implementación	200
12.4. Pruebas	200
12.5. Despliegue	200

13. Sexta iteración	201
13.1. Características a desarrollar	202
13.2. Diseño	202
13.3. Implementación	202
13.4. Pruebas	202
13.5. Despliegue	202
IV Cierre del proyecto	203
14. Manual de usuario	205
14.1. Controles	206
14.1.1. Controles con teclado y ratón	206
14.1.2. Controles con mando	208
14.2. Menú principal	210
14.3. Menú de pausa	211
14.4. Menú de ajustes	212
14.5. HUD del usuario	214
15. Conclusiones	219
15.1. Informe post-mortem	220
15.1.1. Lo que ha ido bien	220
15.1.2. Lo que ha ido mal	223
15.2. Discusión	227
15.3. Trabajos futuros	228

V Appendices	231
A. Glosario	233
B. Documento de diseño	239
B.1. Visión de conjunto	240
B.1.1. Historia	240
B.1.2. Iconografía	241
B.1.3. Plataformas de destino	242
B.1.4. Requisitos mínimos objetivo	242
B.1.5. Estilo gráfico	242
B.1.6. Estilo sonoro	243
B.2. Comienzo	243
B.2.1. Menú principal	243
B.3. Interfaz de usuario	243
B.3.1. Menús	243
B.3.2. In-game HUD	243
B.3.3. In-game Options	244
B.3.4. Pantalla de fin de juego	244
B.3.5. Pantalla de selección de nivel	244
B.4. Gameplay	245
B.4.1. Mecánicas	245
B.4.2. Controles	250
B.4.3. Modos de juego	250
B.4.4. Ganando el juego	250
B.4.5. Logros / Trofeos	251

B.5. Assets	260
B.5.1. Personajes	260
B.5.2. Enemigos	261
B.5.3. Animaciones	262
B.5.4. Mejoras de personaje / equipo / habilidades	262
B.5.5. Assets de escenarios	263
B.5.6. Assets variados	263
B.6. Cámaras disponibles	264
B.7. Sistema de guardado	264
Referencias bibliográficas	264

ÍNDICE DE FIGURAS

1.1. Beneficios de los videojuegos en Europa Occidental (datos en billones americanos) [20]	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 [19]	7
1.5. Comparativa, beneficios y previsiones entre los juegos para móvil y el resto de aplicaciones (datos en billones americanos) [18]	8
1.6. Relevancia del programa «Greenlight» en los lanzamientos indies [29]	10
1.7. Ejemplos de videojuegos indies se hicieron un hueco en el mercado	11
1.8. Captura del juego «E-Motion» (1990), para «ZX Spectrum», 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 [12]	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

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	27
3.2. Esquema del desarrollo basado en funcionalidades	29
3.3. Esquema del desarrollo basado en funcionalidades adaptado para el proyecto	31
4.1. Equivalencias fases-metodología	37
4.2. Horas planificadas contra extras	44
4.3. Horas de adelanto o retraso acumuladas	45
5.1. Diferencias euro-dólar entre Agosto de 2015 y Julio de 2017 [16]	51
5.2. Diferencia entre el valor presupuestado y el real	63
6.1. Arquitectura Unreal Engine 4 [11]	71
6.2. Estructura del proyecto	72
7.1. Ventana del creador de proyectos de Unreal Engine	76
7.2. Captura de la ventana del editor de «bindings» o enlaces	78
7.3. Captura del esqueleto destino adaptado	80
7.4. Captura del esqueleto origen adaptado y con las opciones de retargeting fijadas	81
7.5. Captura del editor de animaciones de Unreal Engine, mostrando el proceso de re-asignación	81
7.6. Captura de la aplicación «Autodesk Maya», mostrando el hueso «root» ya añadido a un esqueleto	82
7.7. Vista de una pequeña parte de la máquina de estados implementada para animar al personaje del proyecto	84
7.8. Captura de una animación mostrando las notificaciones de sonido	85

7.9. Imagen de ejemplo de «blend space» en 2D: en función de la dirección y la velocidad	86
7.10. Ejemplo de «animation montage», usado cuando el protagonista escala un saliente	87
8.1. Fotograma de los componentes básicos del personaje: capsula de colisión, modelo, cámara en primera persona real y cámara en tercera persona	90
8.2. Fotograma del mapa de pruebas empleado en la iteración 1	115
8.3. Configuración de empaquetamiento. Marcada en rojo la opción por defecto («Development»)	122
8.4. Ruta para generar una build del proyecto	123
9.1. Imagen en la que podemos apreciar la esfera detectora de superficie escalable	126
9.2. Configuración de canales del proyecto	127
9.3. Fotograma del mapa de pruebas empleado en la iteración 2	155
10.1. Captura de un «checkpoint» (o punto de control) usado en el juego	166
10.2. Imagen de un cristal de teletransportación rojo, usado para avanzar al siguiente nivel	167
11.1. Representación de la herencia de los «powerables»	189
11.2. Representación de la herencia de los «activatables»	190
14.1. Diagrama de la asignación de acciones del teclado	206
14.2. Diagrama de la asignación de acciones del ratón	206
14.3. Diagrama de la asignación de acciones del controlador	208
14.4. Fotograma del menú principal del software	210
14.5. Fotograma del menú de pausa del software	211
14.6. Fotograma del menú de ajustes del software	212

ÍNDICE DE FIGURAS

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	214
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)	215
14.9. Captura del juego tomada durante la obtención de un logro	216
14.10. Captura del juego tomada durante un tutorial	218
15.1. Captura del menú principal durante versiones tempranas del desarrollo	225
15.2. Fotograma del título «Fallout 4» (2015) que muestra una farola que emite un cono de luz	230
B.1. Logo del juego	241
B.2. Logo del juego	241

ÍNDICE DE CUADROS

3.1. Manifiesto ágil	26
4.1. Tabla resumen de tiempos y planificación	35
4.2. Planificación temporal de fases e iteraciones (planificación)	38
4.3. Planificación temporal detallada de fases e iteraciones	39
4.4. Tabla del calculo de horas por rol en iteraciones	40
4.5. Tabla resumen de horas de trabajo por rol	40
4.6. Tabla de tiempos empleados en fases e iteraciones detallados	41
4.7. Tabla de desviación de tiempo por iteración	42
4.8. Tabla de desviación de tiempo por rol	42
4.9. Tabla de retrasos del proyecto	43
5.1. Tabla resumen de costes	48
5.2. Salarios medios brutos según la encuesta salarial de «Gamasutra» (2014) [14]	49
5.3. Tabla comparativa costes de vida [21]	50
5.4. Salarios medios brutos (adaptados) según la encuesta salarial de «Gamasutra» (2014) [14]	52
5.5. Tabla salarial (sueldos brutos) del proyecto por hora	53
5.6. Tabla costes sueldos brutos	53
5.7. Tabla de costes de personal	54
5.8. Tabla de características del dispositivo sobremesa	56

5.9. Tabla de características del dispositivo portátil	56
5.10. Tabla de amortización de materiales	57
5.11. Tabla de costes indirectos	58
5.12. Tabla de costes directos e inmateriales	60
5.13. Tabla de desviación de tiempo por rol	61
5.14. Tabla costes sueldos brutos (tiempo extra)	62
6.1. Prefijos de clases de juego en Unreal Engine [11]	73
8.1. Memorando técnico 1-001: Creación del personaje	94
8.2. Memorando técnico 1-002: Creación de la cámara TFP	95
8.3. Memorando técnico 1-003: Creación de la cámara TP	96
8.4. Memorando técnico 1-004: Movimiento del personaje	97
8.5. Memorando técnico 1-005: Rotación de cámara y personaje	98
8.6. Memorando técnico 1-006: Rotación de cámara y personaje (para controladores)	99
8.7. Memorando técnico 1-007: Sprint	100
8.8. Memorando técnico 1-008: Salto	100
8.9. Memorando técnico 1-009: Agachar	101
8.10. Memorando técnico 1-010: Tiempo bala	102
8.11. Memorando técnico 1-011: Teletransporte o «blink»	103
8.12. Memorando técnico 1-012: Control de límites del mapa	104
8.13. Memorando técnico 1-013: Control de máxima distancia de caída	104
8.14. Memorando técnico 1-014: Animación de rotación del personaje al girarse.	105
8.15. Memorando técnico 1-015: Sistema de gestión de adrenalina.	106
8.16. Batería de pruebas: Cámara «primera persona real» (1-002)	116

8.17. Batería de pruebas: Cámara «tercera persona» (1-003)	116
8.18. Batería de pruebas: Movimiento del personaje y sprint (1-004 y 1-007) . . .	117
8.19. Batería de pruebas: Rotación de cámara y personaje (1-005 y 1-014) . . .	118
8.20. Batería de pruebas: Rotación de cámara y personaje con controlador (1-006 y 1-014)	118
8.21. Batería de pruebas: Salto (1-008)	119
8.22. Batería de pruebas: Agachar (1-009 y 1-014)	119
8.23. Batería de pruebas: Tiempo bala (1-010)	120
8.24. Batería de pruebas: Teletransporte o «blink» (1-011)	120
8.25. Batería de pruebas: Control de límites del mapa (1-012)	121
8.26. Batería de pruebas: Control de máxima distancia de caída (1-013) . . .	121
 9.1. Trazadores de la base del sistema de escalada 1/2	128
9.2. Trazadores de la base del sistema de escalada 2/2	129
9.3. Memorando técnico 2-001: Base del sistema de escalada	132
9.4. Memorando técnico 2-002: Agarrar un saliente	133
9.5. Memorando técnico 2-003: Agarrar un saliente pequeño para escalarlo .	133
9.6. Memorando técnico 2-004: Escalar un saliente al que estemos agarrados	134
9.7. Memorando técnico 2-005: Soltar un saliente	134
9.8. Memorando técnico 2-006: Movernos por un saliente	135
9.9. Memorando técnico 2-007: Saltar desde saliente lateralmente	135
9.10. Memorando técnico 2-008: Mirar hacia atrás desde saliente	136
9.11. Memorando técnico 2-009: Saltar desde saliente mirando hacia atrás .	137
9.12. Memorando técnico 2-010: Escalada en pared	138
9.13. Memorando técnico 2-011: Giro y salto durante escalada en pared . . .	138
9.14. Batería de pruebas: Agarrar un saliente (2-002)	156

ÍNDICE DE CUADROS

9.15. Batería de pruebas: Auto-escalar un saliente (2-003)	157
9.16. Batería de pruebas: Escalar un saliente al que estemos agarrados (2-004)	157
9.17. Batería de pruebas: Soltar un saliente (2-005)	158
9.18. Batería de pruebas: Movernos por un saliente (2-006)	159
9.19. Batería de pruebas: Saltar desde saliente lateralmente (2-007)	160
9.20. Batería de pruebas: Mirar hacia atrás desde saliente (2-008)	160
9.21. Batería de pruebas: Saltar desde saliente mirando hacia atrás (2-009)	161
9.22. Batería de pruebas: Escalada en pared (2-010)	161
9.23. Batería de pruebas: Giro y salto durante escalada en pared (2-011)	162
10.1. Memorando técnico 3-001: Sistema de guardado / carga	170
10.2. Memorando técnico 3-002: Sistema de muerte	171
10.3. Memorando técnico 3-003: Opción menú - Nuevo juego	171
10.4. Memorando técnico 3-004: Opción menú - Cargar juego	172
10.5. Memorando técnico 3-005: Opción menú - Ajustes	172
10.6. Memorando técnico 3-006: Opción menú - Salir	173
10.7. Memorando técnico 3-007: Opción menú - Cerrar menú	173
10.8. Memorando técnico 3-008: Opción menú pausa - Volver al menú principal	174
10.9. Batería de pruebas: Sistema de guardado / carga (3-001)	182
10.10Batería de pruebas: Sistema de muerte (3-002)	183
10.11Batería de pruebas: Opción menú - Nuevo juego (3-003)	183
10.12Batería de pruebas: Opción menú - Cargar juego (3-004)	184
10.13Batería de pruebas: Opción menú - Ajustes (3-005)	184
10.14Batería de pruebas: Opción menú - Salir (3-006)	184
10.15Batería de pruebas: Opción menú pausa - Cerrar menú (3-007)	185

10.16Batería de pruebas: Opción menú pausa - Volver al menú principal (3-008)	185
11.1. Tabla de pesos de las estatuillas «pickables»	191
B.1. Logro historia - Juego	251
B.2. Logro historia - Tutorial	251
B.3. Logro historia - Capítulo I	252
B.4. Logro historia - Capítulo II	252
B.5. Logro prueba de tiempo - Capítulo I	253
B.6. Logro prueba de tiempo - Capítulo II	253
B.7. Logro colecciónables - 1 colecciónable	254
B.8. Logro colecciónables - 5 colecciónables	254
B.9. Logro colecciónables - 10 colecciónables	255
B.10. Logro colecciónables - 50 % de colecciónables	255
B.11. Logro colecciónables - 100 % de colecciónables	256
B.12. Logro misceláneos - 100 saltos	257
B.13. Logro misceláneos - 1000 saltos	257
B.14. Logro misceláneos - 10000 saltos	258
B.15. Logro 100 %	259
B.16. Assets de personajes	260
B.17. Assets de enemigos	261
B.18. Tabla descripción de archivos de guardado	264

PARTE I

INTRODUCCIÓN

CONTEXTO

En este primer capítulo abordaremos el entorno que rodea al producto software que pretendemos crear, la industria del videojuego, con el objetivo de situarnos para conocer las oportunidades que ofrece el mercado y los riesgos, relacionados con el mismo, que 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, 20] 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 [20].

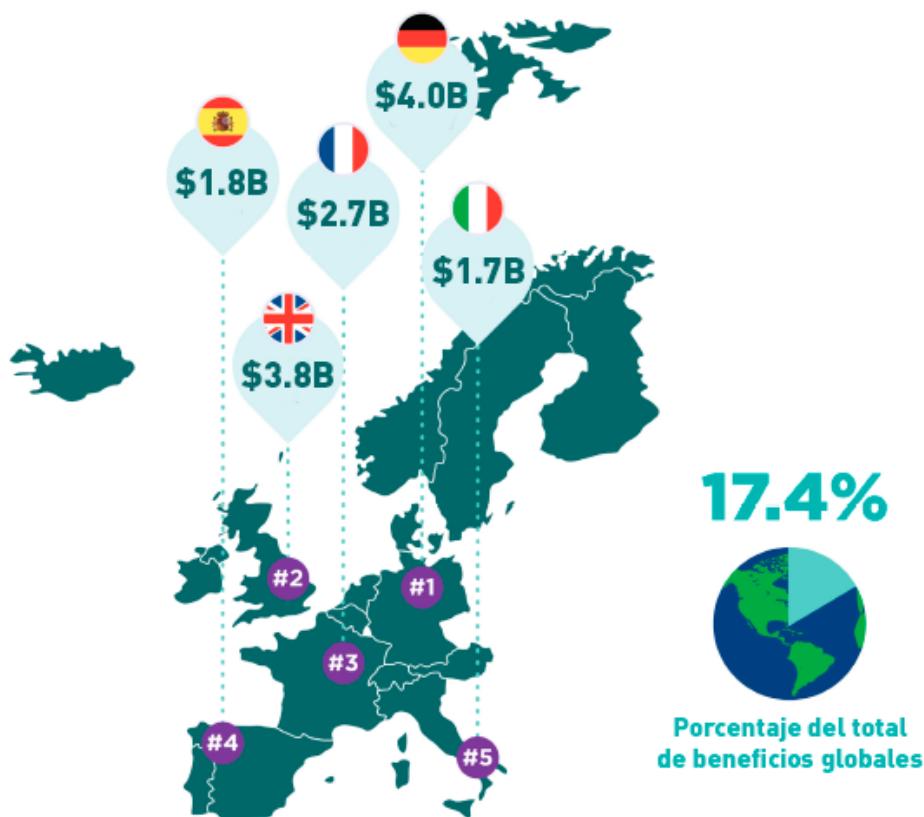


Figura 1.1: Beneficios de los videojuegos en Europa Occidental (datos en billones americanos) [20]

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



Figura 1.2: Evolución gráfica de los videojuegos

Aún queda un largo camino por recorrer, es un problema que persigue al sector desde su creación, pero en los últimos años hemos vivido una apertura de los videojuegos a la sociedad de tal magnitud que una parte relevante de la misma lo ha pasado a ver de una forma distinta, dejando de lado ese citado recelo o cualquier connotación negativa, lo cual sin duda no puede ser más que favorable para la industria y su 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 [27] 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 del 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].

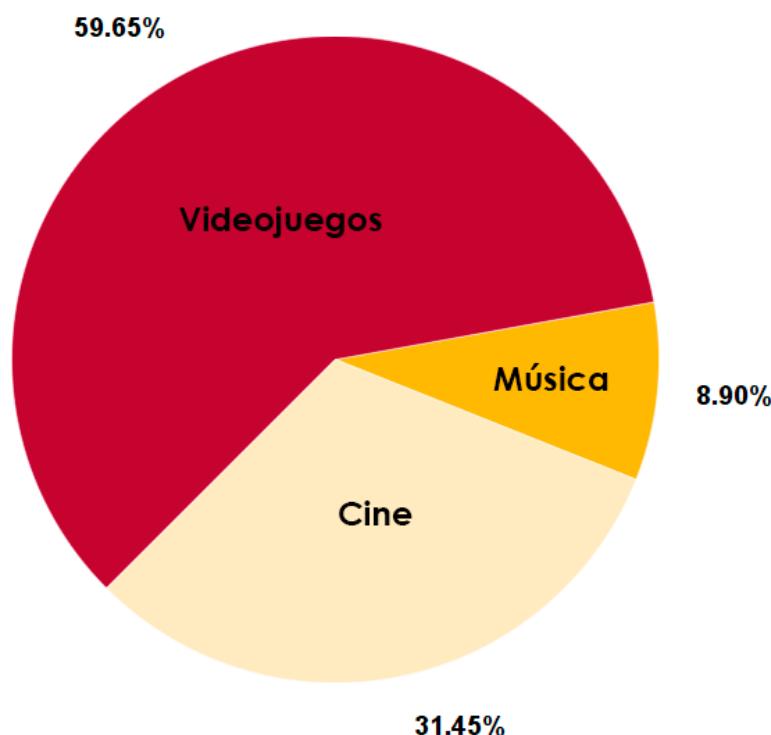


Figura 1.3: Relación entre el dinero generado por los videojuegos, la música grabada y el cine durante 2015 en España [8]

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 [19].

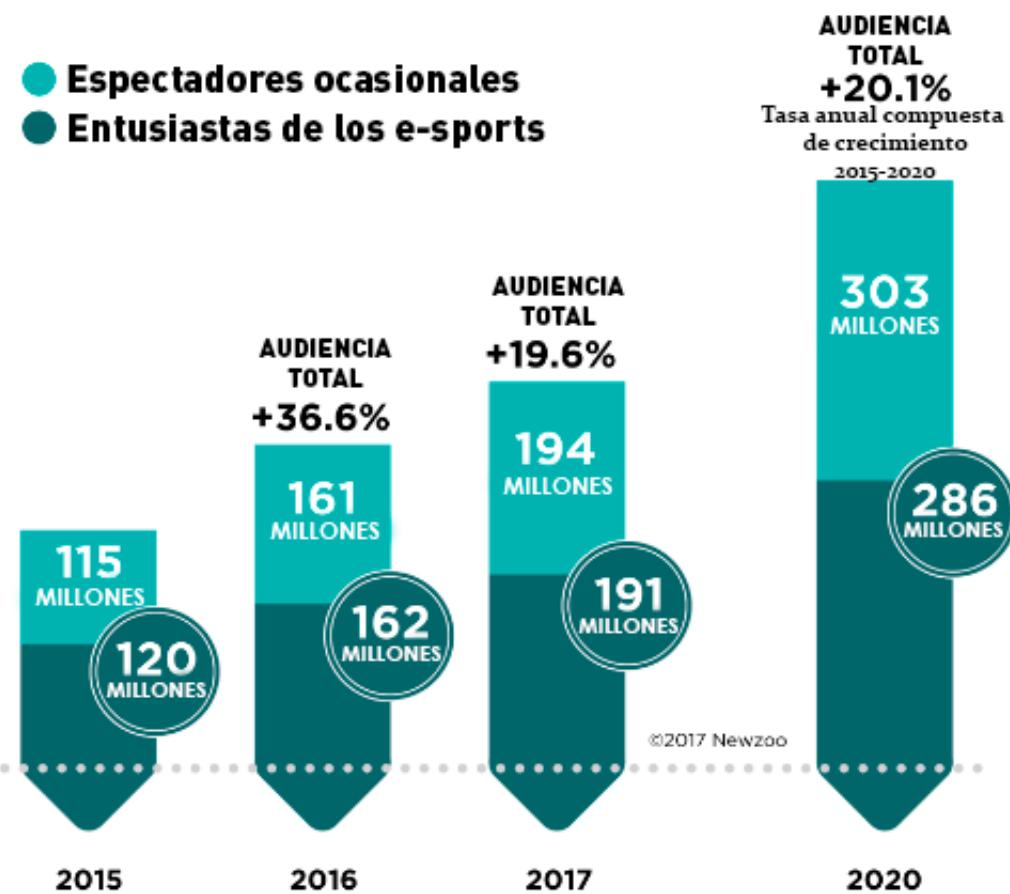


Figura 1.4: Audiencia y previsión de crecimiento de los espectadores de deportes electrónicos [19]

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 entran.

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 [18, 20], lo que supone en torno al 37% de los beneficios totales de la industria obtenidos el pasado año [20].



Figura 1.5: Comparativa, beneficios y previsiones entre los juegos para móvil y el resto de aplicaciones (datos en billones americanos) [18]

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

Este cambio en la accesibilidad amplía no sólo el espectro de jugadores que llegarán a poseer una consola o un ordenador gaming para tener más títulos a su disposición, generando beneficios para la industria en el proceso, sino que llega a sectores de la población a los que antes era difícil acceder, como el de las personas de mediana edad, y provocan cambios en las tendencias tales como que las mujeres igualen a los hombres [28], 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, los indies 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 de la 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

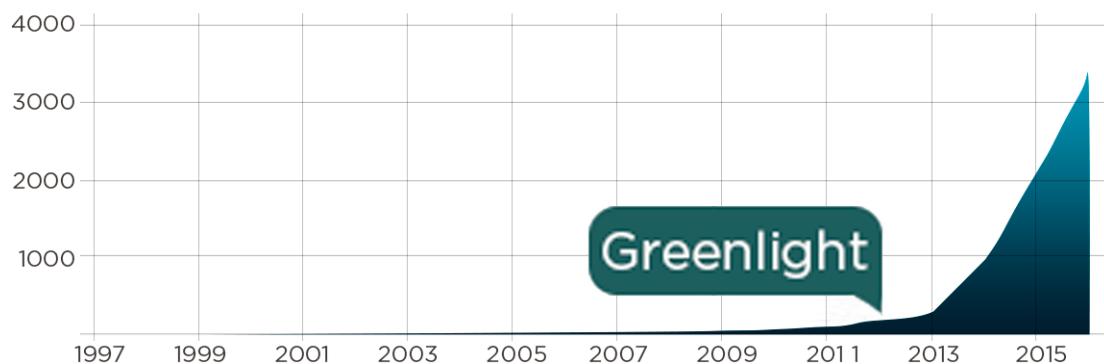


Figura 1.6: Relevancia del programa «Greenlight» en los lanzamientos indies [29]

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, 9]. 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.

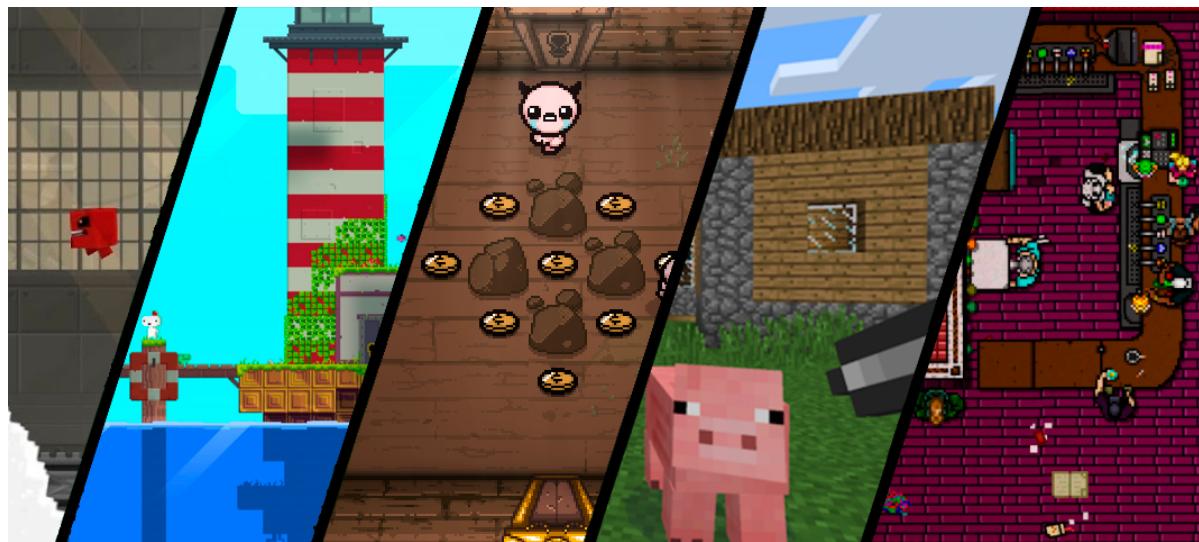


Figura 1.7: Ejemplos de videojuegos indies se hicieron un hueco en el mercado

Cabe remarcar que recientemente, desde Junio de 2017, el programa de «VALVe» para sus juegos independientes de «Steam» denominado «Steam Greenlight» no acepta nuevas aportaciones. Esto es porque la compañía está haciendo algunas modificaciones y el sistema pasará a denominarse «Steam Direct». El principal cambio será que los desarrolladores tendrán que abonar una fianza 100\$ por juego, en lugar de realizar un único pago como en «Greenlight» y que permitía publicar tantos juegos en el programa como se quisiese, y al recaudar 1000\$ en beneficios la fianza será devuelta. El principal motivo de esto es mejorar la calidad de los juegos que se incorporan finalmente a la plataforma al salir del programa de lanzamientos [23, 24].

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

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

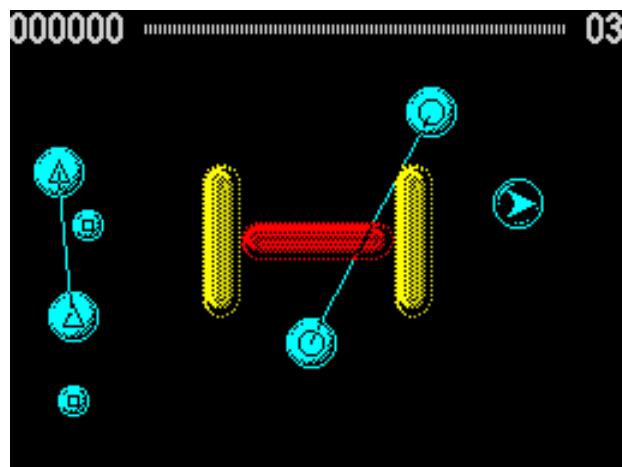


Figura 1.8: Captura del juego «E-Motion» (1990), para «ZX Spectrum», uno de los primeros juegos en implementar algún tipo de logro

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) [22] 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 [15].

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 [12]

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»

Volviendo a nuestros días y centrándonos en lo que la industria supone en nuestro país, cada vez las universidades y academias ofertan más titulaciones o cursos relacionados con los videojuegos, y esto ayuda a que poco a poco el tejido empresarial relacionado con la industria del videojuego vaya creciendo [27]. La existencia de poco tejido empresarial no ha impedido que en el pasado, durante la corta historia de la industria, en nuestro país hayan aparecido títulos de primer orden a nivel internacional: el caso más conocido es el de la saga «Commandos», que hizo su aparición a finales de los noventa y, más recientemente, un título que podríamos catalogar de súper producción, «Castlevania: Lord of the Shadows» (2010).

Aunque hay excepciones, la notable la mayoría de los juegos actuales son realizados a partir de un motor gráfico comercial ya definido. Estos motores suelen ser propiedad de la compañía, o bien de terceros bajo acuerdo, pero en los últimos años han surgido varios motores de uso gratuito que han ganado enorme popularidad y que hacen más accesible el desarrollo de videojuegos, entre los que encontramos Unreal Engine 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].

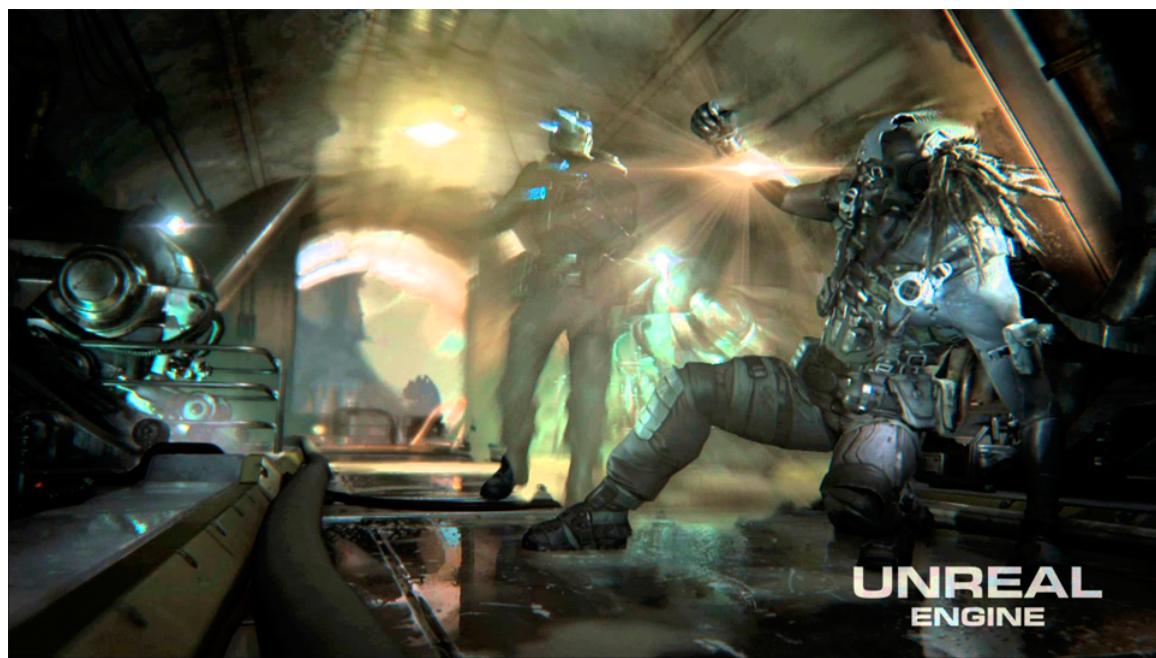


Figura 1.12: Imagen de la primera demostración técnica en tiempo real de Unreal Engine 4, titulada «Infiltrator»

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

La siguiente sección presenta más detalladamente el proyecto, la razón de ser del mismo y un listado de objetivos, tanto del producto como individuales, que se pretenden 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.

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.

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.

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.

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.

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

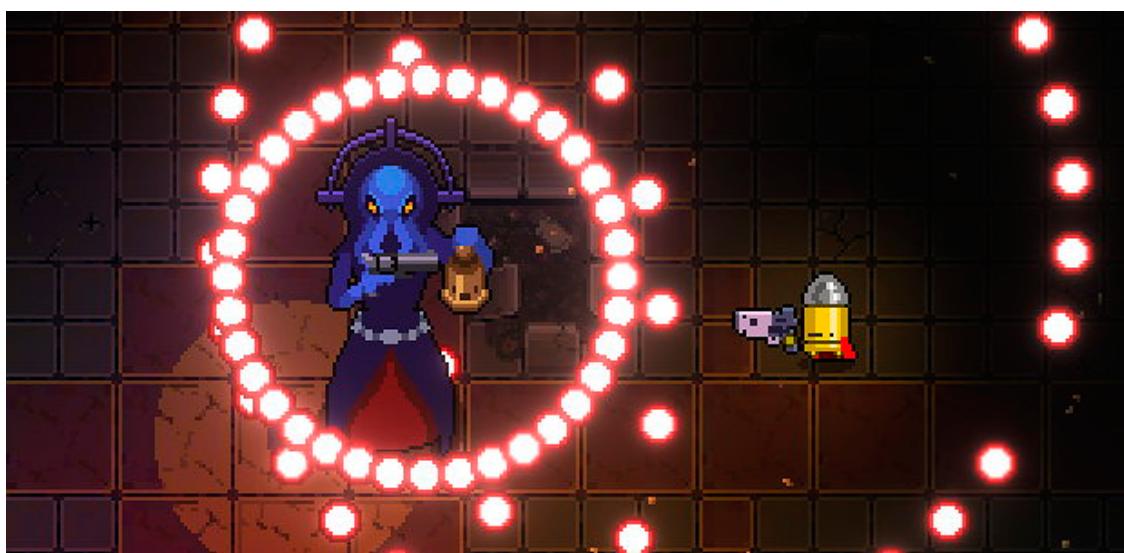


Figura 2.2: Captura de «Enter the gungeon» (2016), ejemplo de la forma más tradicional de bullet hell en dos dimensiones

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 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.

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

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

Sistema de guardado / carga: El producto debe ser capaz de guardar y cargar los avances del jugador cuando así se requiera. En concreto se hará uso del auto-guardado cuando el jugador entre en contacto con la superficie asociada a un punto de control.

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

Optimización: Optimizar lo máximo posible realizando todo el contenido posible en C++ ya que como explicaba anteriormente (tomando la propia Epic Games como fuente, desarrolladora de Unreal Engine), los tiempos de ejecución de ese código se reducen en torno a 10 veces usando el lenguaje de programación en C++ sobre 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

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

3.1 INTRODUCCIÓN

El uso de una metodología es esencial cuando estamos hablando de ingeniería software: ayudan a planear y controlar el proceso de desarrollo del software, así como a gestionar los recursos de los que se disponen.

En cuanto a metodologías existen dos grandes corrientes: la tradicional y la denominada ágil. Las principales diferencias entre las ágiles y las tradicionales se recogen en el manifiesto ágil:

Manifiesto ágil: metodologías ágiles contra tradicionales		
Individuos e interacciones	«sobre»	Procesos y herramientas
Software funcionando	«sobre»	Documentación exhaustiva
Colaboración con el cliente	«sobre»	Negociación contractual
Respuesta ante el cambio	«sobre»	Seguir un plan

Cuadro 3.1: *Manifiesto ágil*

Las secciones que componen este capítulo son, en detalle:

- La **estructura organizacional** (§3.2), en este caso simple, con la que se ha desarrollado este proyecto.
- Una **presentación de la metodología** que vamos a usar (§3.3.1), en la que se describen de forma breve su nacimiento, razón de ser e historia.
- Un **resumen de la metodología** (§3.3.2), donde se introduce el concepto de «Feature-Driven Development» o «Desarrollo basado en funcionalidades», se explica y se identifica los roles que lo conforman.
- La **adaptación de la metodología** (§3.3.3) y sus roles que se han usado para desarrollar este proyecto.
- Y, por último, las **conclusiones sobre la metodología** (§3.4) en el que se extraerán unas breves conclusiones.

3.2 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

3.3 METODOLOGÍA DE DESARROLLO

3.3.1 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 [7].

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 [25].

3.3.2 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, 26]:

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.

3.3.3 Adaptación de la metodología

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

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.
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.
3. **Planificación por funcionalidad:** Nuevamente como en la metodología FDD tradicional, ordenamos las funcionalidades resultantes según su prioridad y teniendo en cuenta sus dependencias.
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.
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.

3.4 CONCLUSIONES

La principal peculiaridad que nos encontramos en el proyecto es que está compuesto por un solo integrante, lo que tiene algunas ventajas, como se describe en la estructura organizacional (Ver sección §3.2), pero también ciertos inconvenientes si queremos implantar una metodología en la que intervienen tantos roles, lo que nos obliga a «simular» desempeñar más de un rol y, por tanto, a tener siempre en mente a qué rol pertenece la tarea que estamos realizando, lo que supone un inconveniente en campos como el registro de tiempos.

Por otra parte, se ha podido adaptar correctamente la metodología a nuestro caso específico sin mucho esfuerzo: el principal cambio reside en el cambio de realizar el «modelo global del sistema» por el «documento de diseño del videojuego», que es lo que se hace en el mundo del desarrollo de videojuego por lo que, tras este cambio, encaja a la perfección.

Y, por último, también se ha podido adaptar correctamente los roles, cuyo cambio más drástico ha sido la incorporación del «diseñador».

En definitiva, podemos concluir que es una metodología que, tras su adaptación, concuerda perfectamente con el mundo del desarrollo de videojuegos y por tanto nos será de gran ayuda durante el desarrollo de este software.

PLANIFICACIÓN

En la siguiente sección se expone la planificación del proyecto y el informe de tiempos generado durante el transcurso del mismo que, comparado con lo anterior, nos permitirá ver la desviación en cuanto a tiempo del proyecto.

4.1 INTRODUCCIÓN

La planificación es una de las fases más importantes en el desarrollo software: una mala planificación puede abocar al fracaso un proyecto en el que hayamos invertido mucho dinero y recursos.

Las secciones de este capítulo recogen:

- Una **tabla resumen de la planificación** del proyecto (§4.2), que nos permitirá obtener de un vistazo aspectos fundamentales sobre la planificación de este proyecto.
- La **planificación inicial** del proyecto (§4.3), en la cuál se nos muestra la planificación como tal y los criterios y decisiones que nos han llevado a estructurarla de la forma en la que se ha hecho.
- El **informe de tiempos** (§4.4), que contiene los datos de tiempo reales obtenidos durante el transcurso del proyecto, así como las desviaciones y adelantos o retrasos que ha sufrido el proyecto.
- Por último, en la sección **conclusiones** (§4.5) se discute la planificación del proyecto, comparando los datos planificados con los obtenidos tras la finalización del proyecto.

4.2 RESUMEN TEMPORAL DEL PROYECTO

Resumen del proyecto	
Fecha de inicio	20/02/2017
Fecha de fin	31/08/2017
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

4.3 PLANIFICACIÓN INICIAL

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:

- **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.

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).

- **Iteraciones 1-6:** Son las etapas más puramente de desarrollo. Intervienen tanto el **diseñador**, que será el encargado de preparar las iteraciones, como el **programador**, que será quien codifique y genere documentación de las iteraciones (como ya se explicó en la sección §3.3.3), además del **jefe de proyecto** que tendrá una labor de supervisión (principalmente gestionar tiempos empleados y retrasos). A continuación, se describen muy escuetamente las iteraciones (para más información, consultar el documento de diseño §B):
 - **Iteración 1. Mecánicas básicas y poderes:** La primera iteración se implementarán las mecánicas más básicas, como saltar, agacharse, correr, etc. y, además, las mecánicas de las que podrá hacer uso el personaje cuando esté potenciado.
 - **Iteración 2. Mecánicas de escalada:** En la segunda iteración se implementarán las mecánicas de «plataformeo» avanzadas.
 - **Iteración 3. HUD, menús, sistema de muerte y sistema de guardado:** Durante la tercera iteración se creará el menú principal, el menú de pausa y el HUD del que dispondrá el usuario. Además, se implementará el sistema de muerte y el de guardado.
 - **Iteración 4. Peligros del entorno:** La cuarta iteración tratará sobre la implementación los peligros del entorno que deberá esquivar el personaje.
 - **Iteración 5. Enemigos e inteligencia artificial:** Durante la quinta iteración se crearán los enemigos que aparecen en el videojuego y se implementará la IA de los mismos.
 - **Iteración 6. Diseño de escenarios:** En la sexta y última iteración se llevarán a cabo la creación de los escenarios de los que dispondrá, a modo de demostración, el videojuego a entregar. Esto incluye el diseño de un tutorial.
- **Fase de cierre del proyecto:** En esta iteración intervienen el **diseñador**, encargado de elaborar el manual de la aplicación, y el **jefe de proyecto**, que será quien finalice los últimos apartados de la memoria, haga las últimas modificaciones sobre la misma y posteriormente realice la presentación mediante la cual se expondrán los resultados del proyecto.

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

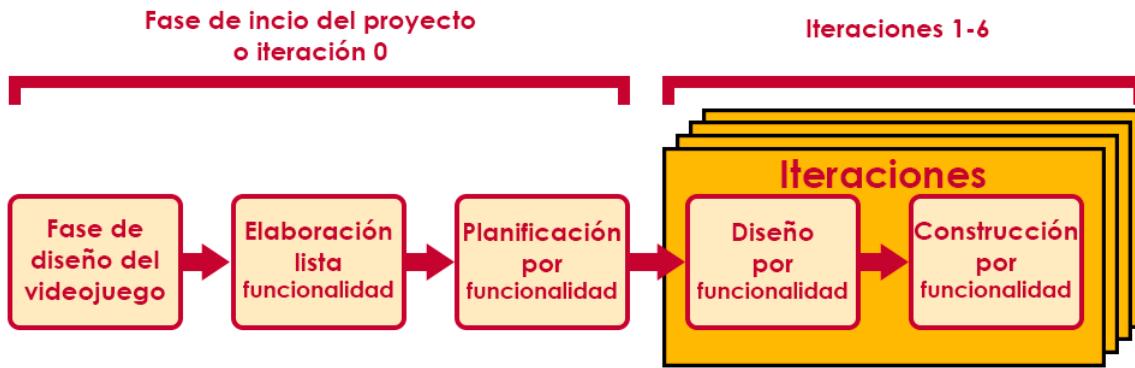


Figura 4.1: Equivalencias fases-metodología

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.

La siguiente tabla muestra un resumen de la planificación proyectada para las iteraciones 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)

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

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

Además, como también figura en la tabla, se ha decidido establecer un periodo denominado «**Red de seguridad**» a modo de medida de contingencia para paliar cualquier retraso que surja durante la ejecución del proyecto. Esto es una medida recomendable en todos los proyectos pero más antes el que nos encontramos, ya que es el primer proyecto de este tipo / tecnología. En concreto tendremos un colchón de seguridad 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:

Cálculo 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

4.4 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

La tabla que se muestra a continuación muestra la **desviación de tiempo** por iteración que se ha obtenido (en la cuál 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
<i>Desviación final</i>		+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	Decisión tomada durante el desarrollo: Como llegados a esta iteración aún había suficiente margen de tiempo, se prefirió dotar a los escenarios de los niveles ordinarios, que en principio iban a tener un acabado de «prototipos», de mejoras visuales para que todos los niveles del producto tuvieran un acabado homogéneo.

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.

4.5 CONCLUSIONES

Uno de los datos más importantes que tenemos que analizar es la desviación en cuanto a tiempo total que se ha producido al término del proyecto. En este caso, han sido necesarias **31,5 horas** más de las planificadas para finalizar el proyecto, lo que supone un 10% sobre el total de horas planeadas en un principio (que fueron un total de 310). Esto se debe en gran parte a la pequeña re-planificación que se ha efectuado durante la última iteración para añadir «funcionalidad» extra, ya que recordemos que se decidió alargar una semana la ésta iteración para mejorar los escenarios de juego que componen el software, aprovechando la red de seguridad de la que disponíamos. Además, teniendo en cuenta que es el primer proyecto con esta tecnología y de este tipo no es una diferencia demasiado significativa o que no fuese esperable en un principio.

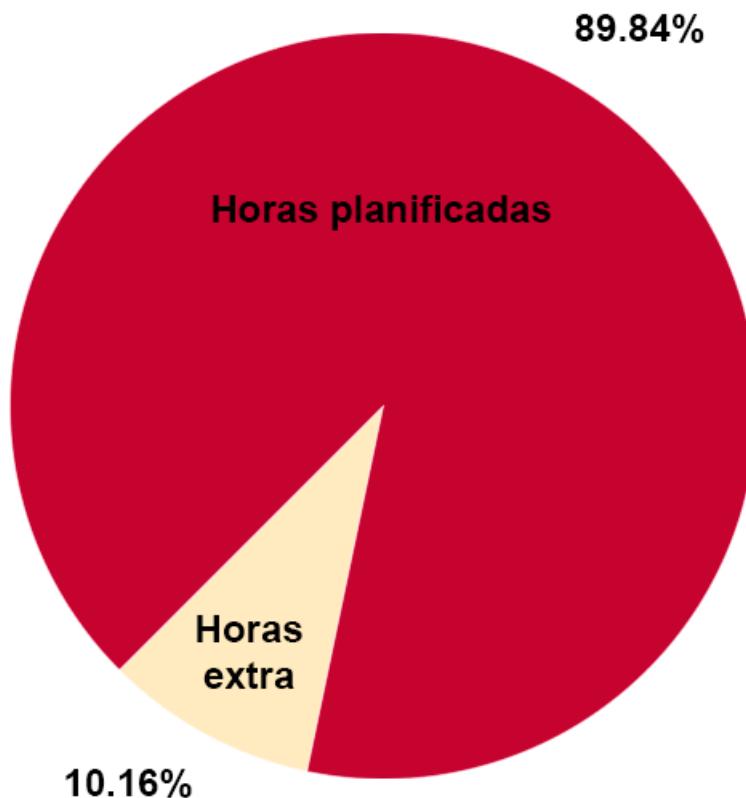


Figura 4.2: Horas planificadas contra extras

Por otra parte, podemos afirmar que el mayor éxito ha consistido en dividir satisfactoriamente el trabajo entre las diferentes iteraciones, identificando cada tarea (o al menos la inmensa mayoría de ellas) desde el principio, lo que nos ha permitido esquivar de lleno el denominado «síndrome del noventa por ciento» [17].

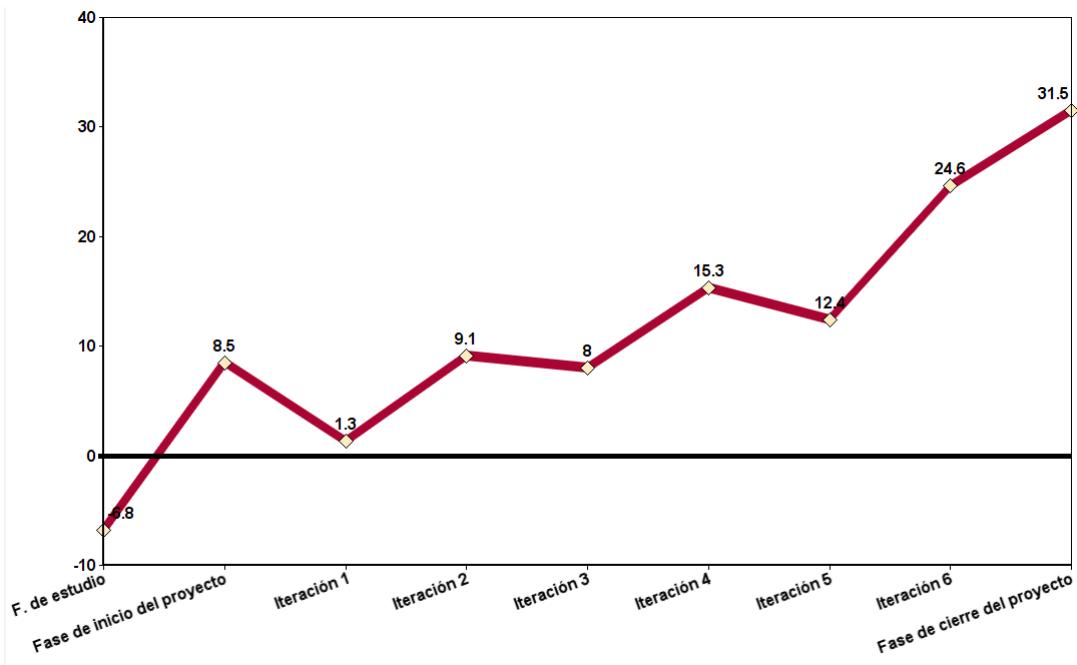


Figura 4.3: Horas de adelanto o retraso acumuladas

La amplia «red de seguridad» establecida durante la planificación ha permitido realizar este trabajo sin demasiadas prisas, evitando gran parte de estrés (que nunca se proyecta bien en el resultado final), y además, como se comentaba, ha permitido realizar mejoras al producto final que no estaban planificadas en un principio y que, bajo mi punto de vista, han merecido la pena.

Respecto a los roles debemos concluir que los roles que más horas han desempeñado han sido, por este orden: programador, jefe de proyecto y diseñador, como así se había planificado en un principio. Los tres roles trabajaron más horas de las proyectadas, principalmente por el motivo que se ha expresado antes, siendo el jefe de proyecto el que más se ha desviado de sus horas planificadas, seguido del diseñador y del programador, este último con una diferencia mínima.

En definitiva podemos concluir que se ha llevado a cabo una buena planificación: las horas de más empleadas, en su inmensa mayoría, son fruto de la necesidad de querer mejorar este producto lo mayor posible, un software que ha sido tratado con mimo y cariño desde el principio y que espero que ello se note en el resultado final. Prueba de este hecho ha sido evitar el citado anteriormente «síndrome del 90%» [17] ya que de ser de otra forma debería haberse hecho mucho más notable en un proyecto en el que se han empleado más horas de las planificadas.

COSTES

En este capítulo abordaremos una de las partes más importantes en la planificación de un proyecto: los costes, y para ello los dividiremos en costes de personal, de material e 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	10.183,16 €
Sueldo bruto	7.833,2 €
Costes Seg. Social	2.349,96 €
Costes materiales	357,78 €
Costes indirectos	4.127,34 €
Costes directos e inmateriales	66,29 €
TOTAL	14.734,57 €

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 §3.3.3).

Con el fin de calcular el coste que supondrían los honorarios nos apoyamos en los estudios realizados por «Gamasutra», en este caso en la encuesta salarial de 2014 [14] (la última realizada hasta la fecha). Este estudio nos muestra el salario medio para varios puestos o roles del sector, pero a continuación se muestran los roles de nuestro 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) [14]

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

Eligiendo esta encuesta se nos plantean dos problemas principalmente:

1. **Diferencia de nivel de vida:** Debemos tener presente que el nivel de vida estadounidense y el español están lejos de ser parejos, por lo que no podemos compararlos directamente como si se tratase de una encuesta salarial de un país de nuestro entorno en ámbitos económicos.
2. **Diferencia de moneda:** La encuesta salarial nos ofrece los datos en dólares, mientras que necesitamos los datos en euros.

Nótese además que nomenclatura de los roles de nuestro proyecto se adaptan a la perfección a las usadas por «Gamasutra», salvo en el caso del «Jefe de proyecto» que se ha decidido relacionarlo con «Negocios y gestión». Para justificar esta relación tenemos que volver a la encuesta salarial y buscar la definición que hace «Gamasutra» de esta 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 [14].

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 §3.3.3), 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. [13] 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 [21]

Como observamos en la tabla, a San Francisco se le asigna un índice de **101,94** mientras que a Sevilla de **56,49** [21], 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 [16]:

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 [16]

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 último valor del registro.

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:

Salarios medios brutos (adaptados) según la encuesta salarial de «Gamasutra»		
Categoría	Categoría «Gamasutra»	Salario medio anual
Jefe de proyecto	«Business and management» <i>(Negocios y gestión)</i>	50.278,1 €
Diseñador	«Game designers» <i>(Diseñadores de juego)</i>	36.562,7 €
Programador	«Programmers and engineers» <i>(Programadores e ingenieros)</i>	46.159,2 €

Cuadro 5.4: Salarios medios brutos (adaptados) según la encuesta salarial de «Gamasutra» (2014) [14]

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:

$$Salario_{Hora} = \frac{Salario_{Anual}}{JornadaLaboralAnual_{Horas}}$$

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	27,93 €
Diseñador	20,31 €
Programador	25,64 €

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	27,93 €	2.625,42 €
Diseñador	62 horas	20,31 €	1.259,22 €
Programador	154 horas	25,64 €	3.948,56 €
TOTAL			7.833,2 €

Cuadro 5.6: Tabla costes sueldos brutos

Por tanto, el coste del proyecto en cuanto a sueldos brutos asciende a **7.833,2 €**.

Además de esto, debemos calcular el coste que supone cada trabajador en cuanto a **costes sociales**. Estos gastos se sitúan alrededor de un **30%**:

$$\text{Costes Sociales} = \text{Total Sueldos Brutos} * 0,30$$

Aplicándolo, obtenemos que el coste social asciende a **2.349,96 €**.

En este punto ya tenemos todo lo necesario para calcular los costes de personal, tan sólo tenemos que sumar los costes de sueldo bruto con los sociales, por lo que concluimos:

Costes de personal	
Sueldo bruto	7.833,2 €
Costes Seg. Social	2.349,96 €
Costes de personal (TOTAL)	10.183,16 €

Cuadro 5.7: Tabla de costes de personal

Con esto finalizamos la sección de costes de personal, quedando fijado el presupuesto para ellos en **10.183,16 €**.

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.

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 5.8: 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 5.9: 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 5.10: 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 5.11: 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:

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

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

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

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 €.**

5.5 COSTES DIRECTOS E INMATERIALES

Esta sección recoge la parte de los costes referente principalmente a los assets, elementos de diferente índole y que se han incorporado en el juego. Todos estos elementos tienen algo en común: el proyecto sólo se ha aprovechado de ellos de forma meramente estética.

Costes directos e inmateriales		
Referencia	Nombre asset	Coste
Modelo y texturas del personaje	GR Male Hero 01	22,24 €*
Modelo y texturas torreta	Turret Kit	27,80 €
Modelo y texturas enemigo «vigilante»	Modular Characters Blueprint	7,56 €
Rayo disparo «vigilante»	Beams for FPS	8,69 €
Animaciones personaje (I)	Animation Starter Pack	Gratis
Animaciones personaje (II)	Mixamo Animation Pack	Gratis
Efectos de partículas variados	Infinity Blade: Effects	Gratis
Assets variados para escenarios (I)	Epic Zen Garden	Gratis
Assets variados para escenarios (II)	Infinity Blade: Grass Lands	Gratis
Assets variados para escenarios (III)	Open World Demo Collection	Gratis
Follaje de escenarios	Advanced Cinematic Grass Blueprint	Gratis
TOTAL		66,29 €

Cuadro 5.12: Tabla de costes directos e inmateriales

Además de estos, se han usado otros assets (modelados, música, efectos de sonido) gratuitos y de libre uso. Se puede encontrar más información acerca de esto en el documento de diseño (Ver anexo §B).

5.6 CONCLUSIONES

Lo primero que tenemos que remarcar es que, si bien es cierto que el presupuesto es totalmente válido para la fase de desarrollo, cuando se obtengan ganancias aparecerán los costes de licencia que han quedado marcados con un asterisco en la sección «Costes indirectos». Como se explica esto es debido a la política de royalties de Epic Games, que se quedarán con un porcentaje del beneficio obtenido a partir de una determinada cantidad de beneficios, como se explica en dicha sección.

También hay que insistir en que, como se ha comentado previamente, no hay estudios sobre los salarios en el sector del videojuego en nuestro entorno, y es por ello que hemos tenido que recurrir a adaptar los de otros países, intentando tener en cuenta la mayor parte de las variables.

Remarcados estos destalles, más importante aún es realizar un estudio del sobre-coste, es por ello que emplearemos una subsección específica para esta labor.

5.6.1 Desviación de costes

El éxito de un presupuesto está muy subordinado a la correcta planificación que se haya hecho del proyecto. Es por esto que, a continuación, en esta sección vamos a comparar los resultados presupuestados con los reales.

Costes de personal

Para ello, vamos a recuperar la «tabla de desviación de tiempo por rol» del capítulo planificación por mera comodidad:

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 5.13: Tabla de desviación de tiempo por rol

Lo que nos interesa aquí son las horas extra que ha desempeñado cada rol (columna «Desviación final»).

Con estos datos, vamos a proceder a calcular la cuantía por rol y, de paso, la total:

Costes sueldos brutos			
Rol	Total de horas (extra)	Sueldo bruto por hora	Sueldo bruto (extra)
Jefe de proyecto	13 horas	27,93 €	363,09 €
Diseñador	11,2 horas	20,31 €	227,47 €
Programador	7,3 horas	25,64 €	187,17 €
TOTAL			777,73 €

Cuadro 5.14: Tabla costes sueldos brutos (tiempo extra)

El sobrecoste de personal en cuanto a sueldos brutos asciende a **777,73 €**, ahora, al igual que hicimos en la fase de planificación, vamos a calcular el coste social.

Para ello, calculamos el 30% de este sueldo bruto, lo que sería un total de **233,32 €**.

Sumando estas dos cantidades, podemos concluir que el sobrecoste de personal asciende a **1.011,05 €**.

Costes materiales

Los costes materiales presupuestados para el proyecto ascienden a **59,63 €** al mes.

Como nos hemos desviado dos semanas, podemos asumir que el sobrecoste en cuanto a materiales es, aproximadamente, la mitad de esa cantidad. Por tanto, el sobrecoste de material asciende a **29,81 €**.

Costes indirectos

En cuanto a costes indirectos, como se vio en el presupuesto suponían **687,89 €** al mes.

Siguiendo el mismo procedimiento que con los costes materiales, y recordando que nos hemos desviado dos semanas, el sobrecoste de costes indirectos será la mitad. Por tanto, suponen **343,95 €** más de los planificados.

Costes directos e inmateriales

Respecto a los costes directos e inmateriales no hace falta hacer ningún cálculo: el sobrecoste aquí es de **0 euros** ya sólo supusieron un desembolso único.

Sobrecoste total

Realizados los cálculos en las anteriores subsecciones ya sólo tenemos que sumar todos ellos para obtener el sobrecoste total. Por tanto, podemos concluir que el sobrecoste del proyecto, respecto al planificado, asciende a **1.384,81 €**.

Teniendo en cuenta que el valor presupuestado fue **14.734,57 €**, el sobrecoste supone un 10,64 % del total, ascendiendo el coste real del proyecto a **16.119,38 €**.

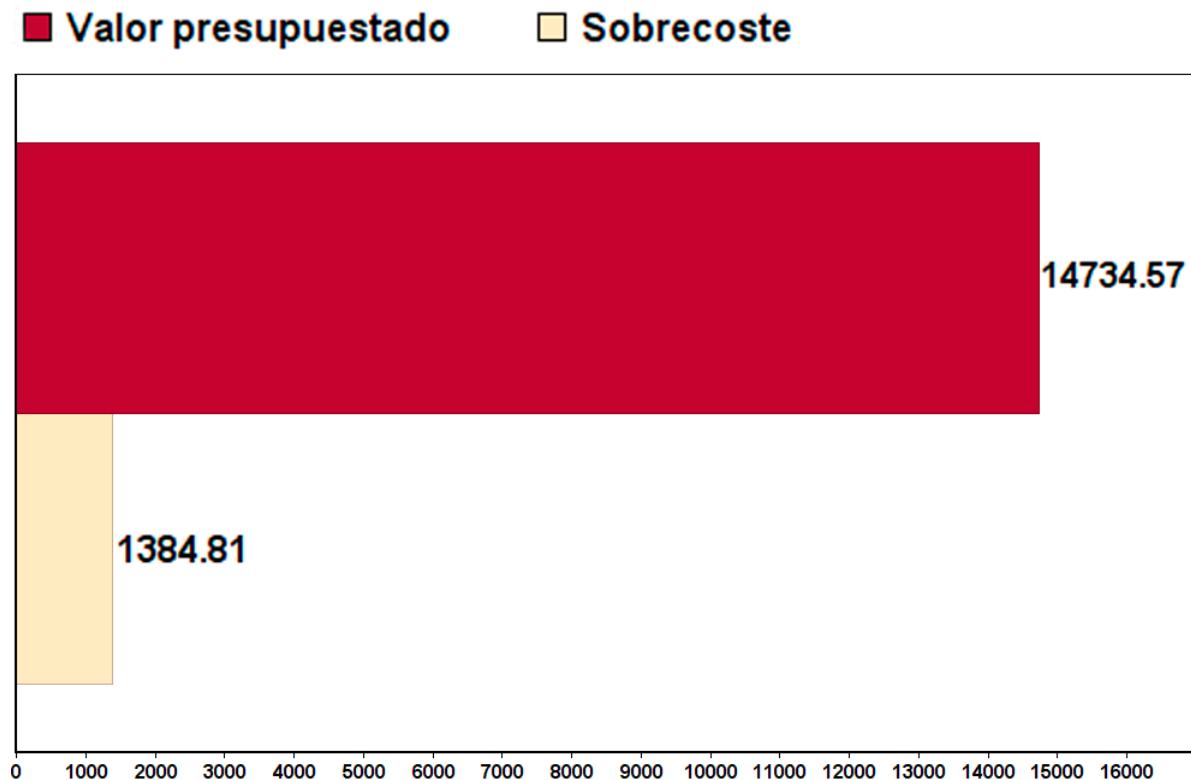


Figura 5.2: Diferencia entre el valor presupuestado y el real

PARTE III

DESARROLLO DEL PROYECTO

ARRANQUE

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

6.1 LISTA DE CARACTERÍSTICAS

La lista de características está confeccionada 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.

1. Mecánica: Movimiento del personaje.

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

2. Mecánica: Rotación del personaje.

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

3. Mecánica: Sprint.

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

4. Mecánica: Salto.

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

5. Mecánica: Agachado.

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

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

- Implementación del código de la mecánica.

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

7. Mecánica: Parpadeo (o «Blink»).

- Implementación del código de la mecánica.
- Asignación de teclas y botones vinculados la mecánica.
- Desarrollo de la animación del personaje relacionada con la mecánica.
- Gestión de adrenalina relacionada con la mecánica.
- Implementación del código del indicador de blink disponible.

8. .

9. .

10. .

11. .

12. .

13. .

14. .

15. .

16. .

17. .

18. .

19. .

20. .

21. .

22. .

23. .

24. .

25. .

26. .

27. .

28. .

29. .

30. .

6.2 DISEÑO ARQUITECTÓNICO

En el momento que hablamos de diseño arquitectónico en el marco de este proyecto estamos refiriéndonos indirectamente al diseño arquitectónico del motor gráfico que estamos utilizando, «Unreal Engine 4».

6.2.1 Módulos de juego

El motor gráfico Unreal Engine 4 está compuesto por una colección de módulos y, de la misma forma, cada juego desarrollado en él está compuesto por uno o más módulos, denominados módulos de juego, que en realidad son **DLLs** [11].

De esta manera, la arquitectura básica de un juego desarrollado con Unreal Engine 4 es la siguiente:

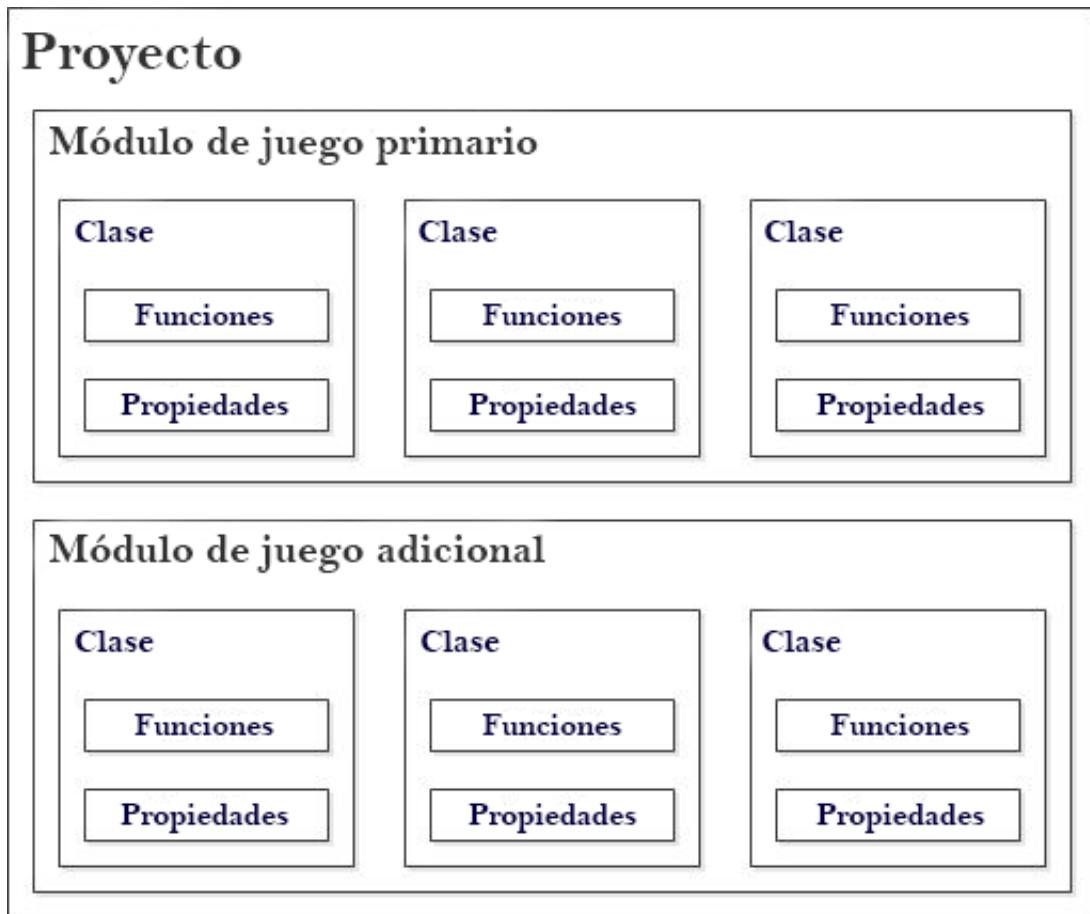


Figura 6.1: Arquitectura Unreal Engine 4 [11]

Como podemos apreciar, podemos estructurar nuestro proyecto en tantos módulos de juego como creamos necesario. Sin embargo, respecto a la creación de múltiples módulos de juego Epic Games nos hace saber:

Hay diferentes puntos de vista respecto a la separación en DLL. Separar el juego en un montón de archivos DLL puede tener más inconvenientes que beneficios, pero es una decisión que debe ser hecha por cada equipo de desarrollo basada en sus necesidades. Usar múltiples módulos de juego puede conducir en mejores tiempos de enlace e iteración de código más rápida, pero con más módulos será necesario lidiar con exportaciones de DLL e interfaces más a menudo. Esta concesión es la correcta para el motor y el editor, pero es cuestionable para el juego [11].

Como podemos apreciar, no hay una ventaja clara en usar varios módulos de juego, por lo que para la estructuración de este proyecto usaremos la opción más simple: usar un único módulo de juego, llamado «TFG» que contenga todas las clases de juego:

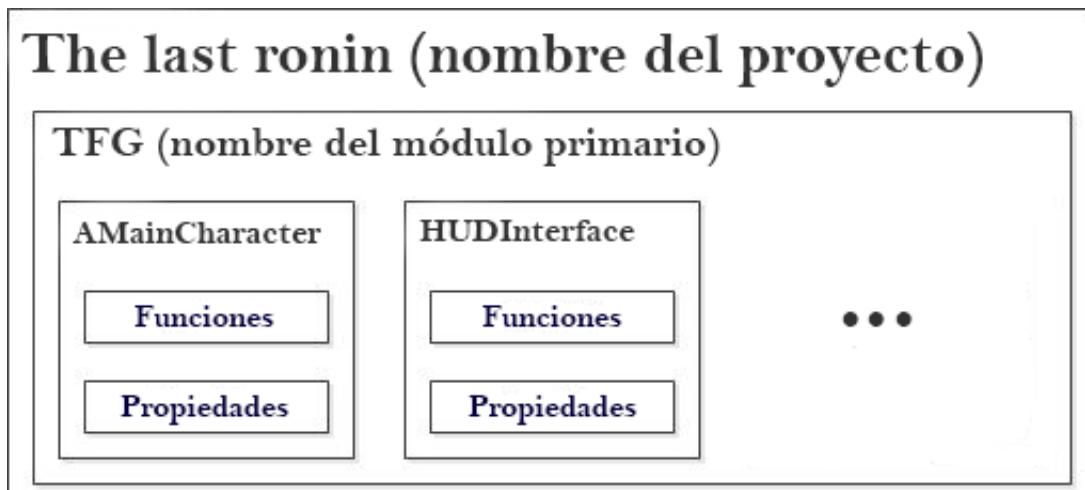


Figura 6.2: Estructura del proyecto

6.2.2 Clases de juego

Cada módulo de juego contiene indeterminadas clases C++ y estas están compuestas 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 [11].

Además, cabe destacar que en Unreal Engine se usa un sistema de prefijos, que se seguirá en la medida de lo posible en el desarrollo de este proyecto. Los prefijos pueden ser:

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».

Cuadro 6.1: *Prefijos de clases de juego en Unreal Engine* [11]

ITERACIÓN CERO

Durante este capítulo se documentará la inicialización del proyecto y se recogerán los métodos que se utilizarán durante el resto de iteraciones para realizar tareas comunes a todas ellas.

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 «4.15.1».

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 configuraremos el proyecto de la siguiente manera:

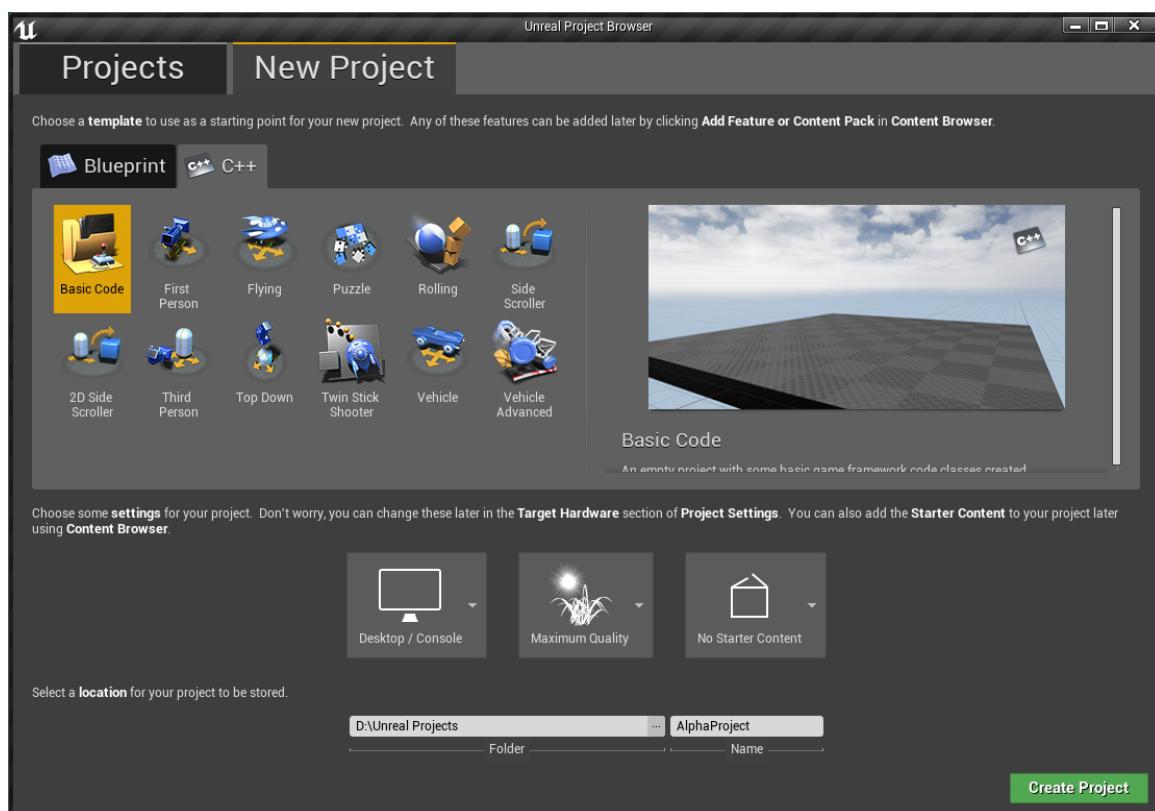


Figura 7.1: Ventana del creador de proyectos de Unreal Engine

- «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 lista la base sobre la que empezar a trabajar.

7.2 DIRECTRICES GENERALES

Durante la ejecución del proyecto se seguirán una serie de pautas, métodos y en definitiva trabajos durante las iteraciones que son recurrentes y se realizan de forma paralela a estas iteraciones. Es por ello que esta sección pretende explicarlos y que ese trabajo, difícil de documentar, esté documentado aquí, aliviando la documentación de las posteriores iteraciones.

7.2.1 Definición de controles

Respecto a cómo definir los controles de nuestro juego tenemos, principalmente, dos opciones:

- Asignar las mecánicas directamente cada una de las teclas, botones, etc. que realizarán la misma.
- Asignar las mecánicas a un «binding» (enlace), nombrado igual que la mecánica y definido dentro de las propiedades del proyecto, y que contendrá información de todos los, teclas, botones, etc. que se podrán pulsar para realizar la acción.

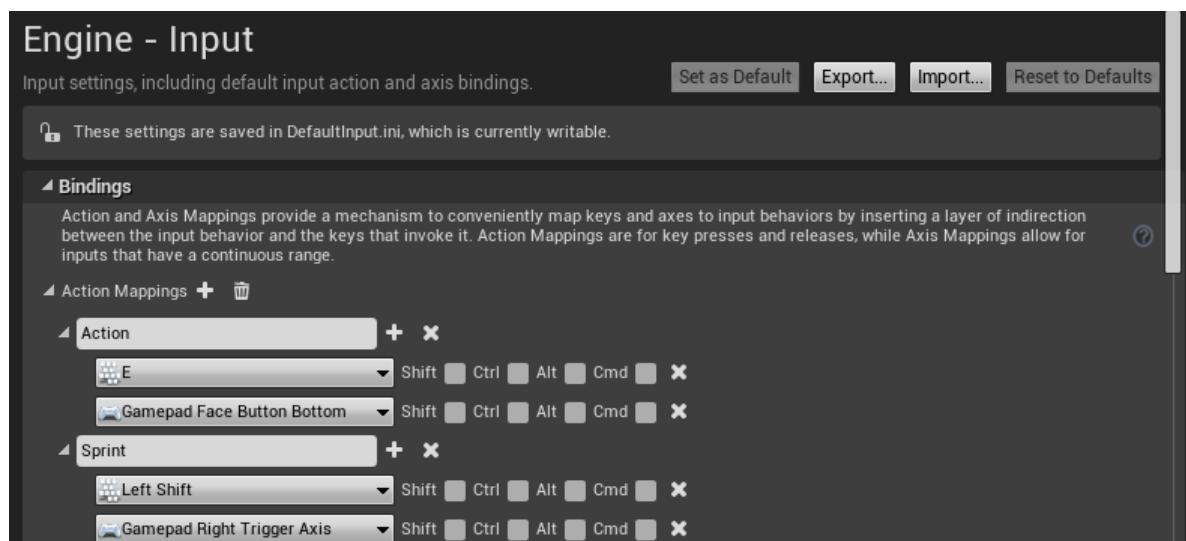


Figura 7.2: Captura de la ventana del editor de «bindings» o enlaces

Para el desarrollo de este producto vamos a elegir la segunda opción. La razón de esto es simple: esto nos permitirá cambiar los controles en un futuro o añadir nuevas teclas, botones, joysticks, gatillos, etc. que hagan estas funciones con la comodidad que ello supone.

Para acceder al menú de definición de enlaces tenemos que dirigirnos, dentro del editor de Unreal Engine, a «**Edit / Project Settings**» y hacer click, en la parte izquierda de la ventana que se nos mostrará, en «**Input**», dentro de la categoría «**Engine**».

7.2.2 Adaptación de animaciones no compatibles

En concreto, se ha decidido la incorporación de animaciones «Mixamo» a este proyecto. «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.

Retargeting

Sin embargo, hay un gran problema derivado del uso de estas animaciones: el personaje que se usa en este proyecto utiliza el esqueleto estándar de Unreal Engine, y hace ya muchas versiones del motor gráfico que «Mixamo» retiró el soporte para personajes que usen el esqueleto «por defecto» de Unreal Engine.

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 incorporados al proyecto el esqueleto destino y el esqueleto auxiliar es el siguiente [10]:

1. Primero accedemos al asset del esqueleto al que queremos importar los cambios dentro del editor de Unreal Engine, hacemos click en «Retarget manager» en la parte superior de la pantalla y en la opción «Select Rig» seleccionamos «Select Humanoid Rig». Estos son todos los cambios que tenemos que hacer en nuestro esqueleto principal.

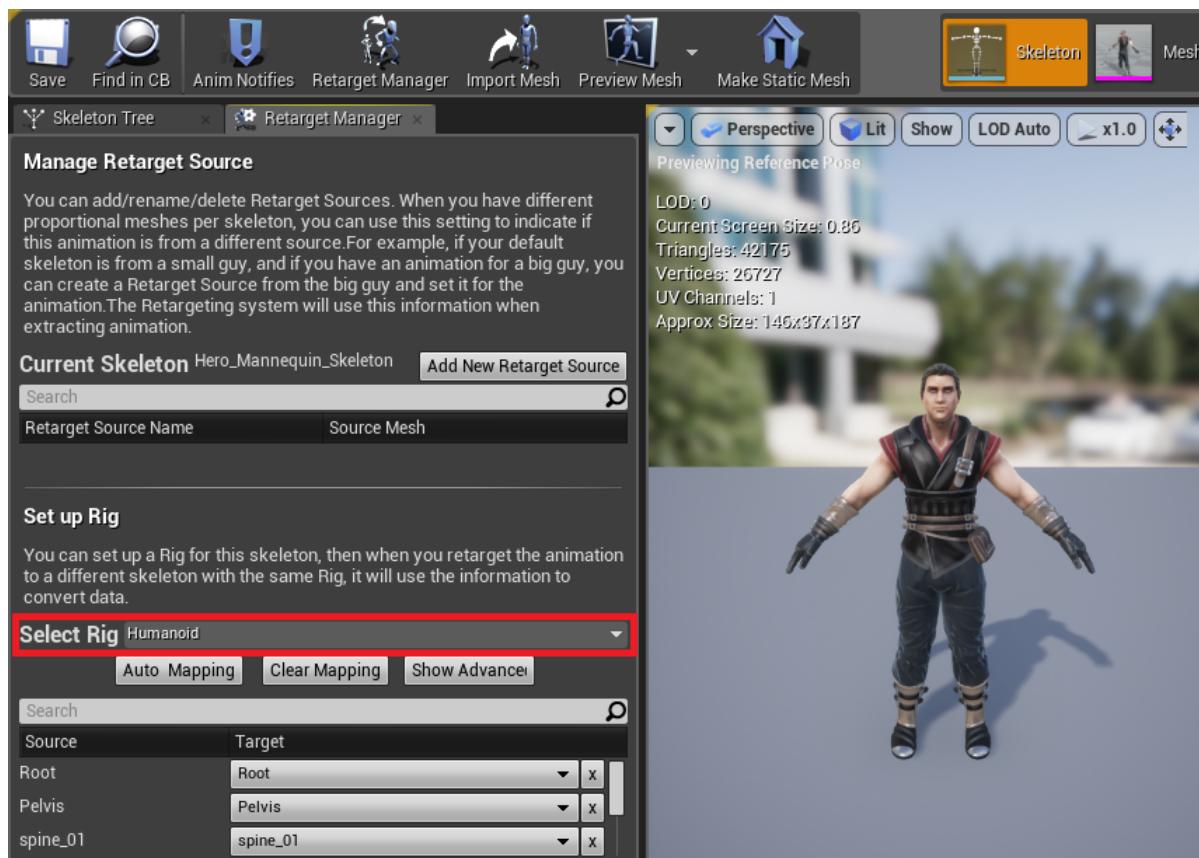


Figura 7.3: Captura del esqueleto destino adaptado

2. Realizamos las mismas acciones que en el primer paso pero esta vez en nuestro esqueleto auxiliar.
3. El tercer paso es el más tedioso pero también el más importante: a cada uno de los nodos de este esqueleto auxiliar debemos asignarle un hueso, para que al hacer el proceso de retargeting el sistema sepa qué huesos debe intercambiar. Lo hacemos según las indicaciones de la wiki de Unreal Engine (Ver [10], pasos 8 a 10). Con esto tendremos preparados los esqueletos.
4. Importamos la animación que queremos adaptar de un esqueleto a otro, y la importamos respecto al esqueleto auxiliar (ya que es una animación que, todavía, usa los huesos del mismo).
5. Hacemos click derecho en la animación que acabamos de importar y clickamos en la opción «Retarget Anim Assets» y seguidamente pulsamos «Duplicate Anim Assets and Retarget».



Figura 7.4: Captura del esqueleto origen adaptado y con las opciones de retargeting fijadas

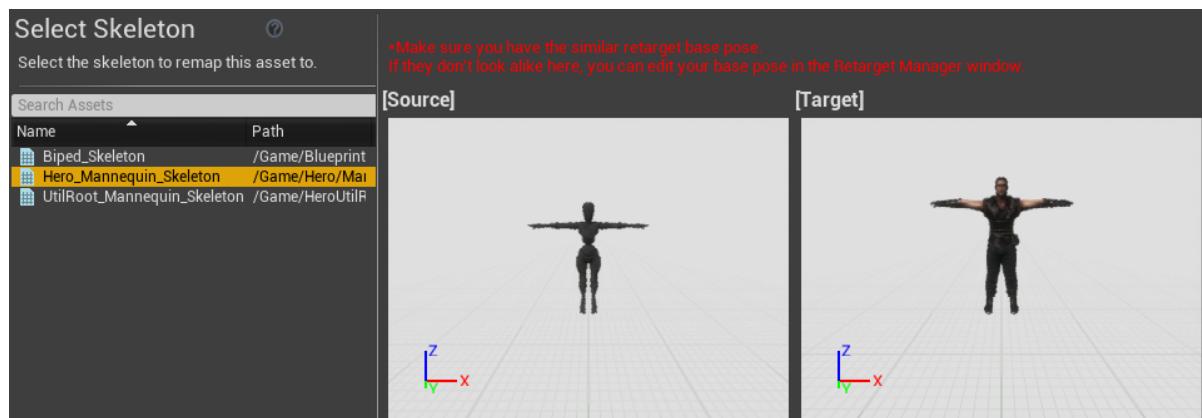


Figura 7.5: Captura del editor de animaciones de Unreal Engine, mostrando el proceso de re-asignación

Con esto completaríamos el proceso de adaptar la animación de un esqueleto a otro, nótese que tendremos que repetir todos los pasos 4 y 5 para cada una de las animaciones que se quieren incorporar al proyecto.

7.2.3 Procesado de animaciones carentes de nodo raíz

Igualmente, si utilizamos aplicaciones «Mixamo» y necesitamos habilitar la opción «Enable root motion», opción que nos permite que la capsula de colisión se desplace junto con la animación (impidiendo que el personaje traspase objetos), nos encontraremos otro problema, puesto que las animaciones provenientes de esta fuente no poseen hueso «root» (o «raíz»).

Para solucionar este problema tenemos que recurrir a programas externos a Unreal Engine, en concreto se ha elegido utilizar «Autodesk Maya», y la idea es simple: editar el esqueleto auxiliar añadiéndole el hueso que le falta.

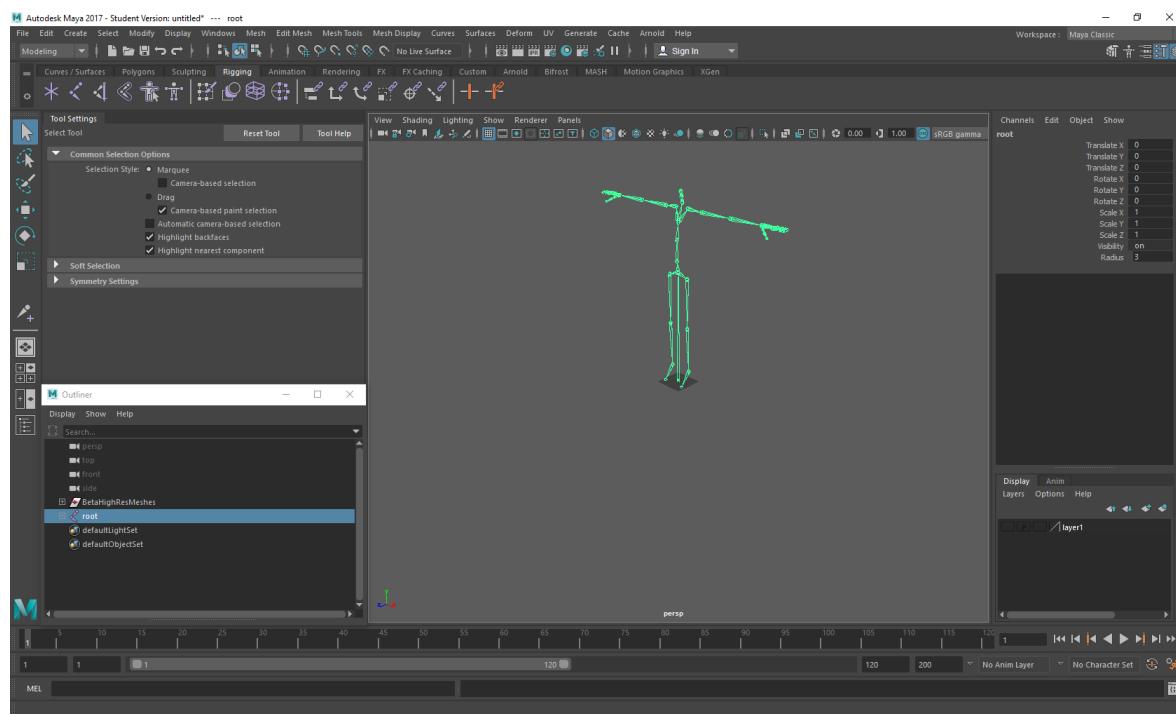


Figura 7.6: Captura de la aplicación «Autodesk Maya», mostrando el hueso «root» ya añadido a un esqueleto

El proceso, esta vez, es el siguiente:

1. Importamos nuestro esqueleto auxiliar.
2. Seleccionamos todos los componentes del esqueleto.
3. Con dichos componentes seleccionados, añadimos un nuevo hueso con la opción «Create joints» y hacemos click en la base del personaje (la posición en la que se debe encontrar el hueso raíz, justo debajo de la pelvis). No necesitamos colocar el hueso a la perfección puesto que eso lo haremos más fácilmente en los pasos siguientes.
4. Cambiamos el nombre del nodo / hueso que acabamos de crear de «joint1» a «root» (en minúsculas), el campo que tenemos que editar está, por defecto, en el panel derecho del programa.
5. Ajustamos la opción «Translate X», «Translate Y» y «Translate Z» (podemos hacer esto en el mismo panel del paso anterior, justo debajo), a «0» para que, ahora sí, el hueso quede perfectamente colocado en el centro.
6. A continuación seleccionamos el hueso de la cadera de la animación, mantenemos pulsado «shift» y seleccionamos el nodo raíz que acabamos de crear y, finalmente, seleccionamos la opción «Connect joint» para que todos los huesos del esqueleto dependan del nodo raíz.
7. Por último, exportamos el esqueleto editado (en formato «.fbx» para poder importarlo a Unreal Engine como esqueleto auxiliar).

Una vez completados estos pasos, cualquier animación que se exporte con el esqueleto modificado generado como base tendrá nodo raíz, solucionando el problema.

7.2.4 Sistema de animación del personaje

Para la animación del personaje se utilizará un blueprint de animación, totalmente independiente de la clase de nuestro personaje, que (entre otras cosas) implementará una máquina de estados y que se comunicará con la clase del mismo a través de una interfaz. Estos blueprints de animación se seleccionan dentro de la clase de cada personaje, delegando totalmente la responsabilidad de realizar las animaciones en ellos.

El funcionamiento interno de esta clase es muy parecido al de una máquina de estados convencional: se dispone de un punto de entrada («Entry») que está conectado con la animación inicial (típicamente una animación del personaje de pie y parado) y que a partir de ahí, en base a una serie de parámetros, pasará de un nodo a otro, es decir: de una animación a otra.

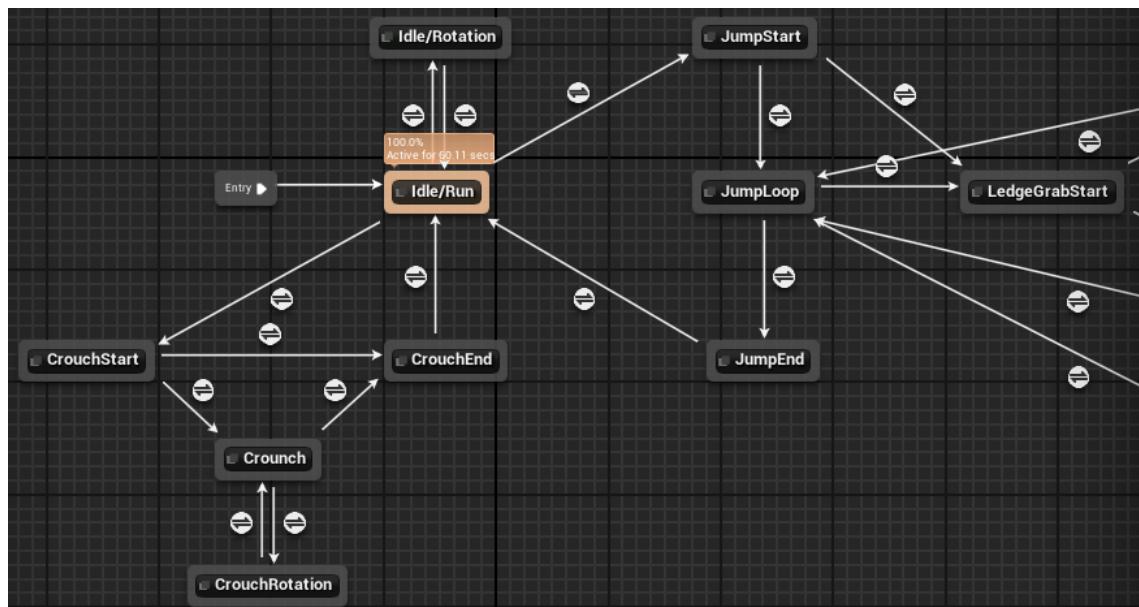


Figura 7.7: Vista de una pequeña parte de la máquina de estados implementada para animar al personaje del proyecto

Además de esta máquina de estados el blueprint se encarga de recoger información tanto de la interfaz de animación (a la que le pasa información la clase del personaje) como del componente de movimiento del personaje para saber si el personaje está quieto o no, en la dirección que se está moviendo, si está agachado, si está agarrado a algún vértice, etc.

Notificaciones en animaciones

Las notificaciones de animaciones son una funcionalidad de Unreal Engine que utilizaremos para varias tareas a lo largo del desarrollo del proyecto. Principalmente para controlar cuándo acaba un «animation montage» (descritos más adelante en esta sección) pero también se usarán para que el sonido asociado con las animaciones (por ejemplo, el sonido de un paso, generado cuando el personaje va corriendo y pisa el suelo) encaje perfectamente con las mismas. Concretamente esto último se realiza en dos pasos:

1. En la animación se emplazará una notificación de sonido cada vez que se deba producir el mismo.
2. En el blueprint de animación de personaje se implementará que cada vez que se reciba esa notificación se reproduzca el sonido pertinente (según la notificación que recibamos) en la localización del personaje.

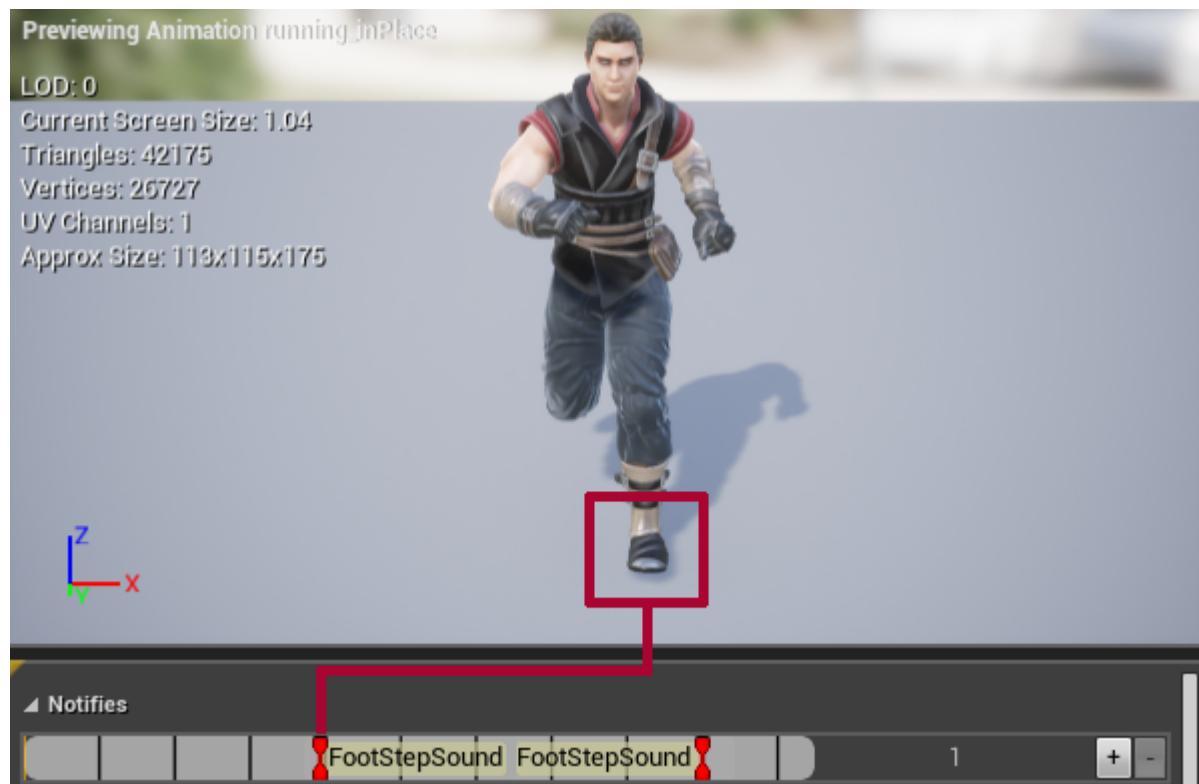


Figura 7.8: Captura de una animación mostrando las notificaciones de sonido

Blend spaces

Un «blend space» es un tipo especial de blueprint que se utiliza para pasar de una animación a otra, de forma suave, en función de unos parámetros.

Estos «blend spaces» pueden ser de una o dos dimensiones (es decir, dependientes de uno o dos parámetros) y también pueden ser nodos de la máquina de estados del blueprint de animación, por lo que es una forma también de encapsular animaciones.

Dentro del proyecto se usan en repetidas ocasiones, por ejemplo para cambiar de una animación a otra en función de la dirección y velocidad del personaje.



Figura 7.9: Imagen de ejemplo de «blend space» en 2D: en función de la dirección y la velocidad

En la imagen que se muestra se puede apreciar precisamente el caso citado, cambiar de una animación a otra en función de la dirección y velocidad: los ejes horizontal y vertical representan la dirección (0 sería movimiento hacia el frente, -90 movimiento hacia la izquierda, 90 movimiento hacia la derecha y 180 o -180 movimiento hacia atrás) y la velocidad (0, el mínimo, representaría parado y 800, el máximo, corriendo), respectivamente, y cada punto del mapeado que aparece en la imagen representa una animación distinta (salvo en las columnas de -180 y 180 que estarían duplicadas a un lado y a otro del mapeado, por lo que se comentaba anteriormente).

Cabe destacar que el «blend space» creará «animaciones intermedias» si, según sus parámetros de entrada, no encuentra una animación que se ajuste perfectamente. Siguiendo con el ejemplo expuesto anteriormente, si el personaje se dirige hacia delante a una velocidad de 400 y teniendo en cuenta que para ese caso tenemos 3 animaciones: una en velocidad 0, representando al personaje parado, otra en velocidad 250, representando al personaje andando hacia delante, y otra en velocidad 800, representando al personaje corriendo hacia delante, entonces la animación resultado será una mezcla (proporcional al parámetro dado) entre la animación de andar y la de correr.

Animation montages

Un «animation montage» es un tipo especial de blueprint de animación, que depende siempre de una animación en concreto, y que puede ser usado con múltiples propósitos.

En el proyecto que nos ocupa son usados, principalmente, para reproducir animaciones una sola vez (es decir, sin bucles) y de forma independiente al estado en el que se encuentra el personaje dentro de la máquina de estados del blueprint de animación. En otras palabras, cuando se ejecuta el «animation montage» este invalida la animación que debiera reproducirse según la máquina de estados y cuando acaba vuelve a autorizar la animación pertinente de la máquina de estados.



Figura 7.10: Ejemplo de «animation montage», usado cuando el protagonista escala un saliente

PRIMERA ITERACIÓN

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

8.1 INTRODUCCIÓN

Nuestro objetivo principal en esta iteración es familiarizarnos con el entorno, con la programación en C++ y dejar lista la base sobre la que trabajaremos en el resto del proyecto. No obstante, también nos centraremos en dar vida al personaje, dotándolo de movimientos básicos, y la realización de algunas mecánicas más avanzadas como la de teletransporte.

Nuestro personaje heredará de la clase «ACharacter» y se compondrá principalmente de una **capsula de colisión**, que será la encargada de gestionar las colisiones con el entorno (reduciendo drásticamente la complejidad computacional) y de un **modelo de personaje**, que irá dentro de la citada cápsula. Como se explica en la iteración cero, el modelo de personaje estará animado por un blueprint de animación, que también se empezará a crear en esta iteración.

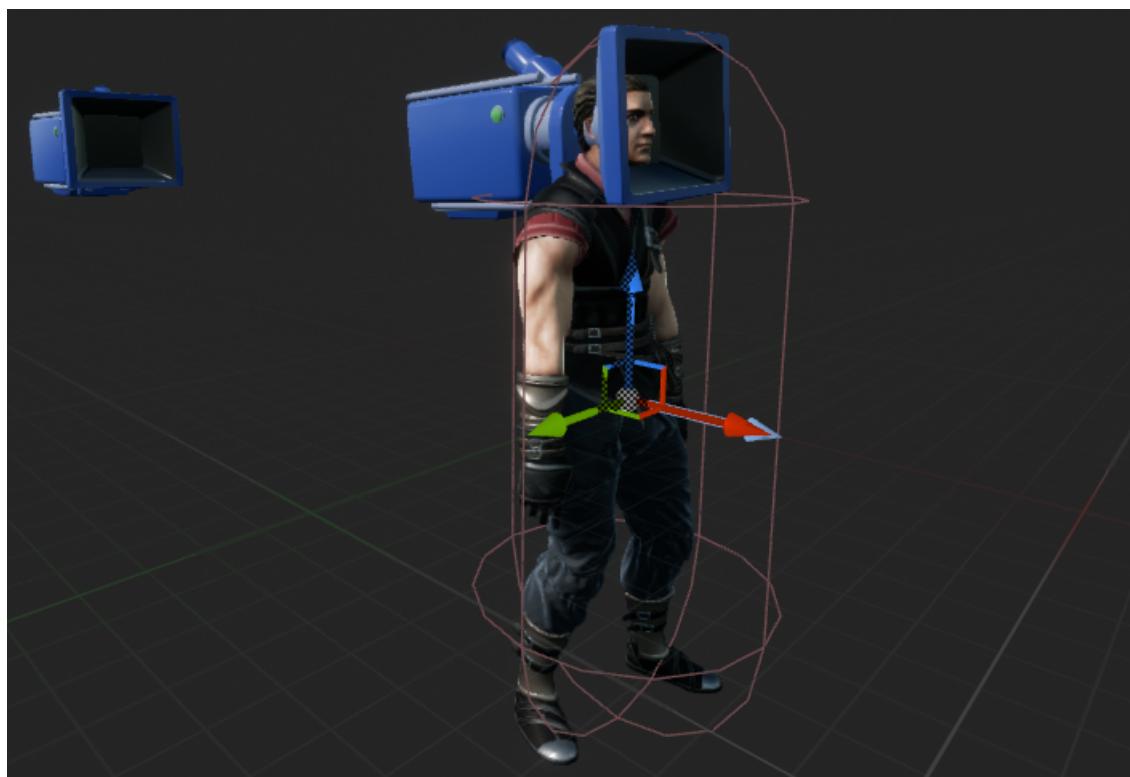


Figura 8.1: Fotograma de los componentes básicos del personaje: capsula de colisión, modelo, cámara en primera persona real y cámara en tercera persona

Como se especifica en el documento de diseño, podemos alternar entre las dos **cámaras** disponibles: una anclada al la cabeza del modelo del personaje (primera persona real o «true first person») y otra que se situaría detrás y levemente a la derecha del personaje (cámara en tercera persona).

La cámara de primera persona estará anclada a un brazo de cámara que se contraerá al detectar una colisión con el entorno, para evitar traspasar la pared y ver el contenido de las siguientes salas.

La cámara en tercera persona, por el contrario, estará unida a dos brazos de cámara:

1. El primero saldrá de detrás de la cabeza del personaje y se alejará del personaje por su espalda.
2. El segundo estará atado al primero y girará 90 grados a la derecha del mismo. A este segundo brazo de cámara se atará a su vez la cámara en tercera persona.

Con esto conseguimos que cuando la cámara en tercera persona colisione con un obstáculo retroceda tanto hacia la izquierda como hacia delante (dependiendo de dónde se encuentre el obstáculo retrocederá más o menos en una dirección u otra).

Otro aspecto a tratar es el del **movimiento** del personaje (gravedad, control en el aire, etc.): la clase de nuestro personaje principal también hereda de «ACharacter» un componente de movimiento. El objetivo aquí es conseguir adaptar el movimiento del personaje para que sea cómodo de manejar en situaciones límite, primando este control sobre el realismo.

Respecto al **sistema de teletransporte**: la idea general es que el usuario pueda teletransportar al personaje mirando hacia un obstáculo y pulsando el botón adecuado, siempre que se encuentre dentro de un rango. En posteriores iteraciones se añadirá un indicador para saber, en cada momento, si se puede llevar a cabo este teletransporte.

Por último, para conseguir un mayor realismo vamos a aplicar un pequeño truco para que el personaje rote cuando estemos parados, estemos usando ratón o mando para rotar. Este truco consiste en crear un nuevo enlace (en este caso llamado «RotationTrick») que se active cada vez que reciba un input de rotación (tanto de ratón como de mando) y a partir de esto se comunique con el blueprint de animación para hacer rotar al personaje. Esto se debe hacer así por la naturaleza de Unreal, ya que si intentamos mandar la señal de rotación a la animación individualmente ambos inputs entrarán en conflicto, adulterando el valor enviado al blueprint de la animación.

8.2 CARACTERÍSTICAS A DESARROLLAR

1. Creación del personaje.
2. Creación de una cámara «True first person camera».
3. Creación de una cámara en tercera persona.
4. Mecánica: Movimiento del personaje.
 - Implementación del código de la mecánica.
 - Asignación de teclas y botones vinculados la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
5. Mecánica: Rotación del personaje.
 - Implementación del código de la mecánica.
 - Asignación de teclas y botones vinculados la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
6. Mecánica: Rotación del personaje vinculada a un rate.
 - Implementación del código de la mecánica.
 - Asignación de teclas y botones vinculados la mecánica.
7. Mecánica: Sprint.
 - Implementación del código de la mecánica.
 - Asignación de teclas y botones vinculados la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
8. Mecánica: Salto.
 - Implementación del código de la mecánica.
 - Asignación de teclas y botones vinculados la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
9. Mecánica: Agachar.
 - Implementación del código de la mecánica.

- Asignación de teclas y botones vinculados la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
10. Mecánica: Tiempo bala (o «Time dilation»).
- Implementación del código de la mecánica.
 - Asignación de teclas y botones vinculados la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
 - Gestión de adrenalina relacionada con la mecánica.
11. Mecánica: Teletransporte (o «Blink»).
- Implementación del código de la mecánica.
 - Asignación de teclas y botones vinculados la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
 - Gestión de adrenalina relacionada con la mecánica.
 - Implementación del código del indicador de blink disponible.
12. Control de límites del mapa (en otras palabras, que el usuario no pueda abandonar el mapa y si lo hace, muera).
- Implementación del código de la mecánica.
13. Control de máxima distancia de caída (en otras palabras, que el usuario al caer en una superficie situada en un foso, muera).
- Implementación del código de la mecánica.
14. Animación de rotación del personaje al girarse.
- Implementación del código.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
15. Sistema de gestión de adrenalina.
- Implementación del código.

8.3 DISEÑO

Memorando técnico 1-001: 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 añadamos, 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 1-001: Creación del personaje

Memorando técnico 1-002: 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 1-002: Creación de la cámara TFP

Memorando técnico 1-003: 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 1-003: Creación de la cámara TP

Memorando técnico 1-004: 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 1-004: Movimiento del personaje

Memorando técnico 1-005: 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 1-005: Rotación de cámara y personaje

Memorando técnico 1-006: 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 1-006: Rotación de cámara y personaje (para controladores)

Memorando técnico 1-007: 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 1-007: Sprint

Memorando técnico 1-008: 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 1-008: Salto

Memorando técnico 1-009: 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 1-009: Agachar

Memorando técnico 1-010: 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 1-010: Tiempo bala

Memorando técnico 1-011: 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 1-011: Teletransporte o «blink»

Memorando técnico 1-012: 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 1-012: Control de límites del mapa

Memorando técnico 1-013: 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 1-013: Control de máxima distancia de caída

Memorando técnico 1-014: Animación de rotación 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 1-014: Animación de rotación del personaje al girarse.

Memorando técnico 1-015: Sistema de gestión de adrenalina.

Asunto	Se desea que cuando el personaje realice alguna habilidad especial se consuma adrenalina y ésta se recarge periódicamente con el tiempo.
Resumen	Cuando el usuario desee ejecutar algún poder / habilidad especial, primero se comprobará si hay adrenalina suficiente (todos los poderes llevarán asociado un consumo de adrenalina) y, si hay suficiente, se ejecutará el poder y se restará la cantidad de adrenalina asociada a la actual. Si no hay ningún poder activo, se recargará con cada tick del juego.
Factores causantes	-
Solución	El consumo de adrenalina se delega a cada habilidad especial, siendo la función relacionada con esta la encargada de comprobar si hay adrenalina suficiente, restar la precisa, etc. y el sistema recargará automáticamente un punto de adrenalina con cada tick del juego.
Motivación	Solución propuesta por simplicidad.
Cuestiones abiertas	-
Alternativas	-

Cuadro 8.15: Memorando técnico 1-015: Sistema de gestión de adrenalina.

8.4 IMPLEMENTACIÓN

ID	Descripción de la acción de alto nivel	
1-004	Añade movimiento en la dirección del personaje o en la opuesta	
Métodos de alto nivel		
[void] MoveFoward (Value:float)		
Pasos		
<p>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.</p> <p>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.</p> <p>1.2. Si es igual a 0, no se hace nada.</p>		
Métodos de bajo nivel necesarios		
Paso	Clase	Método
1.1	APawn	[void] AddMovementInput (Direction:FVector, Value:float)
ID	Descripción de la acción de alto nivel	
1-004	Añade movimiento a izquierda o derecha de la dirección del personaje	
Métodos de alto nivel		
[void] MoveRight (Value:float)		
Pasos		
<p>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.</p> <p>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.</p> <p>1.2. Si es igual a 0, no se hace nada.</p>		
Métodos de bajo nivel necesarios		
Paso	Clase	Método
1.1	APawn	[void] AddMovementInput (Direction:FVector, Value:float)

ID	Descripción de la acción de alto nivel	
1-005	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		
Paso	Clase	Método
1	APawn	[void] AddControllerYawInput (Value:float)

ID	Descripción de la acción de alto nivel	
1-005	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		
Paso	Clase	Método
1	APawn	[void] AddControllerPitchInput (Value:float)

ID	Descripción de la acción de alto nivel			
1-006	Añade rotación horizontal a la cámara para controladores			
Métodos de alto nivel				
[void] TurnAtRate (Rate:float)				
Pasos				
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.				
Métodos de bajo nivel necesarios				
Paso	Clase	Método		
1	APawn	[void] AddControllerYawInput (Value:float)		
1	UWorld	[Float] GetDeltaSeconds ()		
ID	Descripción de la acción de alto nivel			
1-006	Añade rotación vertical a la cámara para controladores			
Métodos de alto nivel				
[void] LookUpAtRate (Rate:float)				
Pasos				
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.				
Métodos de bajo nivel necesarios				
Paso	Clase	Método		
1	APawn	[void] AddControllerPitchInput (Value:float)		
1	UWorld	[Float] GetDeltaSeconds ()		

ID	Descripción de la acción de alto nivel
1-007	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
1-007	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	
1-008	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		
Paso	Clase	Método
1	ACharacter	[void] Jump ()

ID	Descripción de la acción de alto nivel			
1-008	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				
Paso	Clase	Método		
1	ACharacter	[void] StopJumping ()		
ID	Descripción de la acción de alto nivel			
1-009	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				
Paso	Clase	Método		
1	ACharacter	[void] Crouch ()		
ID	Descripción de la acción de alto nivel			
1-009	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				
Paso	Clase	Método		
1	ACharacter	[void] UnCrouch ()		

ID	Descripción de la acción de alto nivel	
1-010	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		
Paso	Clase	Método
1.1	AMainCharacter	[void] StartTimeDilation ()
1.2	AMainCharacter	[void] EndTimeDilation ()

ID	Descripción de la acción de alto nivel
1-010	El personaje entra en tiempo bala
Métodos de alto nivel	
[void] StartTimeDilation ()	
Pasos	
<p>1. Se reproduce el sonido asignado a la entrada en el tiempo bala en la localización del jugador.</p> <p>2. El tiempo del mundo se ralentiza un 50 %.</p> <p>3. El tiempo para el personaje se ralentiza un 25 %.</p> <p>4. Ajusta el valor de la variable «bTimeDilationIsActive» a verdadero.</p>	

ID	Descripción de la acción de alto nivel
1-010	El personaje entra en tiempo bala
Métodos de alto nivel	
[void] EndTimeDilation ()	
Pasos	
<p>1. Se reproduce el sonido asignado a la entrada en el tiempo bala en la localización del jugador reproducido al revés.</p> <p>2. El tiempo del mundo vuelve a su estado natural.</p> <p>3. El tiempo para el personaje vuelve a su estado natural.</p> <p>4. Ajusta el valor de la variable «bTimeDilationIsActive» a falso.</p>	

ID	Descripción de la acción de alto nivel
1-011	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
1-012	El jugador abandona el área de juego
Métodos de alto nivel	
[void] ActorEndOverlap () - Clase «TriggerBoxSafeZone»	
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>	

ID	Descripción de la acción de alto nivel
1-013	El jugador cae en un foso profundo
Métodos de alto nivel	
[void] ActorBeginOverlap () - Clase «TriggerBoxDeath»	
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>	

ID	Descripción de la acción de alto nivel
1-014	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>	

8.5 PRUEBAS

Para la realización de todas las pruebas de esta iteración se ha decidido crear un mapa llamado «**Test_BasicMechanics**» que no será accesible desde el juego (pero sí desde la versión de editor del desarrollador) cuyo objetivo no es otro que testear cada una de las implementaciones de las mecánicas desarrolladas en la iteración.



Figura 8.2: Fotograma del mapa de pruebas empleado en la iteración 1

Este mapa contiene lo necesario para probar las mecánicas que hemos implementado en esta iteración: saltos, lugares en los que tendremos que agacharnos y un área de pruebas donde podremos probar si nuestro personaje se detiene al colisionar correctamente, si las cámaras colisionan correctamente y no traspasan las paredes, si las animaciones del personaje se reproducen correctamente para cada acción, etc.

Como dato importante, con propósitos de testeo y hasta que llegue la iteración en la que implementemos la forma de ganar y perder poderes del personaje, el personaje tendrá poderes en todo momento.

En este mapa se han realizado las siguientes pruebas:

Batería de pruebas: Cámara «primera persona real» (1-002)

1. La cámara no se introduce dentro del modelo del personaje.
2. La cámara se sitúa a la altura de los ojos del personaje cuando éste está de pie.
3. La cámara se sitúa a la altura de los ojos del personaje cuando éste se agacha.
4. Cuando el personaje y, por ende, la cámara colisionan con una pared (o similar), ésta última no traspasa la misma.

Cuadro 8.16: *Batería de pruebas: Cámara «primera persona real» (1-002)*

Batería de pruebas: Cámara «tercera persona» (1-003)

1. La cámara se sitúa atrás y levemente a la derecha del personaje.
2. Si nos acercamos a una pared (o similar) de espaldas la cámara se va acercando al personaje pero no llega a adentrarse dentro del modelo del mismo.
3. Si nos acercamos a una pared (o similar) por la derecha la cámara se va centrando (respecto al personaje) hasta que se para justo en el centro.

Cuadro 8.17: *Batería de pruebas: Cámara «tercera persona» (1-003)*

Batería de pruebas: Movimiento del personaje y sprint (1-004 y 1-007)

1. Probar movimiento hacia delante del personaje (con y sin sprint). Comprobar que ambas animaciones se reproducen correctamente.
2. Probar movimiento hacia detrás del personaje (con y sin sprint). Comprobar que ambas animaciones se reproducen correctamente.
3. Probar movimiento hacia la izquierda del personaje (con y sin sprint). Comprobar que ambas animaciones se reproducen correctamente.
4. Probar movimiento hacia la derecha del personaje (con y sin sprint). Comprobar que ambas animaciones se reproducen correctamente.
5. Probar movimiento hacia delante-izquierda del personaje (con y sin sprint) y comprobar que las animaciones se mezclan correctamente en ambos casos.
6. Probar movimiento hacia delante-dererecha del personaje (con y sin sprint) y comprobar que las animaciones se mezclan correctamente en ambos casos.
7. Probar movimiento hacia detrás-izquierda del personaje (con y sin sprint) y comprobar que las animaciones se mezclan correctamente en ambos casos.
8. Probar movimiento hacia detrás-dererecha del personaje (con y sin sprint) y comprobar que las animaciones se mezclan correctamente en ambos casos.
9. Repetir para mando o teclado y ratón (según lo que se haya usado previamente).

Cuadro 8.18: Batería de pruebas: Movimiento del personaje y sprint (1-004 y 1-007)

Batería de pruebas: Rotación de cámara y personaje (1-005 y 1-014)

1. Cuando movemos la cámara con el eje X del ratón, se mueve la misma en sentido horizontal y se modifica con ella la rotación del personaje, aplicando una animación si está parado.
2. Cuando movemos la cámara con el eje Y del ratón, se mueve la misma en sentido vertical.
3. Cuando movemos la cámara usando ambos ejes, se mezclan ambos pasos.

Cuadro 8.19: Batería de pruebas: Rotación de cámara y personaje (1-005 y 1-014)

Batería de pruebas: Rotación de cámara y personaje con controlador (1-006 y 1-014)

1. Cuando movemos la cámara con el eje X de la palanca analógica derecha del mando, se mueve la misma en sentido horizontal y se modifica con ella la rotación del personaje, aplicando una animación si está parado.
2. Cuando movemos la cámara con el eje Y de la palanca analógica derecha del mando, se mueve la misma en sentido vertical.
3. Cuando movemos la cámara usando ambos ejes, se mezclan ambos pasos.

Cuadro 8.20: Batería de pruebas: Rotación de cámara y personaje con controlador (1-006 y 1-014)

Batería de pruebas: Salto (1-008)

1. Cuando pulsamos la tecla / botón «saltar» el personaje salta, reproduciendo la animación del salto.
2. Si, mientras estamos en un salto, dejamos de pulsar «saltar», el personaje dejará de ascender en su salto si aún lo está haciendo. Nota: Esto puede ser difícil de apreciar debido a la altura de salto.
3. Cuando el personaje toca el suelo, termina la animación de salto y empieza la correspondiente (parado, andando, corriendo, etc).

Cuadro 8.21: Batería de pruebas: Salto (1-008)**Batería de pruebas: Agachar (1-009 y 1-014)**

1. Si mantenemos pulsado «agachar» el personaje permanecerá agachado, reproduciendo la animación de inicio de agachado y, a continuación, la animación de agachado-parado.
2. Si soltamos «agachar» el personaje se levantará siempre que pueda hacerlo, reproduciendo la animación de fin de agachado, si no permanecerá agachado hasta que pueda.
3. Si nos movemos mientras que estamos agachado, el personaje se moverá en esa dirección y se reproducirá la animación oportuna.
4. Si rotamos mientras que estamos agachados, el personaje rotará conforme la dirección de la cámara, reproduciendo además la animación oportuna.
5. Si pulsamos y soltamos «agachar» rápidamente, el personaje sólo se agachará por un instante.

Cuadro 8.22: Batería de pruebas: Agachar (1-009 y 1-014)

Batería de pruebas: Tiempo bala (1-010)

1. Cuando pulsamos la tecla / botón «tiempo bala», si hay suficiente adrenalina y tenemos poderes, empieza el tiempo bala y la adrenalina comienza a disminuir.
2. Si pulsamos «tiempo bala» y no hay suficiente adrenalina o no tenemos poderes, no pasará nada.
3. Si pulsamos «tiempo bala» estando ya en tiempo bala, se parará dicho tiempo bala y la adrenalina comenzará a recargarse.
4. Si agotamos la adrenalina sin salir del tiempo bala, entonces automáticamente se parará el tiempo bala y la adrenalina comenzará a recargarse.

Cuadro 8.23: Batería de pruebas: Tiempo bala (1-010)

Batería de pruebas: Teletransporte o «blink» (1-011)

1. Cuando pulsamos «blink», si hay suficiente adrenalina, tenemos poderes y estamos mirando hacia un punto que esté en rango, se teletransportará nuestro personaje a dicho punto. La rotación de nuestro personaje se adaptará correctamente a su nueva posición.
2. Si pulsamos «blink» y, durante el hipotético camino que seguiría el personaje al realizar el teletransporte, hay algún obstáculo el personaje se parará en él (evitando, por ejemplo, traspasar paredes con esta técnica).
3. En caso contrario, cuando pulsamos «blink» no pasará nada.

Cuadro 8.24: Batería de pruebas: Teletransporte o «blink» (1-011)

Batería de pruebas: Control de límites del mapa (1-012)

1. Si estamos dentro de una caja de colisión de control de límites y la abandonamos, se ejecutará la instrucción de muerte del personaje (en este punto del desarrollo, aparecerá un mensaje de debug).

Cuadro 8.25: Batería de pruebas: Control de límites del mapa (1-012)**Batería de pruebas: Control de máxima distancia de caída (1-013)**

1. Si durante el juego, entramos en contacto con esta caja de colisión, se ejecutará la instrucción de muerte del personaje (en este punto del desarrollo, aparecerá un mensaje de debug).

Cuadro 8.26: Batería de pruebas: Control de máxima distancia de caída (1-013)

8.6 DESPLIEGUE

Para realizar los despliegues, antes de nada tenemos que tener en cuenta si vamos a desplegar en el propio equipo de desarrollo o en un equipo de producción o preproducción. El motivo de esto es que debemos configurar el proyecto de una forma u otra según dicha peculiaridad:

- Para desplegar en el propio equipo de desarrollo no tendremos que configurar nada, ya que vendrá preparado por defecto (modo de construcción «Development»).
- Si queremos desplegar en un entorno de **pre-producción**, debemos seleccionar la opción «Shipping» en la configuración de la build, ya que de otra forma no se preparará al software para funcionar en otros equipos.

Para cambiar esta opción tan sólo tenemos que dirigirnos, dentro del editor de Unreal Engine, a «**Edit / Project Settings**» hacer click, en la parte izquierda de la ventana que se nos mostrará, en «**Packaging**», dentro de la categoría «**Project**» y seleccionar la opción que se adapte a nuestras necesidades.

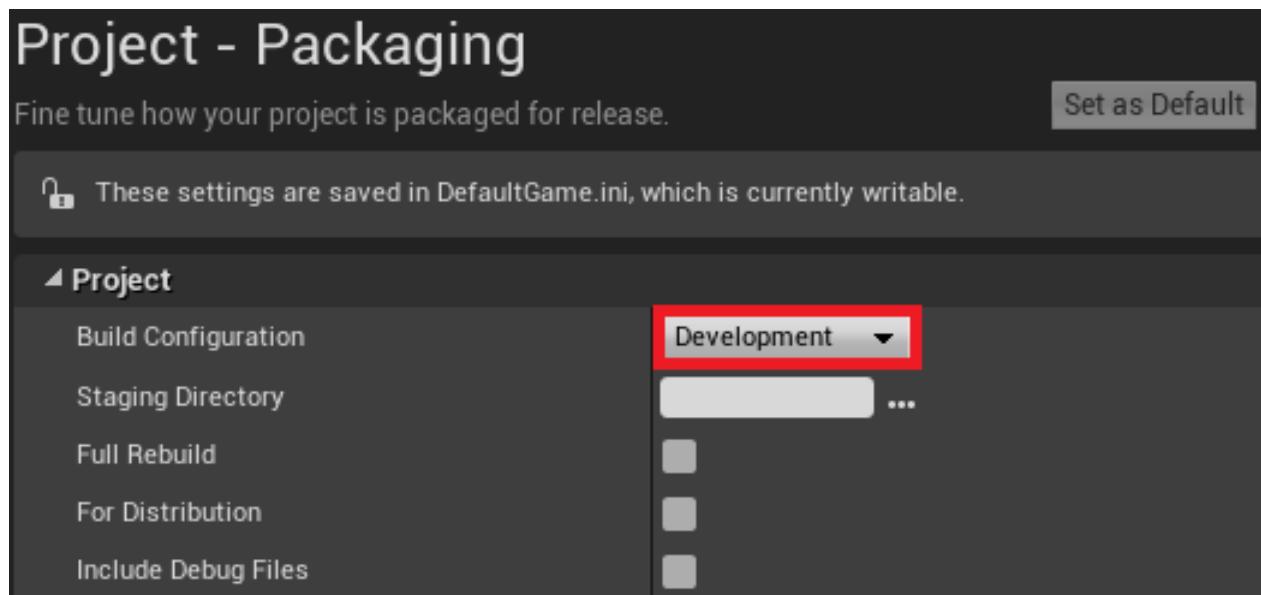


Figura 8.3: Configuración de empaquetamiento. Marcada en rojo la opción por defecto («Development»)

Un punto importante a remarcar es que si falla la exportación, Unreal Engine rara vez nos dará pistas de qué ha ido mal, así que es enormemente aconsejable realizar despliegues constantes (algo que tenemos solucionado siguiendo la metodología que estamos utilizando). Como nota, la mayoría de veces estos errores se deben a que hay algún fallo en algún blueprint (o blueprint de nivel), aunque ni esté siendo usado en el proyecto.

Dicho esto, en realidad no se realizan un despliegue sino dos: uno para la versión de 32 bits del producto y otro para la de 64 bits, pero el procedimiento es el mismo, y es tan simple como hacer click, dentro del editor de Unreal Engine, en «**File / Package project / Windows / [Versión]**».

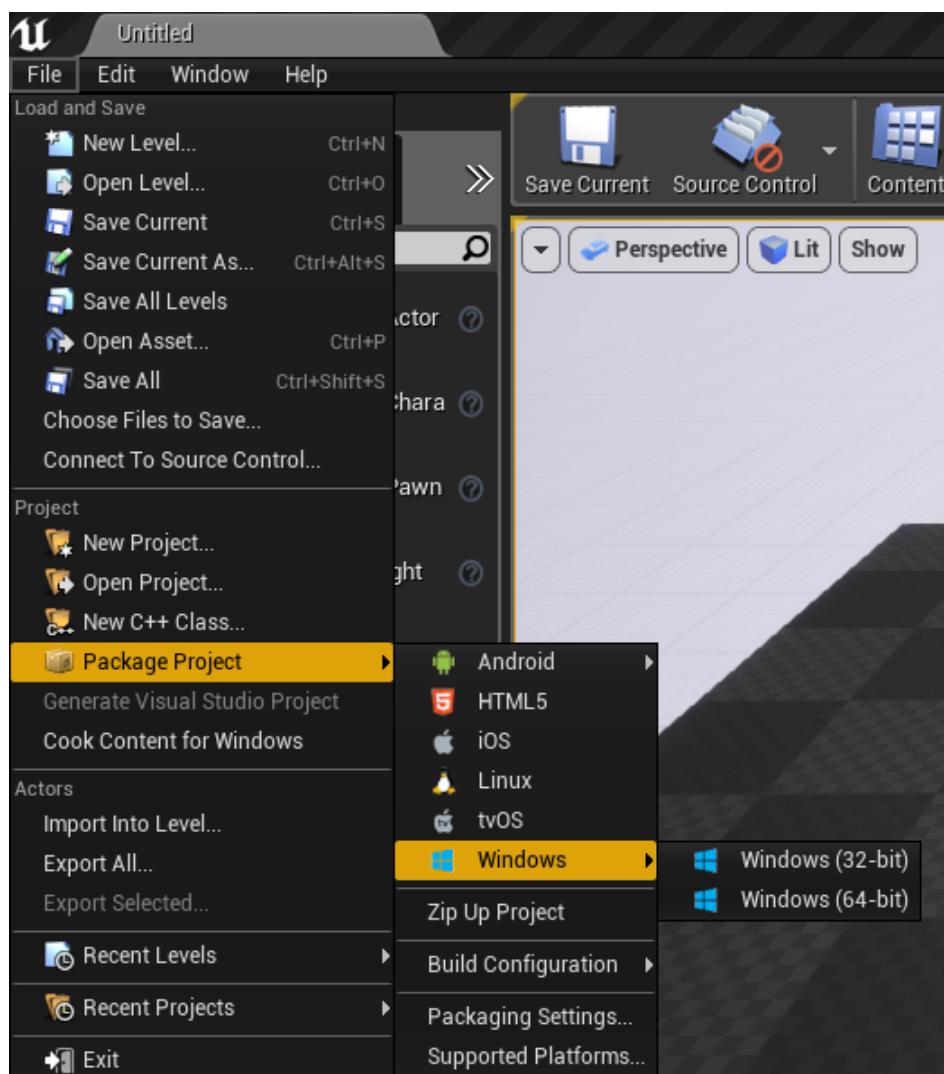


Figura 8.4: Ruta para generar una build del proyecto

Una vez terminado el lento proceso de construcción tendremos disponible nuestra build del proyecto lista para ejecutar en nuestro dispositivo de preproducción.

En este proyecto se ha optado por lo siguiente:

- Realizar builds de nuestro producto en el modo «Development» cada vez que se implementase por completo una nueva funcionalidad, para probarla individualmente, aprovechando el proceso de construcción para elaborar la documentación referente a la funcionalidad.
- Realizar una build por iteración en el modo «Shipping», trasladar esta versión del producto al equipo de pruebas, y testear todas y cada unas de las funcionalidades que componen la iteración.

SEGUNDA ITERACIÓN

Durante esta iteración nos centraremos en el sistema de escalada, uno de los ejes principales del proyecto, e implementaremos acciones como agarrar salientes, saltar de uno a otro o carreras por las paredes.

9.1 INTRODUCCIÓN

En primer término, vamos a intentar centrarnos en la parte más importante del sistema de escalada: la base del mismo. Como se verá más adelante esta base es la misma, con algunas pequeñas modificaciones y añadidos, del tutorial seguido para realizar parte de esta iteración.

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 comprobar 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 pero, además, por el mismo motivo se hace sólo cuando un componente adherido al personaje, denominado «climb surface detector», encuentra alguna superficie escalable.

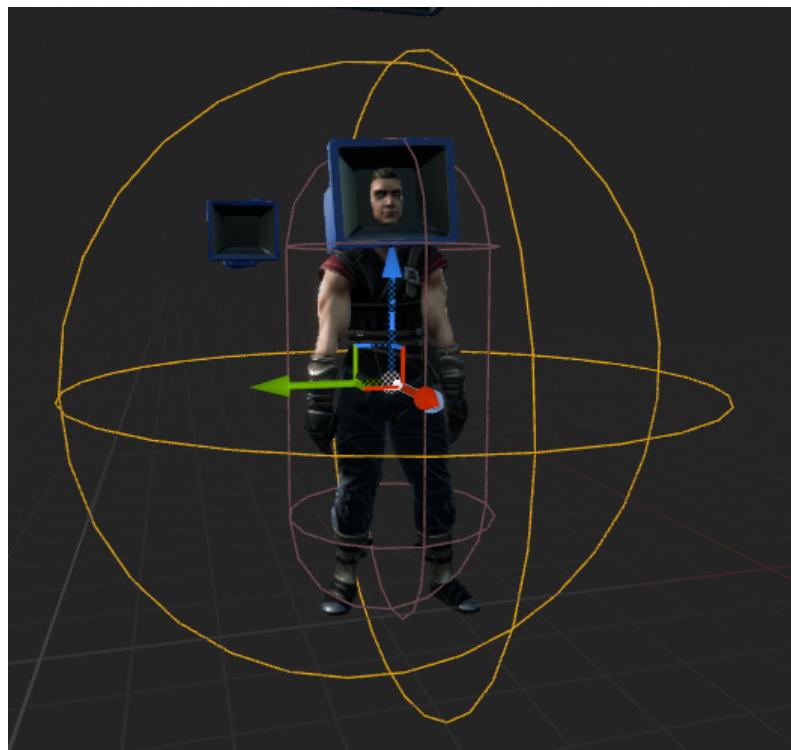


Figura 9.1: Imagen en la que podemos apreciar la esfera detectora de superficie escalable

Para comprobar si una superficie es escalable o no utilizaremos «canales», en concreto un **canal especial**, distinto a los creados por defecto en Unreal Engine, de forma que podamos detectar las superficies que queremos e ignorar el resto.

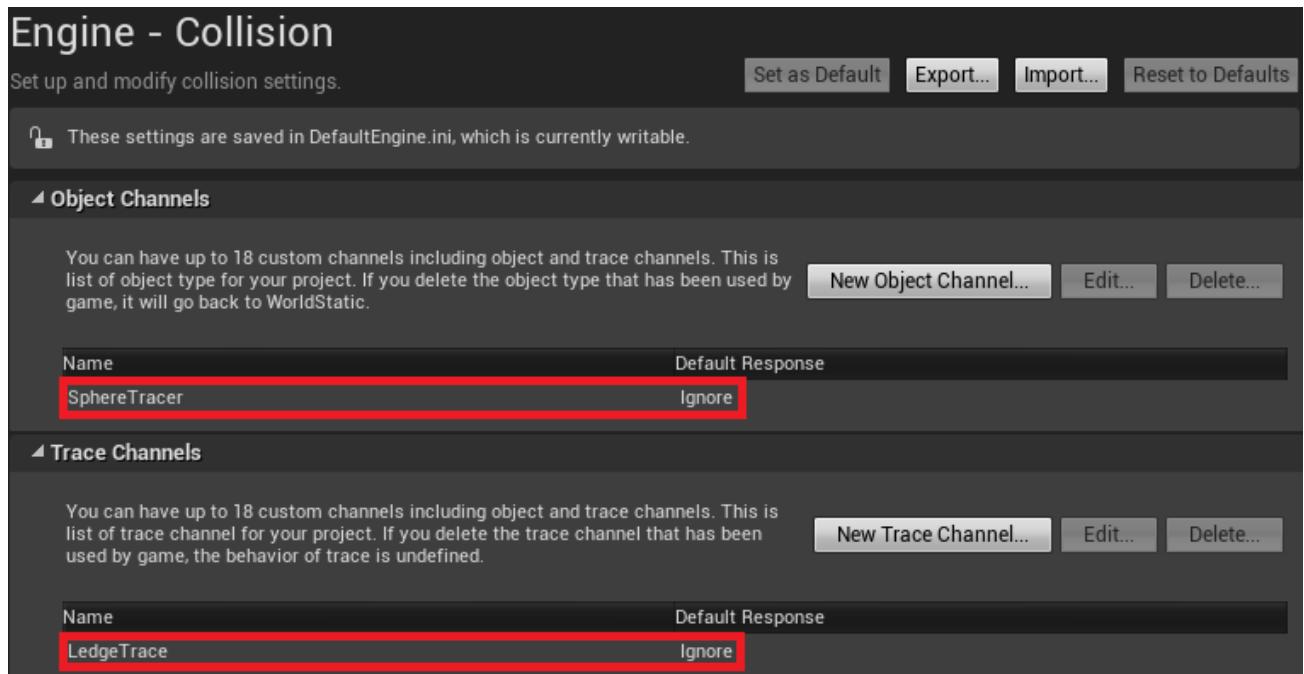


Figura 9.2: Configuración de canales del proyecto

La idea general es que las superficies escalables:

1. Generen un evento de solapamiento sobre el detector de superficie escalable, para activar el sistema. Por tanto, es muy importante activar la opción «Generate overlap events» en la superficie escalable, puesto que en caso contrario nunca funcionará el sistema.
2. Bloqueen la traza que se realice por el canal destinado a la detección de superficies escalables, para que así se genere un impacto sobre ellas y se pueda detectar satisfactoriamente (dicho canal ignorará todo lo demás).

Además de esto, y como ya se intuye, las trazas destinadas a detectar superficies escalables se tienen que generar en el canal «LedgeTracer».

Ahora vamos a ver las comprobaciones que se realizan respecto a este sistema de escalada. Nótese que el canal «especial» hace referencia al sistema de canales que hemos creado.

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 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

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

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

9.2 CARACTERÍSTICAS A DESARROLLAR

1. Base del sistema de escalada (*).
 - Implementación del código de la base del sistema de escalada.
2. Mecánica: Agarrar un saliente (*).
 - Implementación del código de la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
3. Mecánica: Agarrar un saliente pequeño y escalarlo (*).
 - Implementación del código de la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
4. Mecánica: Escalar un saliente al que estemos agarrados (*).
 - Implementación del código de la mecánica.
 - Asignación de teclas y botones vinculados la mecánica.
5. Mecánica: Soltar un saliente (*).
 - Implementación del código de la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
6. Mecánica: Movernos por un saliente.
 - Implementación del código de la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
7. Mecánica: Saltar desde saliente lateralmente.
 - Implementación del código de la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
8. Mecánica: Mirar hacia atrás desde saliente.
 - Implementación del código de la mecánica.
 - Desarrollo de la animación del personaje relacionada con la mecánica.
9. Mecánica: Saltar desde saliente mirando hacia atrás

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

10. Mecánica: Escalada en pared (*).

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

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

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

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

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

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

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

9.3 DISEÑO

Memorando técnico 2-001: Base del sistema de escalada	
Asunto	Base del sistema de escalada
Resumen	Se requiere obtener información del entorno para usarla en las posteriores funcionalidades del sistema de escalada a implementar.
Factores causantes	-
Solución	La capsula de colisión del jugador se envolverá con una esfera de colisión, que siempre que tenga una superficie de escalada cerca empezará a funcionar: se trazarán líneas o esferas desde el personaje hacia diferentes puntos (o viceversa), siguiendo los canales de trazo que sean necesarios, para con esto detectar obstáculos, objetos escalables, etc.
Motivación	Si queremos obtener información en tiempo real del entorno, es la única solución que nos sirve.
Cuestiones abiertas	-
Alternativas	-

Cuadro 9.3: Memorando técnico 2-001: Base del sistema de escalada

Memorando técnico 2-002: 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 9.4: Memorando técnico 2-002: Agarrar un saliente

Memorando técnico 2-003: 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 9.5: Memorando técnico 2-003: Agarrar un saliente pequeño para escalarlo

Memorando técnico 2-004: 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 9.6: Memorando técnico 2-004: Escalar un saliente al que estemos agarrados

Memorando técnico 2-005: 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 9.7: Memorando técnico 2-005: Soltar un saliente

Memorando técnico 2-006: 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 9.8: Memorando técnico 2-006: *Movernos por un saliente*

Memorando técnico 2-007: 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 9.9: Memorando técnico 2-007: *Saltar desde saliente lateralmente*

Memorando técnico 2-008: 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 9.10: Memorando técnico 2-008: Mirar hacia atrás desde saliente

Memorando técnico 2-009: 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 9.11: Memorando técnico 2-009: Saltar desde saliente mirando hacia atrás

Memorando técnico 2-010: 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 9.12: Memorando técnico 2-010: Escalada en pared

Memorando técnico 2-011: 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 9.13: Memorando técnico 2-011: Giro y salto durante escalada en pared

9.3.1 Modificaciones

Respecto a la iteración anterior, debemos de hacer dos pequeñas modificaciones:

1. La acción de «salto» cambia respecto a la iteración anterior, puesto que ahora el personaje saltará o no dependiendo de si puede hacer una carrera en pared: Si se puede realizar una carrera en pared se le dará prioridad a ésta y si no, el personaje simplemente saltará, como se hacía hasta ahora.
2. Igualmente, la acción de «agachar» ahora no sólo se usa para agachar al personaje, sino que en caso de encontrarse en un saliente, al ejecutarse esta acción se soltará de él (en caso contrario, seguirá agachándose).

9.4 IMPLEMENTACIÓN

ID	Descripción de la acción de alto nivel
2-001	Base del sistema de escalada (PRIMERA PARTE)
Métodos de alto nivel	
[void] DoTrace ()	
Pasos	
<p>0. Este algoritmo se ejecutará si la esfera de colisión del personaje tiene dentro de ella algún actor escalable.</p> <p>1. Comprueba que, o bien, el personaje esté «cayendo» y no esté agarrando un saliente, o bien esté realizando una escalada en pared:</p> <p>2. Se traza una esfera desde el pecho del personaje hacia delante del mismo, buscando una pared escalable.</p> <p>2.1. Si hay impacto con una superficie escalable:</p> <p>2.1.1. Si el jugador presiona «saltar» podrá comenzar una escalada por pared.</p> <p>2.2. Si no hay impacto con una superficie escalable:</p> <p>2.2.1. No ocurre nada.</p> <p>3. Se traza una esfera desde arriba hacia abajo un poco a la derecha y delante del personaje para buscar si el personaje podría poner su mano derecha en la pared escalable que encuentre y otra traza de igual manera, pero esta vez buscando el hipotético punto de apoyo de la mano izquierda.</p> <p>3.1. Si ambas líneas consiguen impactar:</p> <p>3.1.1. Según la diferencia de altura entre el saliente encontrado y la cadera del personaje, éste procederá a quedarse enganchado sin más o a hacer una escalada automática con las rodillas.</p>	

ID	Descripción de la acción de alto nivel	
2-001	Base del sistema de escalada (SEGUNDA PARTE)	
Métodos de alto nivel		
[void] DoTrace ()		
Pasos		
4. Si el personaje está agarrado:		
4.1. Se traza una esfera desde el hipotético punto de apoyo del pie del personaje mientras agarra hacia delante para buscar si el personaje podría poner su pie derecho ahí y otra traza de igual manera, pero esta vez buscando el hipotético punto de apoyo del pie izquierdo.		
4.1.1. Si al menos uno de las dos esferas impacta:		
4.1.1.1. Cambia el valor de «bCanBraceHang» de la clase a verdadero.		
4.1.1.2. Cambia el valor de «bCanBraceHang» del blueprint encargado de animar al personaje a verdadero.		
4.1.1.3. Aplica las restricciones de cámara oportunas.		
4.1.2. Si no impacta ninguna esfera:		
4.1.2.1. Cambia el valor de «bCanBraceHang» de la clase a falso.		
4.1.2.2. Cambia el valor de «bCanBraceHang» del blueprint encargado de animar al personaje a falso.		
4.1.2.3. Aplica las restricciones de cámara oportunas.		
Métodos de bajo nivel necesarios		
Paso	Clase	Método
4.1.1.3.	AMainCharacter	[void] RestrictView (YawMin:float, Yaw-Max:float, PitchMin:float)
4.1.2.3.	AMainCharacter	[void] RestrictView (YawMin:float, Yaw-Max:float, PitchMin:float)

ID	Descripción de la acción de alto nivel	
2-002	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		
Paso	Clase	Método
1.2.10.	AMainCharacter	[void] RestrictView (YawMin:float, YawMax:float, PitchMin:float)

ID	Descripción de la acción de alto nivel	
2-003	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		
Paso	Clase	Método
1.2.7.	AMainCharacter	[void] RestrictView (YawMin:float, YawMax:float, PitchMin:float)

ID	Descripción de la acción de alto nivel	
2-004	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		
Paso	Clase	Método
1.2.7.	AMainCharacter	[void] RestrictView (YawMin:float, YawMax:float, PitchMin:float)

ID	Descripción de la acción de alto nivel	
2-004	Escalar un saliente al que estemos agarrados (Final)	
Métodos de alto nivel		
[void] CompleteClimb ()		
Pasos		
<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		
Paso	Clase	Método
7.	AMainCharacter	[void] ResetRestrictView ()

ID	Descripción de la acción de alto nivel	
2-005	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		
Paso	Clase	Método
12.	AMainCharacter	[void] ResetRestrictView ()

ID	Descripción de la acción de alto nivel
2-006	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> <p>1.1.1. Detenemos el movimiento del personaje inmediatamente.</p> <p>1.1.2. Cambia el valor de «GrabRightInput» del blueprint encargado de animar al personaje a 0.</p> <p>1.1.3. Cambia el valor de «GrabLookingRightInput» del blueprint encargado de animar al personaje a 0.</p> <p>1.2. Si por el contrario no se cumple:</p>	

ID	Descripción de la acción de alto nivel
2-006	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
2-006	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	
2-007	Saltar desde saliente lateralmente	
Métodos de alto nivel		
[void] JumpGrabbingSide ()		
Pasos		
<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>		
Métodos de bajo nivel necesarios		
Paso	Clase	Método
1.	AMainCharacter	[void] LeaveLedge ()
2.1.1.	ACharacter	[void] LaunchCharacter (LaunchVelocity:FVector, bXYOverride:bool, bZOverride:bool)
2.2.1.	ACharacter	[void] LaunchCharacter (LaunchVelocity:FVector, bXYOverride:bool, bZOverride:bool)

ID	Descripción de la acción de alto nivel	
2-008	Mirar hacia atrás desde saliente	
Métodos de alto nivel		
[void] GrabLedgeRear ()		
Pasos		
	<p>1. Comprobamos el valor de la variable «bIsGrabbingLookingRear»:</p> <p>1.1. Si es verdadera no hacemos nada.</p> <p>1.2. Si es falsa:</p> <p>1.2.1. Cambia el valor de «bIsGrabbingLookingRear» de la clase a verdadera.</p> <p>1.2.2. Cambia el valor de «bIsGrabingAndLookingRear» del blueprint encargado de animar al personaje a verdadero.</p> <p>1.2.3. Detenemos el movimiento inmediatamente</p> <p>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.</p> <p>1.2.5. Aplica restricciones de cámara.</p>	
Métodos de bajo nivel necesarios		
Paso	Clase	Método
1.2.5.	AMainCharacter	[void] RestrictView (YawMin:float, YawMax:float, PitchMin:float)

ID	Descripción de la acción de alto nivel	
2-009	Saltar desde saliente mirando hacia atrás	
Métodos de alto nivel		
[void] JumpGrabbingRear ()		
Pasos		
	<p>1. Usamos el método «LeaveLedge()» para abandonar el saliente primero.</p> <p>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)</p> <p>3. Giramos 180 grados nuestro personaje</p>	
Métodos de bajo nivel necesarios		
Paso	Clase	Método
2.	ACharacter	[void] LaunchCharacter (LaunchVelocity:FVector, bXYOverride:bool, bZOverride:bool)

ID	Descripción de la acción de alto nivel
2-010	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	
2-010	Giro y salto durante escalada en pared	
Métodos de alto nivel		
[void] WallRunningRearJump ()		
Pasos		
1. Comprobamos que «bIsWallRunning» sea verdadero y «bIsClimbingLedge» falso:		
1.1. Si ocurre:		
1.1.1. Paramos la escalada en pared.		
1.1.2. Giramos 180 grados al personaje.		
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)		
1.1.4. Ajusta el valor de la variable «bCanDoWallRunning» a verdadero.		
1.2. Si no ocurre:		
1.2.1. No pasa nada.		
Métodos de bajo nivel necesarios		
Paso	Clase	Método
1.1.3.	ACharacter	[void] LaunchCharacter (LaunchVelocity:FVector, bXYOverride:bool, bZOverride:bool)

9.5 PRUEBAS

Esta vez, para la realización de las pruebas de esta iteración se ha decidido crear un mapa llamado «**Test_ClimbMechanics**» que, de nuevo, no será accesible desde el juego (pero sí desde la versión de editor del desarrollador) cuyo único objetivo es testear cada una de las implementaciones de las mecánicas desarrolladas en la iteración.

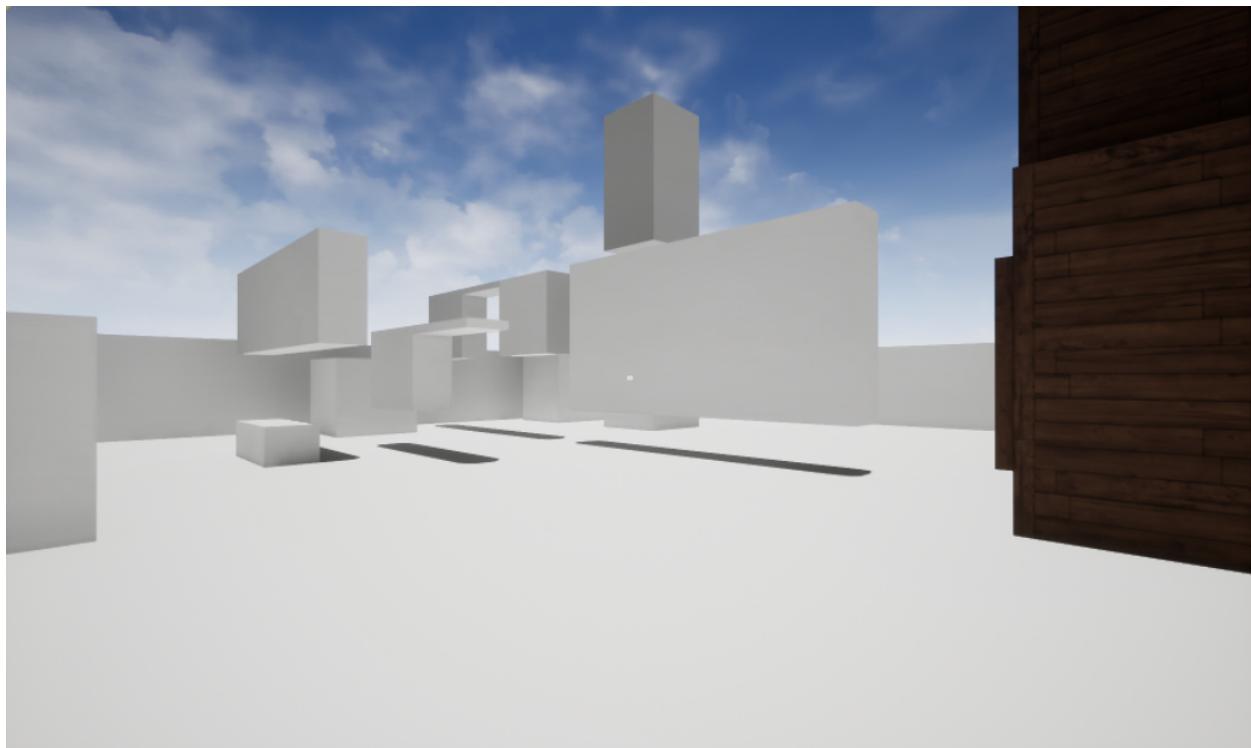


Figura 9.3: Fotograma del mapa de pruebas empleado en la iteración 2

El mapa contiene distintos salientes de distintas alturas suficiente para llevar a cabo las pruebas de escalada.

Batería de pruebas: Agarrar un saliente (2-002)

1. El personaje se debe quedar agarrado a un saliente si está en el aire y alineado con la pared del mismo.
2. El personaje se debe quedar agarrado a un saliente si durante una carrera por pared, al finalizar la misma, el saliente queda por encima de la cintura del personaje (y aún en rango y cumple los requisitos del test anterior).
3. En cualquier caso, al quedarse agarrado a un saliente, se cambiará a la animación de agarre de saliente y se alinearán las manos del personaje.
4. El personaje no debe agarrarse a un saliente no está alineado con la pared del mismo.
5. Cuando agarremos un saliente, se nos debe restringir la cámara para que, si la movemos, no veamos el interior del modelo de nuestro personaje.

Cuadro 9.14: Batería de pruebas: Agarrar un saliente (2-002)

Batería de pruebas: Auto-escalar un saliente (2-003)

1. El personaje debe escalar un saliente si durante una carrera por pared, al finalizar la misma, el saliente queda por debajo de la cintura del personaje, hay superficie donde colocar al personaje después escalar el saliente y el personaje está alineado con la pared del saliente.
2. Cuando el personaje escale, debe reproducir la animación de auto-escalada.
3. En cualquier otro caso, no debe hacer nada.
4. Mientras estemos escalando el saliente se deberá restringir la cámara, impidiendo que veamos el modelo de nuestro personaje si giramos la misma.
5. Al escalar el saliente se nos deberá de eliminar cualquier restricción de cámara que tuviésemos.

Cuadro 9.15: Batería de pruebas: *Auto-escalar un saliente (2-003)***Batería de pruebas: Escalar un saliente al que estemos agarrados (2-004)**

1. Si estamos agarrados a un saliente y hay espacio para colocar al personaje cuando éste escale, al pulsar el botón «saltar» el personaje debe escalar el saliente, reproduciendo la animación de escalada de saliente.
2. Si, por el contrario, no hay espacio, no debe hacer nada.
3. Mientras estemos escalando el saliente se deberá restringir la cámara, impidiendo que veamos el modelo de nuestro personaje si giramos la misma.
4. Al escalar el saliente se nos deberá de eliminar cualquier restricción de cámara que tuviésemos.

Cuadro 9.16: Batería de pruebas: *Escalar un saliente al que estemos agarrados (2-004)*

Batería de pruebas: Soltar un saliente (2-005)

1. Si estamos agarrados a un saliente y pulsamos «agachar», el personaje debe soltarse y dejarse caer del mismo.
2. Al abandonar el saliente, debe abandonarse la animación de agarrar saliente (y reproducir la que proceda, típicamente la de caída).
3. Al abandonar el saliente también debemos abandonar cualquier restricción de cámara que tuviésemos.

Cuadro 9.17: *Batería de pruebas: Soltar un saliente (2-005)*

Batería de pruebas: Movernos por un saliente (2-006)

1. Cuando estamos agarrados a un saliente, si hay espacio para agarrarse a la izquierda y pulsamos el botón de movimiento «izquierda», el personaje debe moverse hacia ese lado, reproduciendo la animación de moverse por el saliente hacia la izquierda.
2. Cuando estamos agarrados a un saliente, si hay espacio para agarrarse a la derecha y pulsamos el botón de movimiento «derecha», el personaje debe moverse hacia ese lado, reproduciendo la animación de moverse por el saliente hacia la derecha.
3. Cuando nos estamos moviendo por un saliente hacia la izquierda, se acaba el espacio para agarrarse por la izquierda y hay algún obstáculo que nos impediría saltar por ese lado, el personaje debe detener su movimiento y parar su animación.
4. Cuando nos estamos moviendo por un saliente hacia la derecha, se acaba el espacio para agarrarse por la derecha y hay algún obstáculo que nos impediría saltar por ese lado, el personaje debe detener su movimiento y parar su animación.
5. Cuando nos estamos moviendo por un saliente hacia la izquierda y se acaba el espacio para agarrarse por ese lado pero no hay ningún obstáculo que nos impida saltar, el personaje debe detener su movimiento pero también soltarse de su mano izquierda, indicando que puede saltar.
6. Cuando nos estamos moviendo por un saliente hacia la derecha y se acaba el espacio para agarrarse por ese lado pero no hay ningún obstáculo que nos impida saltar, el personaje debe detener su movimiento pero también soltarse de su mano derecha, indicando que puede saltar.

Cuadro 9.18: Batería de pruebas: Movernos por un saliente (2-006)

Batería de pruebas: Saltar desde saliente lateralmente (2-007)

1. Cuando nuestro personaje está en posición de saltar hacia la izquierda (mirar prueba anterior) y pulsamos el botón «saltar», nuestro personaje debe hacer un salto hacia ese lado.
2. Cuando nuestro personaje está en posición de saltar hacia la derecha (mirar, de nuevo, prueba anterior) y pulsamos el botón «saltar», nuestro personaje debe hacer un salto hacia ese lado.
3. Si pulsamos el botón de «agachar», el personaje debe soltarse igualmente del saliente.

Cuadro 9.19: Batería de pruebas: Saltar desde saliente lateralmente (2-007)

Batería de pruebas: Mirar hacia atrás desde saliente (2-008)

1. Cuando nuestro personaje está agarrado a un saliente y mantenemos pulsado el botón de movimiento «atrás», debe reproducirse una animación y permitirnos mirar hacia atrás.
2. Mientras que estamos mirando hacia atrás, nuestra cámara se debe restringir, impidiendo que ésta se meta dentro del modelo de personaje.
3. Si pulsamos el botón de «agachar», el personaje debe soltarse igualmente del saliente.

Cuadro 9.20: Batería de pruebas: Mirar hacia atrás desde saliente (2-008)

Batería de pruebas: Saltar desde saliente mirando hacia atrás (2-009)

1. Cuando pulsemos el botón «saltar» estando agarrados y mirando hacia atrás, nuestro personaje debe soltarse del saliente, abandonado la animación de estar agarrado mirando atrás.
2. En el mismo momento de saltar, se nos deben eliminar cualquier restricción de cámara que acarreásemos.

Cuadro 9.21: *Batería de pruebas: Saltar desde saliente mirando hacia atrás (2-009)***Batería de pruebas: Escalada en pared (2-010)**

1. Cuando pulsemos el botón «saltar» estando cerca de una superficie escalable y alineados con ella el personaje debe pegarse a la pared y empezar a escalarla, reproduciéndose la animación de escalada por pared.
2. Mientras estemos escalando por la pared se deberá restringir la cámara, impidiendo que veamos el modelo de nuestro personaje si giramos la misma.
3. Al terminar la escalada por pared se nos deberá de eliminar cualquier restricción de cámara que tuviésemos, siempre que no entremos realizando otra acción (es decir, por ejemplo, hayamos agarrado un saliente).

Cuadro 9.22: *Batería de pruebas: Escalada en pared (2-010)*

Batería de pruebas: Giro y salto durante escalada en pared (2-011)

1. Cuando pulsemos el botón «saltar» estando realizando una escalada en pared, la cámara y nuestro personaje deben rotarse e impulsarse en dirección contraria a la pared.
2. En el mismo momento de saltar, se nos deben eliminar cualquier restricción de cámara que acarreásemos.

Cuadro 9.23: Batería de pruebas: Giro y salto durante escalada en pared (2-011)

9.6 DESPLIEGUE

Los cambios se han desplegado en el sistema de producción de manera similar al resto de iteraciones (Ver sección §8.6).

TERCERA ITERACIÓN

La tercera iteración se centra principalmente en la realización del menú principal (y su mapa asociado), de ajustes y de pausa, la creación del HUD del usuario y la implementación del sistema de muerte y de guardado.

10.1 RESUMEN

La principal característica a implementar será el sistema de guardado y carga del juego. Los datos que queremos guardar se separan en tres archivos diferentes, como recoge el documento de diseño:

1. **Archivo «Save»:** A grandes rasgos guarda el mapa y la localización y rotación del personaje. Se actualizan al pasar un mapa o alcanzar un punto de control. Se borran completamente al iniciar un nuevo juego.
2. **Archivo «Achievements»:** Guarda los logros que hemos conseguido en el juego. Se actualizan cada vez que se consigue un avance en un logro (aunque no se llegue a conseguir ninguno como tal). No se resetean al comenzar una nueva partida.
3. **Archivo «Settings»:** Almacena la configuración del usuario (resolución, opciones gráficas, etc.). Se actualizan cada vez que el usuario cambia un ajuste en el menú. De igual manera que los anteriores no se resetean al comenzar una nueva partida.

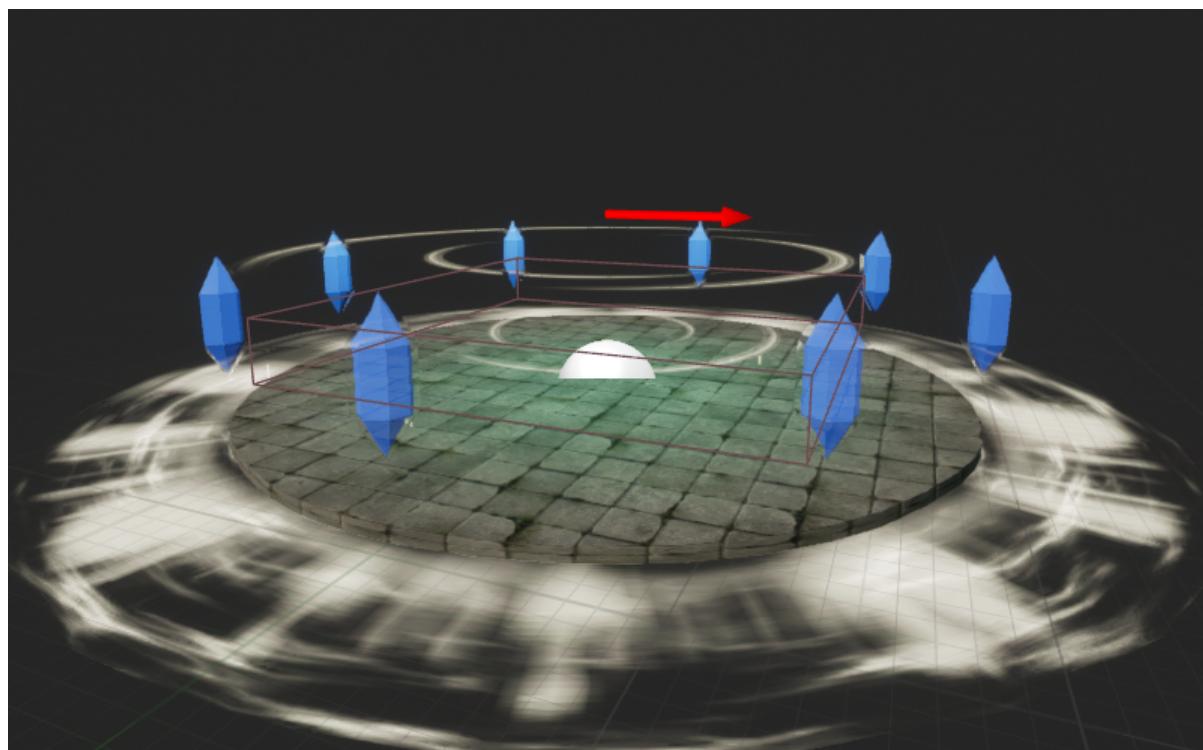


Figura 10.1: Captura de un «checkpoint» (o punto de control) usado en el juego

Otra gran característica implementar es el sistema de muerte, en el que se tratará de buscar, como siempre, la solución más simple posible y se detallará más en la fase de diseño: no tendremos que guardar ningún tipo de vida máxima / restante del personaje puesto que lo que se producirá al interaccionar con algún objeto que dañe (aspecto que exploraremos en más profundidad en la siguiente iteración, puesto que ahora sólo podríamos morir abandonando el escenario o colisionando con alguna caja de colisión colocada en un foso o similar) al jugador será una muerte instantánea. Esto no se hace por simplicidad, aunque obviamente es más simple, sino que es una decisión de diseño, ya que el juego se basa en la repetición.

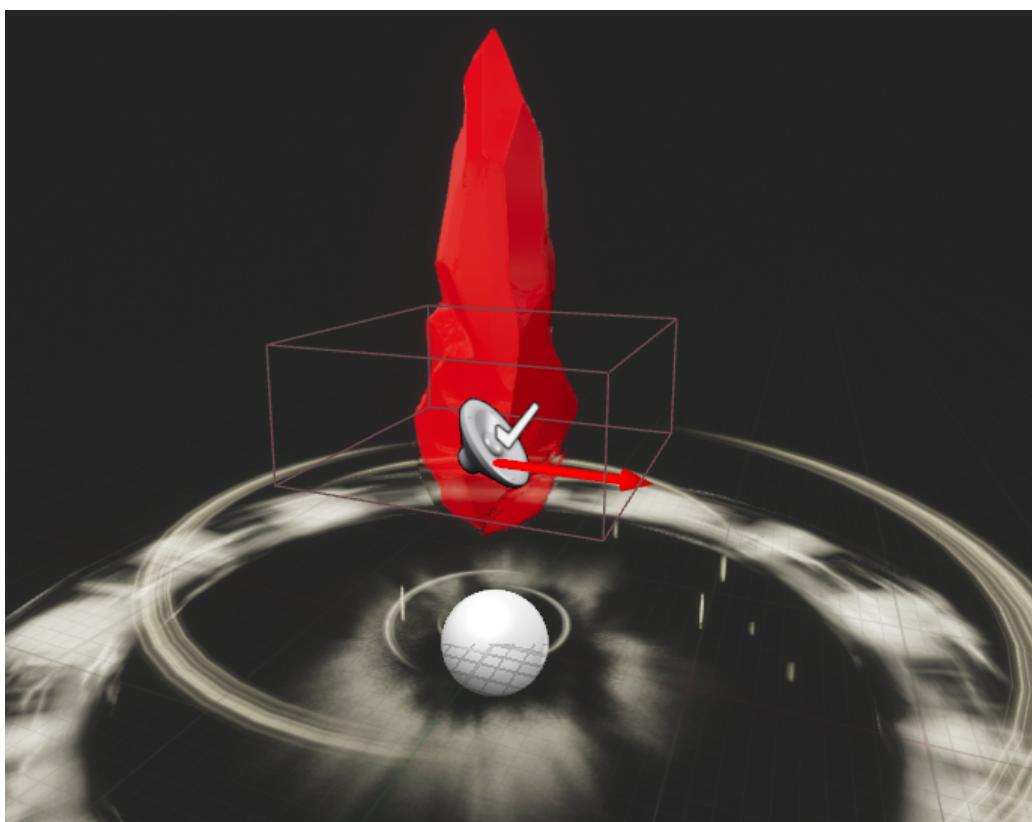


Figura 10.2: Imagen de un cristal de teletransportación rojo, usado para avanzar al siguiente nivel

Para la elaboración del HUD del usuario se implementará una interfaz que hará de puente entre la clase del personaje principal y el HUD en sí, enviando información del primero al segundo.

Dado que el HUD es un trabajo que no encajaría demasiado en el esquema que estamos siguiendo para la iteraciones, podemos ver más sobre él en su subsección del manual (Ver sección §14.5).

10.2 CARACTERÍSTICAS A DESARROLLAR

1. Sistema de guardado / carga.

- Implementación en código.

2. Sistema de muerte.

- Implementación del código de la mecánica.
- Recreación de la muerte del personaje cambiando su estado a «ragdoll» o «muñeco de trapo».

3. Creación del HUD del usuario, con los siguientes elementos:

- Punto de referencia.
- Indicador de «blink».
- Indicador de adrenalina restante.
- Brújula, que le permita conocer al jugador el punto al que debe dirigirse para acabar el nivel.
- Mensaje de muerte, que aparezca sólo cuando el usuario esté muerto.
- Caja de tutorial, que serán usadas para mostrarles mensajes al jugador hasta que pulse el botón «acción».
- Recuadro de logro, que aparecerá en pantalla cuando el jugador obtenga uno.
- Recuadro de juego guardado, que aparecerá al guardar juego.

4. Creación de un menú principal:

- Creación del mapa del menú principal.
- Opción: Nuevo juego, que nos permita empezar una partida desde cero.
- Opción: Cargar juego, que nos permita continuar nuestro progreso.
- Opción: Ajustes, que nos permita realizar cambios en la configuración.
- Opción: Salir del juego.

5. Creación de un menú de pausa:

- Opción: Nuevo juego, que nos permita empezar una partida desde cero.

- Opción: Cargar juego, que nos permita continuar nuestro progreso.
- Opción: Ajustes, que nos permita realizar cambios en la configuración.
- Opción: Cerrar el menú.
- Opción: Volver al menú principal.
- Opción: Salir del juego.

6. Creación de un menú de ajustes, que nos de las siguientes opciones:

- Opción: Seleccionar idioma (inglés o español).
- Opción: Seleccionar relación de aspecto de la resolución.
- Opción: Seleccionar resolución.
- Opción: Seleccionar entre modo pantalla completa y modo ventana.
- Opción: Seleccionar diversos ajustes gráficos.
- Opción: Cerrar el menú.

10.3 DISEÑO

Memorando técnico 3-001: Sistema de guardado / carga	
Asunto	Sistema de guardado / carga
Resumen	Se requiere que el usuario pueda guardar sus progresos en el juego y guardarlos.
Factores causantes	-
Solución	Cuando el personaje controlado por el usuario entre en contacto con un punto de control o pase al siguiente nivel, se actualizará el archivo de guardado. Para cargar la partida sólo tendrá que seleccionar la opción «Cargar partida» en el menú principal o de pausa.
Motivación	Es la solución más fácil para el fin que estamos buscando.
Cuestiones abiertas	-
Alternativas	-

Cuadro 10.1: Memorando técnico 3-001: Sistema de guardado / carga

Memorando técnico 3-002: Sistema de muerte	
Asunto	Sistema de muerte
Resumen	Consistiría en la realización de un sistema que detectase la muerte del personaje y ejecutase la secuencia de muerte.
Factores causantes	-
Solución	Cuando nuestro personaje entre en contacto con algún actor que tenga la etiqueta «deadly» se lanzará la escena de muerte y se le quitará el control al usuario.
Motivación	Es la alternativa más fácil.
Cuestiones abiertas	-
Alternativas	Alternativas hay muchas, pero de las exploradas ésta es la más simple.

Cuadro 10.2: Memorando técnico 3-002: Sistema de muerte

Memorando técnico 3-003: Opción menú - Nuevo juego	
Asunto	Opción menú - Nuevo juego
Resumen	Cuando se pulse esta opción debe iniciarse un nuevo juego.
Factores causantes	-
Solución	Se borra la partida anterior, si la hay, y se inicia un nuevo juego.
Motivación	Es la alternativa más fácil.
Cuestiones abiertas	-
Alternativas	-

Cuadro 10.3: Memorando técnico 3-003: Opción menú - Nuevo juego

Memorando técnico 3-004: Opción menú - Cargar juego

Asunto	Opción menú - Cargar juego
Resumen	Cuando se pulse esta opción se debe continuar desde el último mapa o punto de control.
Factores causantes	-
Solución	Se busca en el archivo de guardado principal y se carga el mapa almacenado y se coloca al personaje en la localización y rotación también almacenada.
Motivación	Es la alternativa más fácil.
Cuestiones abiertas	-
Alternativas	-

Cuadro 10.4: Memorando técnico 3-004: Opción menú - Cargar juego

Memorando técnico 3-005: Opción menú - Ajustes

Asunto	Opción menú - Ajustes
Resumen	Se desea que se acceda al menú de ajustes tanto desde menú principal como desde el de pausa.
Factores causantes	-
Solución	Se oculta el menú principal / de pausa y se muestra el menú de ajustes.
Motivación	Es la alternativa más fácil.
Cuestiones abiertas	-
Alternativas	-

Cuadro 10.5: Memorando técnico 3-005: Opción menú - Ajustes

Memorando técnico 3-006: Opción menú - Salir	
Asunto	Opción menú - Salir
Resumen	Se desea que el menú principal / de pausa contenga una opción para cerrar el juego.
Factores causantes	-
Solución	Al pulsar la opción cerramos el juego.
Motivación	Es la alternativa más fácil.
Cuestiones abiertas	-
Alternativas	-

Cuadro 10.6: Memorando técnico 3-006: Opción menú - Salir

Memorando técnico 3-007: Opción menú pausa - Cerrar menú	
Asunto	Opción menú pausa - Cerrar menú
Resumen	Se requiere que si estamos en el menú de pausa, podamos cerrarlo.
Factores causantes	-
Solución	Al pulsar la opción cerramos el menú de pausa.
Motivación	Es la alternativa más fácil.
Cuestiones abiertas	-
Alternativas	-

Cuadro 10.7: Memorando técnico 3-007: Opción menú - Cerrar menú

Memorando técnico 3-008: Opción menú pausa - Volver al menú principal

Asunto	Opción menú pausa - Volver al menú principal
Resumen	Se requiere que si estamos en el menú de pausa, podamos volver al menú principal.
Factores causantes	-
Solución	Al pulsar la opción volvemos al menú principal.
Motivación	Es la alternativa más fácil.
Cuestiones abiertas	-
Alternativas	-

Cuadro 10.8: Memorando técnico 3-008: Opción menú pausa - Volver al menú principal

10.4 IMPLEMENTACIÓN

ID	Descripción de la acción de alto nivel
3-001	Sistema de guardado (al tocar un punto de control)
Métodos de alto nivel	
[void] ActorBeginOverlap () - Clase RespawnPoint	
Pasos	
<p>1. Comprueba si existe un archivo de guardado:</p> <p>1.1. Si existe:</p> <p>1.1.1. Carga el archivo de guardado.</p> <p>1.2. Si no existe:</p> <p>1.2.1. Crea un nuevo archivo de guardado.</p> <p>2. Crea un objeto «SaveReference» y asigna el nuevo archivo de guardado creado o el cargado, para trabajar más cómodamente.</p> <p>3. Obtiene el nivel actual, actualiza el «SaveReference» con el valor.</p> <p>4. Obtiene la posición del punto de control, actualiza el «SaveReference» con el valor.</p> <p>5. Obtiene la rotación del punto de control, actualiza el «SaveReference» con el valor.</p> <p>6. Fija el valor de «bFromStart» del «SaveReference» a falso.</p> <p>7. Obtiene si el personaje tiene poderes, actualiza el «SaveReference» con el valor.</p> <p>8. Guarda el objeto «SaveReference» en el slot (espacio de guardado) «Save», por lo que sustituye al archivo «Save» anterior.</p>	

ID	Descripción de la acción de alto nivel
3-001	Sistema de guardado (al pasar de nivel)
Métodos de alto nivel	
[void] ActorBeginOverlap () - Clase CrystalLevelEndPoint	
Pasos	
	0. Si el usuario pulsa el botón de acción mirándolo:
	1. Comprueba si existe un archivo de guardado:
	1.1. Si existe:
	1.1.1. Carga el archivo de guardado.
	1.2. Si no existe:
	1.2.1. Crea un nuevo archivo de guardado.
	2. Crea un objeto «SaveReference» y asigna el nuevo archivo de guardado creado o el cargado, para trabajar más cómodamente.
	3. Obtiene el nivel actual, actualiza el «SaveReference» con el valor.
	4. Fija el valor del vector «Location» a (0.0, 0.0, 0.0).
	5. Fija el valor del vector «Rotation» a (0.0, 0.0, 0.0).
	6. Fija el valor de «bFromStart» del «SaveReference» a verdadero.
	7. Fija el valor de «bIsPowered» del «SaveReference» a falso.
	8. Guarda el objeto «SaveReference» en el slot (espacio de guardado) «Save», por lo que sustituye al archivo «Save» anterior.
	9. Carga el nivel indicado en la variable del CrystalLevelEndPoint.

ID	Descripción de la acción de alto nivel
3-001	Sistema de carga
Métodos de alto nivel	
[void] OnBeginPlay ()	
Pasos	
1. Intenta acceder a los datos del archivo de guardado «Save»:	
1.1. Si hay archivos guardados:	
1.1.1. Se comprueba si en el archivo de guardado se especifica que se debe iniciar el nivel desde el principio y el nivel almacenado en el archivo:	
1.1.1.1. Si «RespawnFromStart» es falsa y el nivel almacenado en el save es el nivel en el que nos encontramos:	
1.1.1.1.1. Se coloca el personaje en la localización y rotación indicada en el archivo de guardado.	
1.1.1.2. En caso contrario:	
1.1.1.2.1. No pasa nada (el jugador saldría desde el punto inicial del mapa).	
1.2. Si no hay archivos guardados:	
1.2.1. No pasa nada (el jugador saldría desde el punto inicial del mapa).	

ID	Descripción de la acción de alto nivel
3-002	Sistema de muerte (1/2)
Métodos de alto nivel	
[void] OnDeath ()	
Pasos	
<p>1. Se comprueba si el personaje está muerto («bIsDead»):</p> <p>1.1. Si el personaje no está muerto:</p> <p>1.1.1. Se fija la variable «bIsDead» a verdadero.</p> <p>1.1.2. Se fuerza la parada del tiempo bala.</p> <p>1.1.3. Se desactiva el control para el usuario.</p> <p>1.1.4. Se cambia el modo de colisión del modelo de nuestro personaje a «ragdoll» (muñeco de trapo).</p> <p>1.1.5. Se marca simular físicas como verdadero en el modelo de nuestro personaje.</p> <p>1.1.6. Se desactiva la cámara de primera persona.</p> <p>1.1.7. Se desactiva la cámara de tercera persona.</p> <p>1.1.8. Se activa la cámara de muerte.</p> <p>1.1.9. Se tiñe la pantalla de rojo y se destiñe en 2,5 segundos, progresivamente.</p> <p>1.1.10. Se reproduce el sonido de muerte del personaje.</p> <p>1.1.11. Se fija una cuenta atrás de 3 segundos.</p> <p>1.1.12. Cuando la cuenta atrás termina se llama a la función OnDeathEnd().</p> <p>1.2. Si el personaje está muerto:</p> <p>1.2.1. No ocurre nada.</p>	

ID	Descripción de la acción de alto nivel
3-002	Sistema de muerte (2/2)
Métodos de alto nivel	
[void] OnDeathEnd ()	
Pasos	
<p>1. Intenta acceder a los datos del archivo de guardado «Save»:</p> <p>1.1. Si el archivo existe:</p> <p>1.1.1. Carga el nivel indicado en el archivo de guardado.</p> <p>1.2. Si no existe:</p> <p>1.2.1. Carga el nivel del tutorial.</p> <p>2. Desactiva la simulación de físicas en el modelo de personaje.</p> <p>3. Cambia la variable «bIsDead» a falso.</p> <p>4. Desactiva la cámara de muerte.</p> <p>5. Activa la cámara en primera persona.</p> <p>6. Desactiva la cámara en tercera persona.</p> <p>7. Se le vuelve a dar control al usuario.</p> <p>8. Se cambia el perfil de colisión del modelo del personaje a «CharacterMesh» (perfil que tenía antes de morir).</p>	

ID	Descripción de la acción de alto nivel
3-003	Opción menú - Nuevo juego
Métodos de alto nivel	
[void] OnClicked (Button_NewGame)	
Pasos	
<p>1. Reproduce un sonido de click.</p> <p>2. Comprueba si existe el archivo de guardado «Save»:</p> <p>2.1. Si existe:</p> <p>2.1.1. A efectos prácticos, el sistema borra la partida.</p> <p>2.2. Si no existe:</p> <p>2.2.1. No hace nada.</p> <p>3. El sistema abre el nivel «Introduction».</p> <p>4. Elimina el HUD del menú principal del viewport.</p> <p>5. Fija el modo de input a «Game Only».</p> <p>6. Deja de mostrar el cursor.</p>	

CAPÍTULO 10. TERCERA ITERACIÓN

ID	Descripción de la acción de alto nivel
3-004	Opción menú - Cargar juego
Métodos de alto nivel	
[void] OnClicked (Button_LoadGame)	
Pasos	
<p>1. Reproduce un sonido de click.</p> <p>2. Comprueba si existe el archivo de guardado «Save»:</p> <p>2.1. Si existe:</p> <p>2.1.1. Carga el mapa almacenado en el archivo de guardado.</p> <p>2.1.2. Elimina el HUD del menú principal del viewport.</p> <p>2.1.4. Fija el modo de input a «Game Only».</p> <p>2.1.5. Deja de mostrar el cursor.</p> <p>2.1. Si no existe:</p> <p>2.2. No hace nada.</p>	

ID	Descripción de la acción de alto nivel
3-005	Opción menú - Ajustes
Métodos de alto nivel	
[void] OnClicked (Button_Settings)	
Pasos	
<p>1. Reproduce un sonido de click.</p> <p>2. Oculta, con una animación, el panel principal del menú principal / de pausa.</p> <p>3. Muestra, igualmente con una animación, el panel principal secundario del menú principal / de pausa, que es el que contiene los ajustes.</p>	

ID	Descripción de la acción de alto nivel
3-006	Opción menú - Salir
Métodos de alto nivel	
[void] OnClicked (Button_Exit)	
Pasos	
<p>1. Reproduce un sonido de click.</p> <p>2. Cierra el juego.</p>	

ID	Descripción de la acción de alto nivel	
3-007	Opción menú pausa - Cerrar menú	
Métodos de alto nivel		
[void] OnClicked (Button_Close)		
Pasos		
1. Reproduce un sonido de click.		
2. Llama a la función OnPause().		
Métodos de bajo nivel necesarios		
Paso	Clase	Método
1	AMainCharacter	[void] OnPause ()

ID	Descripción de la acción de alto nivel
3-008	Opción menú pausa - Volver al menú principal
Métodos de alto nivel	
[void] OnClicked (Button_ReturnMainMenu)	
Pasos	
1. Reproduce un sonido de click.	
2. Abre el nivel «MainMenu».	

10.5 PRUEBAS

Batería de pruebas: Sistema de guardado / carga (3-001)

1. Cuando pisamos una superficie de punto de control, el juego se debe guardar instantáneamente.
2. Cuando saltamos de un nivel a otro mediante un cristal de teletransportación rojo, el juego se debe guardar instantáneamente.
3. Cuando cargamos el juego, deben cargarse correctamente todos los datos guardados: mapa, localización del personaje, rotación, etc.
4. En caso de no tener aún un punto de control en el mapa, al cargar se colocará al personaje al principio del mismo.
5. El personaje, aunque acabe un nivel con poderes, no puede aparecer en otro con ellos.

Cuadro 10.9: Batería de pruebas: Sistema de guardado / carga (3-001)

Batería de pruebas: Sistema de muerte (3-002)

1. Cuando nuestro personaje entre en contacto con algún peligro que le haga daño, debe morir instantáneamente.
2. Cuando nuestro personaje muere, se deben activar las físicas de «muñeco de trapo» y dejarse caer colisionando con los objetos con los que impacte.
3. Cuando nuestro personaje muere se desactivará totalmente el control por parte del usuario.
4. Cuando nuestro personaje muere, pasados 3 segundos, el juego debe cargar automáticamente el último punto de control o, en caso de no tener aún un punto de control en el mapa, el mapa desde el principio.
5. Cuando nuestro personaje reaparece, se le devolverá el control sobre el mismo al usuario.

Cuadro 10.10: Batería de pruebas: Sistema de muerte (3-002)

Batería de pruebas: Opción menú - Nuevo juego (3-003)

1. Al pulsar el botón, se reproducirá un sonido de click.
2. Cuando pulsamos en el botón de nuevo juego, si no tenemos partidas guardadas simplemente empezará el juego desde el primer mapa.
3. Cuando pulsamos en el botón de nuevo juego, si tenemos partidas guardadas las borrará y empezará el juego desde el primer mapa.

Cuadro 10.11: Batería de pruebas: Opción menú - Nuevo juego (3-003)

Batería de pruebas: Opción menú - Cargar juego (3-004)

1. Al pulsar el botón, se reproducirá un sonido de click.
2. Si no tenemos partidas guardadas, el botón estará desactivado.
3. Cuando pulsamos en el botón de cargar juego, si tenemos partidas guardadas se nos emplazará en el último punto de control alcanzado o al inicio del mapa en cuestión si no tenemos puntos de control en el mismo.

Cuadro 10.12: Batería de pruebas: Opción menú - Cargar juego (3-004)

Batería de pruebas: Opción menú - Ajustes (3-005)

1. Al pulsar el botón, se reproducirá un sonido de click.
2. Cuando pulsemos en el botón «Ajustes», se ocultará el panel principal del menú principal / de pausa. Se reproducirá una animación.
3. Justo después, se mostrará el panel de ajustes en sí. Igualmente se reproducirá una animación.

Cuadro 10.13: Batería de pruebas: Opción menú - Ajustes (3-005)

Batería de pruebas: Opción menú - Salir (3-006)

1. Al pulsar el botón, se reproducirá un sonido de click.
2. Se cerrará el juego.

Cuadro 10.14: Batería de pruebas: Opción menú - Salir (3-006)

Batería de pruebas: Opción menú pausa - Cerrar menú (3-007)

1. Al pulsar el botón, se reproducirá un sonido de click.
2. Se cerrará el menú de pausa.

Cuadro 10.15: Batería de pruebas: Opción menú pausa - Cerrar menú (3-007)

Batería de pruebas: Opción menú pausa - Volver al menú principal (3-008)

1. Al pulsar el botón, se reproducirá un sonido de click.
2. Se cargará el mapa del menú principal.

Cuadro 10.16: Batería de pruebas: Opción menú pausa - Volver al menú principal (3-008)

10.6 DESPLIEGUE

Los cambios se han desplegado en el sistema de producción de manera similar al resto de iteraciones (Ver sección §8.6).

CUARTA ITERACIÓN

Una vez superado el ecuador de la iteraciones, nos centraremos en los peligros que el usuario tendrá que sortear mientras avanza en el juego y, principalmente, en las mecánicas de los puzzles que tendrá que resolver.

11.1 INTRODUCCIÓN

Lo primero que tenemos que mencionar es que las mecánicas de esta iteración se implementan principalmente mediante glsplblueprint. Esto es por dos razones fundamentales:

1. La primera es que iteración tiene un alto componente «visual», en el sentido de que tendremos que manejar modelos constantemente: colocarlos,
2. Y la segunda es que las mecánicas que se implementan en esta iteración no suponen un coste computacional importante como para mermar el rendimiento del proyecto.

11.1.1 Sistema de puzzles

La iteración girará mayoritariamente sobre la implementación las mecánicas de puzzles. La idea general respecto a esto es los assets que forman parte de estas mecánicas que se dividen en dos grandes categorías:

1. Los denominados «**Powerables**»: objetos capaces de dar poder o dejar de darlo.
2. Y los denominados «**Activatables**»: Objetos capaces de dar activarse o desactivarse.

De esta manera, los objetos capaces de dar poder («powerables») pueden dar o no poder a los objetos capaces de activarse o desactivarse («activatables»), cosa que harán en función de si el otro objeto le da poder o no.

Todos estos objetos funcionan con un sistema de herencias, los objetos capaces de dar poder heredan de la clase «powerable» («abstracta») y, por otro lado, los capaces de activarse o desactivarse heredan a su vez de la clase «activatable» («abstracta» también”).

Nota: Se entrecomilla abstracta por una sencilla razón, y es que no se pueden hacer glsplblueprint abstractos, pero a efectos prácticos lo son.

Powerables

Son, como se decía, los objetos capaces de enviar señales de activado o desactivado a los «activatables».

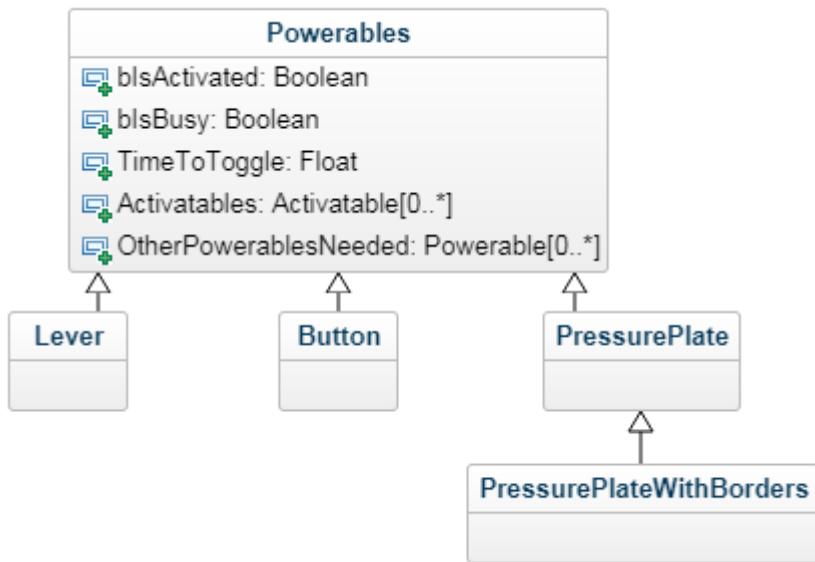


Figura 11.1: Representación de la herencia de los «powerables»

Como se aprecia en la representación de la figura, los «powerables» heredan de la clase principal una serie de atributos, que luego implementan cada una de las clases hijas a su manera.

- **bIsActivated - Booleano:** Representa si el «powerable» está mandando poder o no.
- **bIsBusy - Booleano:** Se usa a modo de candado, para bloquear cuando está en proceso de cambio de estado.
- **Activatables - Array de activatables:** Contiene todos los «activatables» que se activarán o desactivaran en función de este «powerable».
- **OtherPowerablesNeeded - Array de powerables:** En determinadas ocasiones queremos que no se active directamente un «activatable», sino que necesite de más de un «powerable» para activarse. Este array contiene los elementos que son necesarios que estén activados para que se mande la señal de activación a un «activatable».

Activatables

Por el contrario son, como se decía también previamente, los objetos capaces de activarse o desactivarse, dependientes de un «powerable».

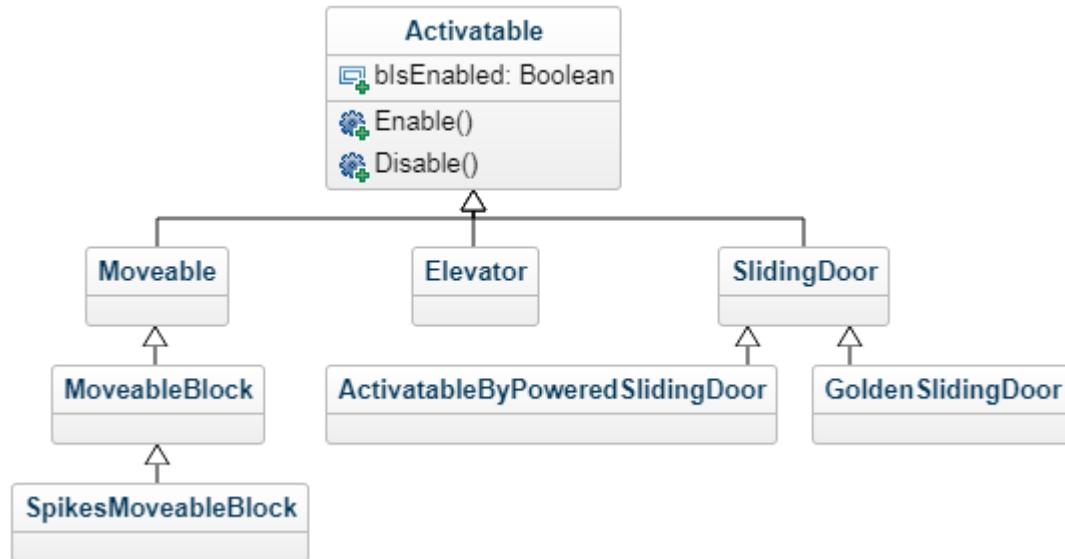


Figura 11.2: Representación de la herencia de los «activatables»

Como también se aprecia esta vez en la representación de la figura, los «activatables» heredan de la clase principal (entre otras cosas) la variable «bIsEnable» y, esta vez, dos funciones «Enable()» y «Disable()»:

- **bIsEnable - Booleano:** Representa si el «activatable» está activado o no (dependiendo de la clase de «activatable» que sea, esto generará un comportamiento u otro).
- **Función «Enable()»:** Activa el «activatable» (el comportamiento dependerá del tipo del mismo).
- **Función «Disable()»:** Desactiva el «activatable» (el comportamiento, también, dependerá del tipo del mismo).

Pickables

Los «pickables» son objetos que podemos recoger del suelo y que tienen forma de estatuillas. Nos sirven para resolver puzzles, normalmente por razones asociadas a su peso, ya que hereda también de una clase padre que vuelve a ser «abstracta» (recordemos, virtualmente) y cada una de las hijas se diferencian en que tienen un material asignado y un peso:

Tabla de pesos de las estatuillas «pickables»	
Material de la estatuilla	Peso
Cristal	0,5 Kg
Madera	1 Kg
Arcilla	1,5 Kg
Piedra	2 Kg
Metal	3,5 Kg

Cuadro 11.1: Tabla de pesos de las estatuillas «pickables»

Estas estatuillas se pueden colocar sobre placas de presión (con bordes o no), sumando peso a la misma y activándolas si se cumplen los requisitos y, además, influyen en puzzles de físicas.

Otros elementos de puzzles

Además, se implementarán:

1. **Un expositor para los «pickables»:** No son más que un pequeño expositor con un actor del tipo «pickable» a elegir.
2. **Rampa y cilindro:** Usan físicas y a su vez se usan para puzzles de físicas. Consiste de un cilindro en cuya parte superior se sitúa una tabla, sobre la que podremos subirnos para alcanzar lugares más elevados (con cuidado, porque bajará). Sobre la tabla también podremos colocar «pickables» para que hagan de contrapeso y poder llegar a lugares más elevados.
3. **Display digital:** Están vinculados con una placa de presión. Muestran el peso actual y el peso objetivo necesario para activar la placa de presión.

11.1.2 Peligros a implementar

Se implementarán:

1. **Láseres (*)**: Serán capaces de moverse al sitio asignado (y retroceder) en un intervalo de tiempo determinado y con un delay entre ida y venida también determinado.
2. **Bloques móviles**: Al igual que los láseres, se mueven a un sitio asignado y vuelven si así se les indica. También se les indica la velocidad y el delay entre ida y venida.
3. **Bloques móviles con pinchos (modulables)**: Heredan de los bloques móviles y son modulables en el sentido de que podemos, activando variables booleanas, activar los pinchos de un lado y otro, siendo mortal el contacto con el bloque sólo por el lado en el que tenga los pinchos activados.

(*) De los peligros que pueden matar al personaje, los láseres no tienen la tag «deadly», que mataría automáticamente el personaje. Esto es porque queremos que el personaje pueda traspasar sin morir estos láseres si utiliza un teletransporte a través de ellos, por lo que no se aplica esta etiqueta y, en su lugar, cuando el personaje contacta con el láser se comprueba que esté haciendo un teletransporte y, sino lo está, se activa directamente la secuencia de muerte.

El resto de peligros vendrán en forma de enemigos, que se implementarán durante la siguiente iteración.

11.2 CARACTERÍSTICAS A DESARROLLAR

11.3 DISEÑO

11.4 IMPLEMENTACIÓN

11.5 PRUEBAS

11.6 DESPLIEGUE

Los cambios se han desplegado en el sistema de producción de manera similar al resto de iteraciones (Ver sección §8.6).

QUINTA ITERACIÓN

T *ODO TODO TODO*

12.1 CARACTERÍSTICAS A DESARROLLAR

12.2 DISEÑO

12.3 IMPLEMENTACIÓN

12.4 PRUEBAS

12.5 DESPLIEGUE

Los cambios se han desplegado en el sistema de producción de manera similar al resto de iteraciones (Ver sección §8.6).

SEXTA ITERACIÓN

La última iteración del proyecto se centra en el diseño y creación de los distintos niveles que componen el juego así como la creación de secuencias de cámara, efectos y otros aspectos similares relacionados con los escenarios del juego.

13.1 CARACTERÍSTICAS A DESARROLLAR

13.2 DISEÑO

13.3 IMPLEMENTACIÓN

13.4 PRUEBAS

Poco podemos hablar sobre este aspecto en esta iteración, puesto que más que software se han generado assets.

Por tanto el testeo en esta iteración ha estado mucho más alejado del mundo de la ingeniería software y más cercano al artesanal: la prueba una y otra vez de que los distintos escenarios, mecánicas implementadas en los mismos, enemigos, etc. funcionan correctamente y no hay fallos.

13.5 DESPLIEGUE

Los cambios se han desplegado en el sistema de producción de manera similar al resto de iteraciones (Ver sección §8.6).

PARTE IV

CIERRE DEL PROYECTO

MANUAL DE USUARIO

Durante este capítulo se muestran los distintos dispositivos de entrada que admite el videojuego, los controles asociados a cada uno de ellos, la explicación detallada de 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

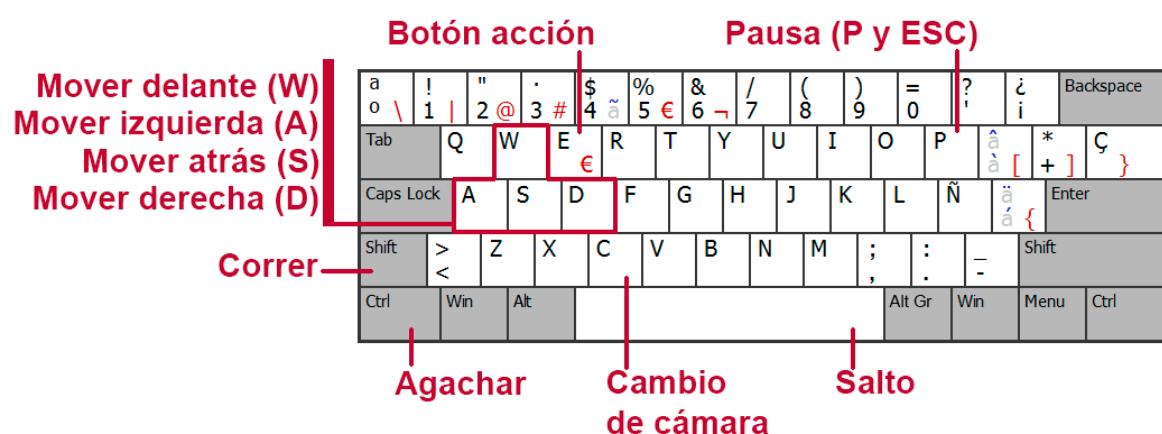


Figura 14.1: Diagrama de la asignación de acciones del teclado



Figura 14.2: Diagrama de la asignación de acciones del ratón

Listado detallado

- Tecla **W**: Desplazamiento del jugador hacia el frente.
- Tecla **A**: Desplazamiento del jugador hacia atrás.
- Tecla **S**: Desplazamiento del jugador hacia la izquierda.
- Tecla **D**: Desplazamiento del jugador hacia la derecha.
- **Eje X** del ratón: Regula el movimiento horizontal de la cámara.
- **Eje Y** del ratón: Regula el movimiento vertical de la cámara.
- Tecla **E**: Botón de acción.
- **Barra espaciadora**: Salto / Wall running / Salto en wall running / Escalar o saltar de saliente.
- Tecla **Ctrl Izquierdo**: Agachado / Abandonar saliente.
- Tecla **Shift Izquierdo**: Sprint.
- **Botón izquierdo** del ratón: Blink (*).
- **Botón derecho** del ratón: Comutador de tiempo bala (*).

Nota: Las acciones marcadas con (*) necesitan que el jugador tenga en su poder un 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

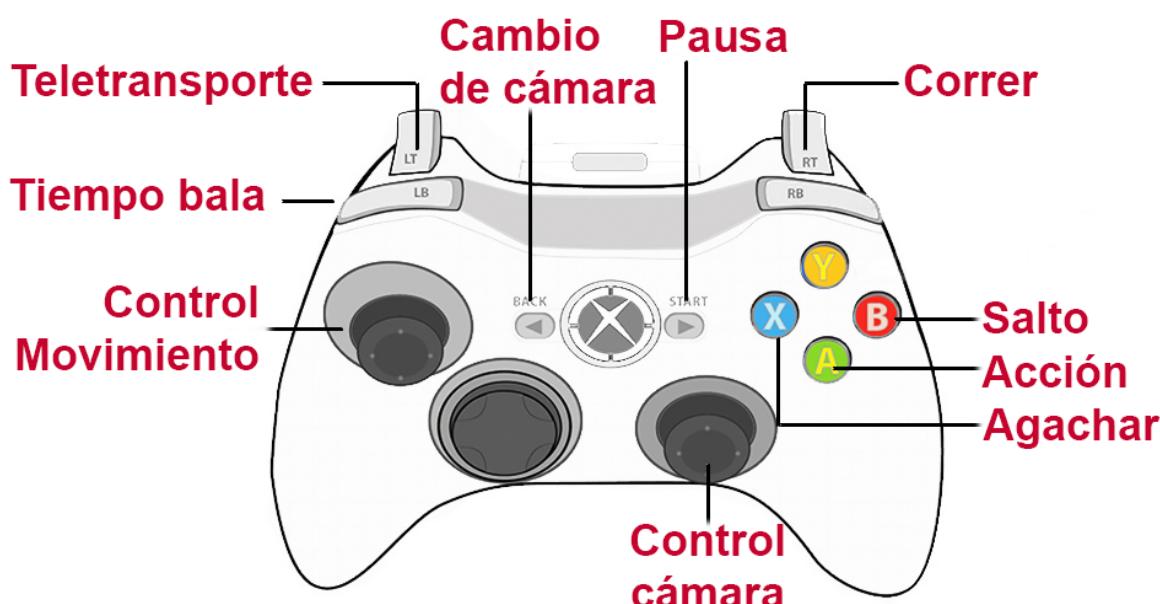


Figura 14.3: Diagrama de la asignación de acciones del controlador

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.

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

Nota: Las acciones marcadas con (*) necesitan que el jugador tenga en su poder un 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).

■ Resolución

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

■ Pantalla completa

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

■ Opciones gráficas

- **Descripción:** Nos permite ajustar la calidad visual del juego (así como, indirectamente, su rendimiento).
- **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

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

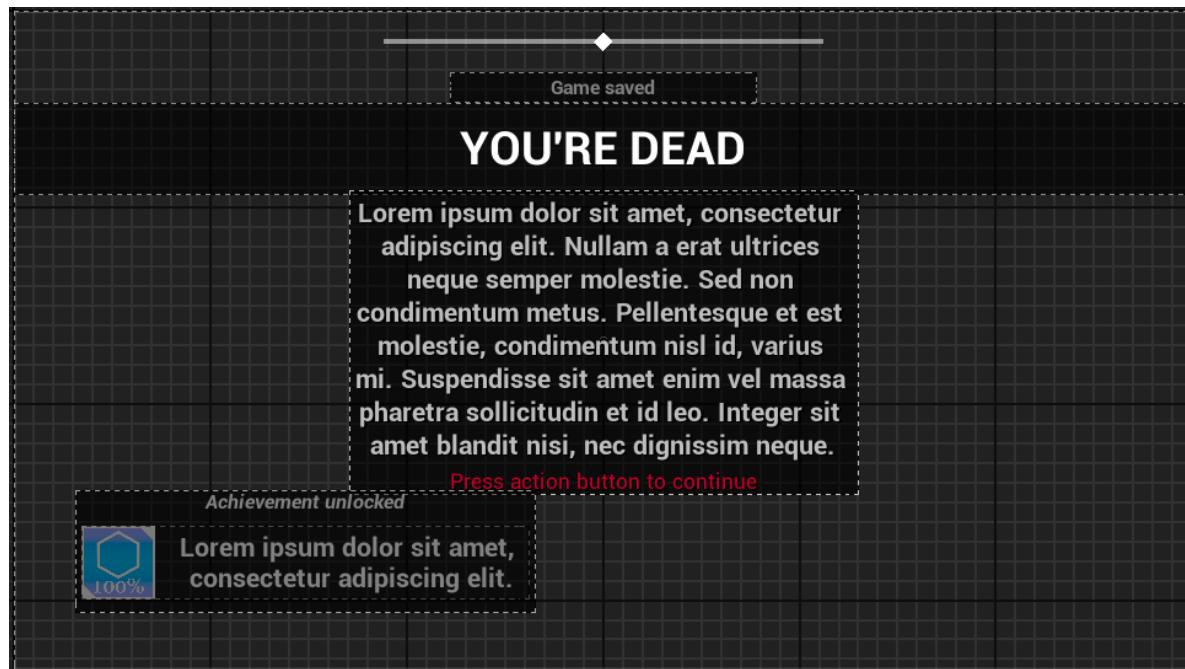


Figura 14.7: Instantánea mostrando el widget blueprint del HUD del usuario, en el que se pueden observar todos los elementos que se pueden mostrar en pantalla

Este HUD se compone de los siguientes elementos:

- **Punto de referencia:**

- **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.

■ **Indicador de habilidad «blink»:**

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

■ **Indicador de adrenalina:**

- **Tipo:** Barra.
- **Funcionalidad:** Muestra la cantidad de adrenalina restante.
- **Posición:** Adyacente al límite inferior de la pantalla.
- **Tamaño:** Ancho de la pantalla.
- **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 teletransportación 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.

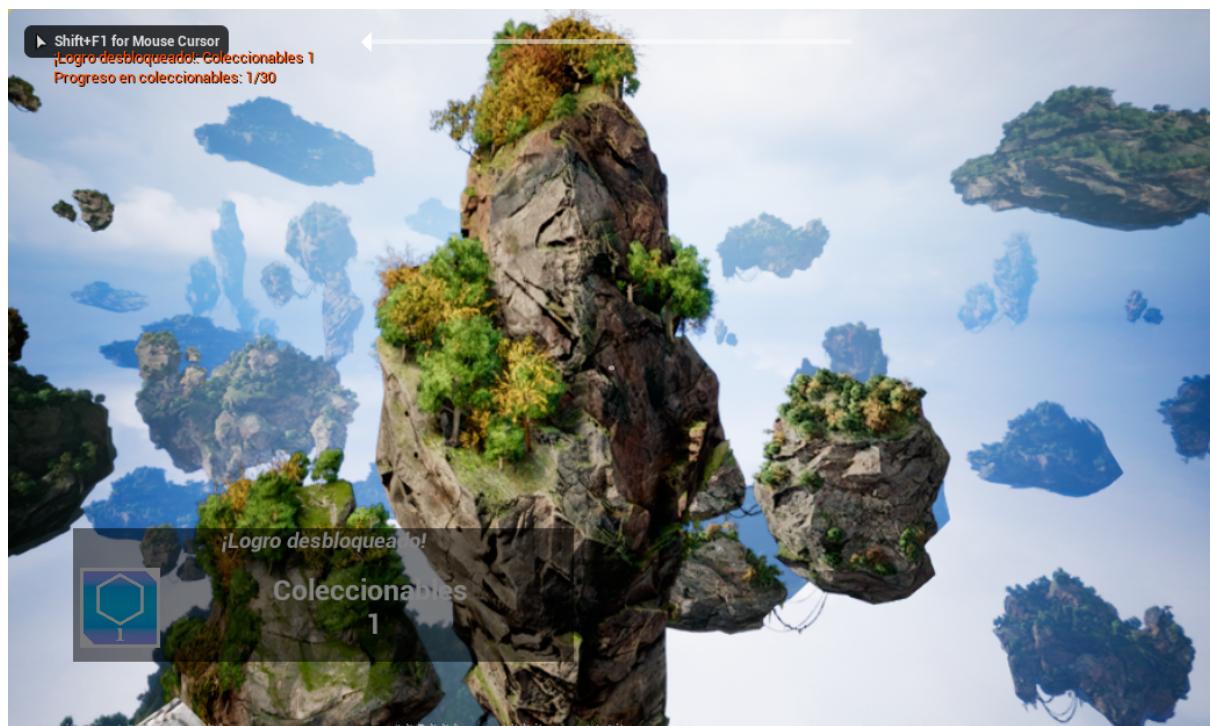


Figura 14.9: Captura del juego tomada durante la obtención de un logro

■ **Recuadro de logro:**

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

■ **Recuadro de tutorial:**

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

■ **Mensaje de fin de juego:**

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

■ **Recuadro de juego guardado:**

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

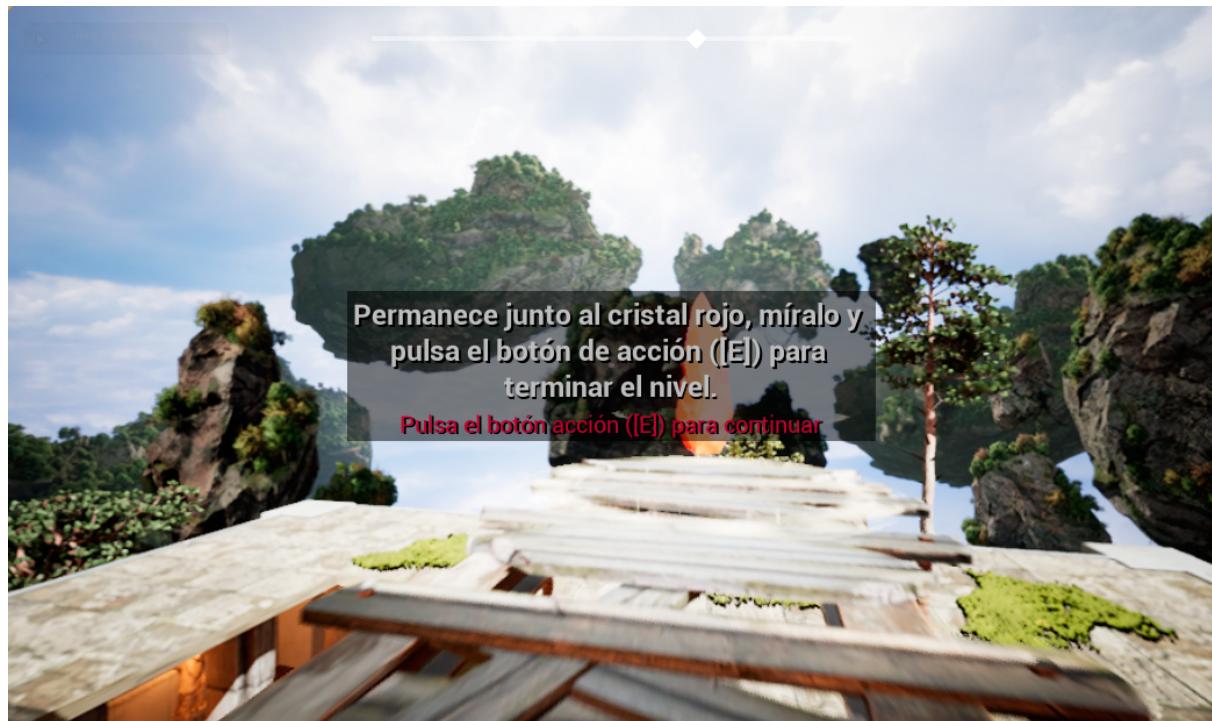


Figura 14.10: Captura del juego tomada durante un tutorial

CONCLUSIONES

En el último capítulo del cuerpo del documento, y a modo de clausura del mismo, se plasman a modo de memoria las conclusiones extraídas de la realización del proyecto, una pieza clave en la ingeniería software, lo que nos será de una inmensa utilidad en 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%» [17], 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.

Código «C++» frente a «blueprints»

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

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

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

- La mayoría de conocimiento que circula por la red está en blueprints, tanto es así que incluso hay diversas carencias en la documentación oficial de C++ de Epic Games frente a la documentación de blueprints, que es mucho más completa. Esto ha ocasionado que aprender a programar Unreal Engine en C++ fuese una tarea mucho más costosa y tediosa de lo que en un principio debería.
- Mientras que si usamos blueprints podemos acceder a variables, funciones o clases definidas en C++ sin ningún tipo de problema no ocurre lo mismo al contrario: para modificar una variable, función o clase de un blueprint no podemos hacerlo directamente, sino que tenemos, por ejemplo, hacer que el blueprint herede de una clase «cpp» y acceder o trabajar con dicho «cpp». Esto es un aspecto muy a tener en cuenta puesto que el uso de blueprints en Unreal Engine es prácticamente obligatorio para tareas como las de animación de un personaje o la implementación de inteligencia artificial.
- El tiempo de compilación de un blueprint se sitúa por debajo del segundo (en el equipo utilizado para el desarrollo de este proyecto), sin embargo, compilar código «C++» toma mucho más tiempo, llegando a tardar minutos en muchos casos, lo que laстра tanto el tiempo empleado para realizar una tarea como el ritmo de trabajo en sí, que se ve seriamente resentido ocasionando un desánimo en muchas ocasiones que debería ser evitable.
- Los blueprints, además de ser más intuitivos si no se domina a la perfección Unreal 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

Dificultad en la detección de errores

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

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

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

Cabe destacar un error en especial que persiguió al software desde la segunda iteración: en muy contadas ocasiones (estamos hablando de alrededor de un 1% de las veces) el personaje se agarraba a un saliente y empezaba a caer poco a poco en vez de quedarse a la misma altura del saliente, pero sin embargo la animación del personaje lo mostraba como agarrado, la variable que controla esto estaba en orden e incluso podíamos soltarnos del saliente o realizar alguna otra acción si las comprobaciones necesarias se daban. Algunas personas completaron el videojuego sin que este error se reprodujese pero otras experimentaron este bug sin que supieran describir mínimamente cómo llegaron a él. Tras algunas horas de testeо se llegó a la conclusión de que el error sólo se producía cuando llegábamos a la acción de agarrar el saliente provenientes de una escalada en pared: esa acción contiene una instrucción que fija el modo de movimiento del personaje en caída y por algún motivo seguía ejecutándose después de para la escalada en pared. Una simple instrucción «if», que controlase que el 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

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

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



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

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

Un videojuego requiere aptitudes muy dispares entre sí

Llevar a buen puerto un producto software con las características de un videojuego 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.

15.2 DISCUSIÓN

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

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

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

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

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

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

Por último, cabe hacer mención a una característica que finalmente ha podido ser incluida en el juego, pero que en un momento se pensó que podría ser descartada por limitaciones tecnológicas y que daría lugar a un nuevo proyecto: la implementación del cono de visión de los enemigos denominados como «vigilantes». Estos enemigos, como se detalla en este documento, entre otras particularidades poseen un cono de visión que debe ser renderizado por el juego para que el jugador pueda conocer el área de visión de dicho enemigo y así ocultarse en el momento oportuno, pero esto es un problema difícil de solucionar en más de un escenario, entre los que se incluye 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.



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.

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).

Dada las características de este proyecto es algo que vamos a ver en numerables ocasiones a lo largo de este documento de diseño, ya que mucha de las secciones del documento de diseño y de la memoria del trabajo, por la naturaleza de los mismos, se solapan.

B.1.1 Historia

En «The last ronin» la historia no es importante, es un título centrado plenamente en la jugabilidad.

No obstante, en el universo del juego encarnamos a un ronin, un samurai sin dueño, que ha obtenido la habilidad de teletransportarse entre templos antiguos y decide explorarlos en busca de sus misterios. Para teletransportarse necesitará encontrar un cristal rojo de teletransportación pero no todo será tan fácil, tendrá que ir superando los obstáculos y trampas mortales que lo separan de él en cada uno de los templos milenarios.

Por el camino, nuestro héroe podrá incluso ganar más poderes consiguiendo los orbes dorados, que le permiten desarrollar habilidades tales como la teletransportación a lugares cercanos y ralentizar el tiempo, lo que algunas veces le será esencial para superar los peligros que le acechan, como láseres, torretas y, sobre todo, los vigilantes del templo: seres de naturaleza extraña que han permanecido por siglos en los tiempos y los custodian, no dudando en eliminar a cualquier intruso.

B.1.2 Iconografía

Se ha generado la siguiente iconografía:

- Logo del juego:



Figura B.1: Logo del juego

- Icono:



Figura B.2: Logo del juego

B.1.3 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 (versiones de 32 y 64 bits).

B.1.4 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) y cabe destacar que gracias a la escalabilidad que dota el producto de las opciones gráficas, configurables en el menú ajustes, hará que el software sea jugable en una amplia gama de dispositivos.

B.1.5 Estilo gráfico

El estilo gráfico como tal 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, este estilo realista está combinado con un estilo fantástico por parte del entorno: islas voladoras, templos abandonados, etc.

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

B.1.6 Estilo sonoro

Hay dos músicas en el juego:

- La primera es la del menú principal: El tema se titula «Pray for Japan» y se eligió esta música puesto que nuestro personaje tiene la apariencia de un ninja.
- Y la restante es la que suena de fondo mientras jugamos al juego. En esta ocasión se llama «Puzzle game» y se eligió porque casi pasa inadvertida a la hora de jugar, y eso lo que se pretende, para que el jugador esté atento a las trampas y puzzles del juego.

Los efectos de sonido (utilizados cuando el personaje anda, salta, cae, etc.) también 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

Sección coincidente (Ver sección §14.2).

B.3 INTERFAZ DE USUARIO

B.3.1 Menús

El único menú que podrá consultar el jugador cuando esté en el juego será el menú de pausa. Sección coincidente (Ver sección §14.3).

B.3.2 In-game HUD

Sección coincidente (Ver sección §14.5).

B.3.3 In-game Options

Sección coincidente (Ver sección §14.4).

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

Si bien no existe pantalla de selección de nivel, el usuario de la aplicación podrá cargar su punto de guardado desde el menú principal o el menú de pausa.

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.
- Nivel de transición: Muestra un mensaje, cinemática, etc. al jugador, pero no le deja jugar en él.

Lista completa de niveles

La etapa de desarrollo que tendrá lugar durante el desarrollo de este proyecto generará 5 niveles en total:

- Nivel tutorial.
- Nivel de juego 1 y nivel de juego 2.
- Niveles de inicio y fin (tipo transición): Introduce el juego o le da las gracias al jugador por jugar el título y le muestra un botón para volver al menú principal, respectivamente.

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 de puzzles y de delimitación de puzzles:** Son las mecánicas relacionadas con la resolución de puzzles.
- **Mecánicas de enemigos:** Describe el comportamiento básico de los enemigos.
- **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. **Agachar:**
 - 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 puzzles

1. El jugador podrá abrir puertas doradas si tiene poderes en ese momento.
2. El jugador podrá abrir puertas de madera si encuentra una palanca, botón, placa de presión, etc. para activarla.
3. El jugador podrá recoger y soltar objetos específicos.
4. El jugador podrá interaccionar con placas de presión subiéndose encima de ellas o dejando algún objeto con peso encima.
5. El jugador podrá interaccionar con elementos móviles pudiéndoles colocar peso encima para modificar su comportamiento.

Mecánicas de delimitación de puzzles

1. El jugador no podrá pasar por una barrera mágica con un elemento de un puzzle de una sala a otra. Si se produce esto, se le quitará al jugador el objeto en cuestión y el juego lo devolverá a su posición original.
2. El jugador no podrá lanzar por una barrera mágica un elemento de un puzzle de una sala a otra. Si se produce esto, el objeto en cuestión desaparecerá y el juego lo devolverá a su posición original.

Mecánicas enemigos

1. Torretas (de cualquier tipo):

- a) Si no detectan ningún peligro (mediante el canal visual) las torretas estarán en modo de espera: se limitarán a elegir una posición dentro del rango especificado para rotar a ellas, dando la sensación de que están escaneando el entorno en buscar de nuestro personaje.
- b) Si detectan a nuestro jugador rápidamente se girarán hacia él y empezarán a dispararle, hasta que lo pierdan de vista. Si esto último pasa volverán al estado anterior.

2. Vigilantes (robots humanoides):

- a) Se moverán por el escenario de punto de control fijado en punto de control fijado. Estos puntos de control están enumerados y si hay más de un punto de control con el mismo número se moverán al más cercano.
- b) Si oyen al personaje, se girarán para intentar encontrarlo.
- c) Si el personaje se encuentra dentro del cono de visión del vigilante intentará atraparlo y dispararle un rayo, pero tendremos la opción de despistarlos.

Remarcando que los «vigilantes» o «los vigilantes del templo» son totalmente ciegos fuera de su cono de visión, pero pueden oírnos y girarse, teniendo más posibilidades de encontrarnos de esta forma.

Mecánicas del entorno

1. Si el personaje entra en contacto con algún peligro del escenario (como balas, láseres, pinchos, 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, morirá instantáneamente.
 - ... 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 el nivel, 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 %		
Logo	100%	
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 (que también, pero sin entrar en detalles assets por assets), 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

La versión entregable de «The last ronin» cuenta con 2 enemigos al uso:

Assets de enemigos			
ID	Personaje	Procedencia	Beneficios aportados
#1	Torreta	Adquisición	Estrictamente estéticos
			Notas: Modelo de torreta usada en el juego. Sus beneficios aportados son estéticos, puesto que aunque supuestamente traía algún tipo de funcionalidad implementada, e incluso un sistema para diferenciar entre «bandos» aparentemente no funcionaba (quizá por la versión del motor) más que dar vueltas sobre sí sin parar. Igualmente no se pensaba utilizar ningún código que pudiese traer consigo.
#2	Vigilante	Adquisición	Estrictamente estéticos
			Notas: A pesar de ser modular no se ha hecho uso de esta funcionalidad, al contrario, sólo ha traído contratiempos puesto que su animación ha sido mucho más complicada de llevar a cabo al tener el esqueleto dividido por piezas.

Cuadro B.17: Assets de enemigos

No obstante, si ampliamos el espectro de lo que consideramos enemigos, podríamos incluir:

- Láseres.
- Paredes que nos pueden aplastar.
- Trampas con pinchos.
- Trampas con pinchos a que se activan / desactivan con el tiempo.
- Trampas que empujan al vacío.
- etc.

B.5.3 Animaciones

Las animaciones usadas en el proyecto tienen cinco 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), animaciones del tutorial de escalada que se menciona en la iteración 2 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 Mejoras de personaje / equipo / habilidades

Durante su aventura nuestro personaje podrá encontrar los denominados «orbes dorados» que le dotarán de poder pudiendo realizar dos habilidades especiales: «blink» (o teletransporte) y pasar a tiempo bala, como ya se ha explicado en la sección de mecánicas.

No obstante, no debemos olvidarnos que también existen las «barreras mágicas» que le extraen al personaje sus poderes al pasar por ella así como resetear todos los puzzles de la zona.

B.5.5 Assets de escenarios

Las fuentes principales de los asset de escenarios son las siguientes:

- «**Epic Zen Garden**»: Empleado sobre todo en el tutorial (islas flotantes, cielo, etc).
- «**Infinity Blade: Grass Lands**»: Es de donde provienen la mayoría de assets de los escenarios: columnas, escaleras, estatuas, etc.
- «**Open World Demo Collection**»: Este último usado en mucha menor medida, tan solo con los árboles empleados en el tutorial.

B.5.6 Assets variados

Además de los descritos en las anteriores secciones, se han usado los siguientes packs:

- «**Advanced Cinematic Grass Blueprint**»: Usado para recrear el follaje del escenario.
- «**Infinity Blade: Effects**»: Empleado para los efectos de partículas.
- «**Beams for FPS**»: Empleado únicamente para recrear el láser que nos lanzará el enemigo «vigilante del templo» si nos alcanza.

Y, además, se han usado assets para:

- Katana usada en los coleccionables.
- Estatuillas (los objetos que podemos recoger con nuestro personaje).
- Palancas usadas para activar puertas o cualquier objeto activable.
- Los pequeños mini cristales que rotan en los puntos de guardado.
- Cristal de teletransportación.

B.6 CÁMARAS DISPONIBLES

Las cámaras disponibles son las siguientes:

1. Se persigue que el producto sea amigable con la tecnología de realidad virtual, por ello se apuesta por la cámara en primera persona real (es decir, que use un modelo de personaje completo) como cámara por defecto.
2. Igualmente, se permite al usuario cambiar de cámara, pulsando el botón de cambio de cámara, y pasar a un modo en tercera persona, donde entre otras cosas, se pueden disfrutar de las animaciones del personaje mucho mejor.

B.7 SISTEMA DE GUARDADO

El software guardará nuestros progresos cada vez que avancemos: ya sea llegando a un punto de control o completando un nivel.

Los archivos de guardado se guardan en la siguiente ruta:

«C:/Users/[Usuario]/AppData/Local/TFG/Saved/SaveGames».

En esta ruta encontraremos 3 archivos diferentes, que son los que se describen a continuación:

Archivos de guardado	
Nombre archivo	Descripción
Achievements	Guarda el progreso en los desafíos del juego. No se resetea al iniciar una nueva partida.
Save	Es el principal archivo de guardado. Contiene el progreso como tal, y más específicamente el mapa a cargar, la posición y rotación del personaje y distintas variables que el sistema necesita para realizar la carga del juego.
Settings	Guarda las preferencias del usuario: idioma, resolución, si está a pantalla completa o no, opciones gráficas, etc.

Cuadro B.18: Tabla descripción de archivos de guardado

BIBLIOGRAFÍA

- [1] A comparison of fdd and scrum, 2011. (pages 28, 30).
- [2] Metodología fdd - feature driven development / desarrollo basado en funciones, 2012. (pages 28, 29).
- [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 52).
- [6] U. E. Developer. Fortnite developer fireside on unreal engine 4, 2014. (pages 16, 21, 221).
- [7] DZone. An introduction to feature-driven development, 2009. (page 28).
- [8] ElEconomista. Los videojuegos facturaron más de 1.000 millones, el doble que la industria del cine, 2016. (pages XIII, 6).
- [9] ElMundo. Playstation premia el talento de los videojuegos españoles, 2016. (page 10).
- [10] EpicGames. Animation retargeting (different skeletons), 2017. (pages 79, 80).
- [11] EpicGames. *Documentación Unreal Engine*. Epic Games, 2017. (pages XIV, XVIII, 71, 72, 73).
- [12] EuropaPress. La increíble historia de et, el videojuego que hundió a atari antes de ser enterrado, 2017. (pages XIII, 14).
- [13] Fortune. The 10 most successful states for video game development, 2015. (page 50).

- [14] Gamasutra. Gamasutra salary survey 2014. Technical report, Gamasutra, 2014. (pages XVII, XVII, 49, 50, 52).
- [15] T. P. O. V. Games. Why do achievements, trophies, and badges work?, 2016. (page 12).
- [16] Investing. Datos históricos eur/usd, 2017. (pages XIV, 51).
- [17] MejoresProyectos. El síndrome de los 90, 2007. (pages 44, 45, 220).
- [18] Newzoo. 2017 global mobile market report. Technical report, Newzoo, 2017. (pages XIII, 8).
- [19] Newzoo. Esports revenues will reach 696 million dollars this year and grow to 1.5 billion dollars by 2020. Technical report, Newzoo, 2017. (pages XIII, 7).
- [20] Newzoo. Global games market 2016 report. Technical report, Newzoo, 2017. (pages XIII, 4, 8).
- [21] Numbeo. Cost of living index 2017 mid-year, 2017. (pages XVII, 50).
- [22] PC-Gamer. A game with over 4,000 achievements has just launched on steam, 2017. (page 12).
- [23] SoftZone. Qué es steam direct y en qué se diferencia de steam greenlight, 2017. (page 11).
- [24] Steam. Tasa de steam direct y próximas actualizaciones de la tienda, 2017. (page 11).
- [25] Step-10. Fdd: History, 2014. (page 28).
- [26] Step-10. Fdd: People, 2014. (page 29).
- [27] U-Tad. La industria del videojuego en españa continúa con su imparable crecimiento, 2015. (pages 6, 15).
- [28] Xataka. Las chicas también juegan a videojuegos, y cada vez más, 2014. (page 9).
- [29] ZehnGames. El desarrollo español en steam: Una lectura desde los datos. Technical report, Zehn Games, 2017. (pages XIII, 10).