

The Loudest Sound.

Desarrollo de un videojuego de terror en Unreal Engine 4



Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:

Pablo Martínez Carratalá

Tutor/es:

Carlos José Villagrá Arnedo

Mireia Luisa Sempere Tortosa

Septiembre 2016



Universitat d'Alacant
Universidad de Alicante

Dedicatoria

En estos últimos años he aprendido miles de cosas valiosas, y la mayoría de ellas no están relacionadas con la ingeniería.

Para comenzar, me gustaría darle las gracias a mi familia, pues siempre me ha apoyado en mis decisiones, me ha animado a trabajar en aquello que despertaba mi pasión y nunca se ha dado por vencida a pesar de mis numerosos fallos.

También me gustaría darles las gracias a todos mis compañeros y compañeras, que se han mantenido a mi lado en los peores momentos, me han animado a trabajar duro y me han alegrado todos los días de clase. En especial, me gustaría darles las gracias a Javier, Vicente y Jorge, por aguantarme en los días más negros, a pesar de que yo les aguento más.

Además, les doy las gracias a mis tutores, Carlos y Mireia, por todos los buenos consejos que me han dado a lo largo de la carrera y por ayudarme en el desarrollo de este proyecto.

Este trabajo da fin a una de las mejores etapas de mi vida, donde he tenido la oportunidad de conocer a personas maravillosas que ahora puedo llamar amigos.

Resumen

En los últimos años los videojuegos han evolucionado mucho y han surgido nuevos géneros que se centran más en contar una historia donde el jugador tiene importancia.

Tras las diversas cancelaciones de videojuegos de un nuevo género de terror más orientado a la exploración y a los eventos planeados, se ha creado un mercado expectante de un videojuego con estas características.

En este proyecto se ha desarrollado un videojuego en Unreal Engine 4 con un presupuesto nulo. Se ha perseguido un estilo original y un apartado gráfico, sonoro y jugable a la altura de equipos de desarrollo independiente. **The Loudest Sound**, destaca en su diseño de nivel y en la buena ambientación que posee y es recomendado para un público adulto que no necesita experiencia en el mundo de los videojuegos, gracias a unos controles sencillos e intuitivos.

Índice de contenido

1. Introducción	19
2. Marco teórico o Estado del arte.....	20
2.1. Concepto de Videojuego	20
2.2. Esencia y referencias.....	21
2.2.1. P.T.....	21
2.2.2. Allison Road.....	22
2.2.3. Visage.....	23
2.2.4. Layers of Fear.....	24
2.2.5. Amnesia: The Dark Descent.....	25
2.2.6. SOMA	26
2.2.7. Alien: Isolation.....	26
2.2.8. Outlast	27
3. Objetivos	29
3.1. Objetivo principal del Trabajo de Fin de Grado.....	29
3.2. Desglose de Objetivos.....	29
4. Metodología	30
4.1. Metodología de desarrollo.....	30
4.1.1. El método Kanban	30
4.1.2. ¿Por qué Kanban?.....	30
4.2. Gestión del proyecto y Trello.....	31
4.2.1. Plus for Trello.....	32
4.3. Control de versiones y repositorio	32
5. Cuerpo del trabajo	33
5.1. Análisis, explicación y elección de herramientas.....	33
5.1.1. Motores destinados a la producción de videojuegos	34
5.1.1.1. Los motores en su contexto	34
5.1.1.2. Los motores de videojuegos actuales	35

5.1.1.2.1. Unreal Engine.....	36
5.1.1.2.2. Unity.....	36
5.1.1.2.3. CryEngine.....	37
5.1.1.3. Elección del motor	37
5.2. Documento de Diseño del Videojuego (GDD)	38
5.2.1. El juego en términos generales.....	38
5.2.1.1. Resumen del argumento	38
5.2.1.2. Conjunto de características.....	39
5.2.1.3. Género	39
5.2.1.4. Audiencia	40
5.2.1.4.1. Clasificación PEGI del videojuego	40
5.2.1.5. Apariencia del juego.....	41
5.2.2. Jugabilidad y mecánicas.....	42
5.2.2.1. Jugabilidad	42
5.2.2.1.1. Objetivo del videojuego	42
5.2.2.1.2. Progresión.....	43
5.2.2.1.3. Libertad y estructura de eventos.....	43
5.2.2.1.4. Acciones del personaje	43
5.2.2.1.5. Controles del juego.....	44
5.2.2.2. Mecánicas.....	45
5.2.2.2.1. Movimiento y cámara en primera persona	46
5.2.2.2.2. Interacción con objeto	47
5.2.2.2.3. Abrir y cerrar puertas	48
5.2.2.2.4. Abrir y cerrar cajones o similares.....	49
5.2.2.2.5. Interacción mediante enfoque de cámara	50
5.2.2.2.6. Coger y guardar objetos	51
5.2.2.2.7. Linterna	52
5.2.3. Historia, características y personajes.....	52
5.2.3.1. Historia	53

5.2.3.2. Personajes.....	54
5.2.3.2.1. Dad	54
5.2.3.2.2. Mom	54
5.2.4. Nivel de juego	55
5.2.5. Sistema de música y sonido.....	55
5.2.6. Inteligencia Artificial	56
5.3. Desarrollo e implementación	56
5.3.1. La arquitectura de Unreal Engine 4 y el sistema de scripting visual Blueprints	56
5.3.1.1. Audio y sonido	56
5.3.1.1.1. Sound Cue Editor	57
5.3.1.1.2. Ambient Sound Actor.....	58
5.3.1.1.3. Sound Attenuation.....	58
5.3.1.2. Rendering and Graphics	59
5.3.1.2.1. Lighting Basics.....	60
5.3.1.2.1.1. Static lights.....	60
5.3.1.2.1.2. Stationary lights.....	61
5.3.1.2.1.3. Movable lights.....	62
5.3.1.2.2. Material conceptos básicos.....	62
5.3.1.2.3. PostProcessVolume conceptos básicos	63
5.3.1.3. Conceptos básicos de la arquitectura.....	63
5.3.1.3.1. UObjects y Actors	64
5.3.1.3.2. Gameplay y las clases del Framework	64
5.3.1.3.3. Representando Jugadores, Aliados y Enemigos en el Mundo.....	65
5.3.1.3.3.1. Pawn	65
5.3.1.3.3.2. Character	65
5.3.1.3.4. Controlando Pawns con Player Input o lógica IA	65
5.3.1.3.4.1. Controller	65
5.3.1.3.4.2. PlayerController	65
5.3.1.3.4.3. AIController	65

5.3.1.3.5. Mostrando información a los Jugadores.....	66
5.3.1.3.5.1. HUD	66
5.3.1.3.5.2. Cámara	66
5.3.1.3.6. Estableciendo y Rastreando las reglas del juego.....	66
5.3.1.3.6.1. GameMode	66
5.3.1.3.6.2. GameState	66
5.3.1.3.6.2. PlayerState.....	67
5.3.1.3.7. Las relaciones de las clases del Framework.....	67
5.3.1.4. Scripting visual Blueprints	68
5.3.1.4.1. ¿Cómo funcionan?.....	68
5.3.1.4.2. Tipos de Blueprints	69
5.3.1.4.2.1. Level Blueprint.....	69
5.3.1.4.2.2. Blueprint Class	69
5.3.1.4.2.3. Data-Only Blueprint.....	70
5.3.2. Diseño artístico.....	70
5.3.2.1. Diseño y arte visual	70
5.3.2.1.1. Modelados y optimizaciones	70
5.3.2.1.1.1. Entornos modulares	71
5.3.2.1.1.2. Libros y script FillMyBookshelves	73
5.3.2.1.1.1. Optimizaciones.....	76
5.3.2.1.2. Materiales y texturizado	80
5.3.2.2. Diseño y arte sonoro.....	84
5.3.2.2.1. Sonido llantos fantasma	84
5.3.2.2.2. Sonido puerta evento baño	84
5.3.2.2.3. Sonido de pasos y el sistema de detección de superficie.....	85
5.3.2.2.4. Sonidos ambiente y la atenuación del sonido.....	88
5.3.3. Creación de nivel de juego	89
5.3.3.1. Resumen	89
5.3.3.2. Idea, adquisición de planos y primeros cimientos.....	89

5.3.3.3. Cimientos con entornos modulares	93
5.3.3.4. Cimientos provisionales para el desarrollo	95
5.3.3.5. Nivel final.....	97
5.3.4. Desarrollo de mecánicas de juego	99
5.3.4.1. Blueprint First Person Character	99
5.3.4.1.1. Resumen del funcionamiento de FirstPersonCharacter	102
5.3.4.2. Objetos interactivos.....	102
5.3.4.2.1. Blueprint Interactable Actor.....	103
5.3.4.2.1.1. Resumen del funcionamiento de InteractableActor.....	105
5.3.4.2.2. Blueprint Interactable Pickup.....	106
5.3.4.2.2.1. Resumen del funcionamiento de InteractablePickup.....	107
5.3.4.2.3. Blueprint Interactable Cabinet.....	108
5.3.4.2.3.1. Resumen del funcionamiento de InteractableCabinet	109
5.3.4.2.4. Blueprint InteractablePhysicDoor	110
5.3.4.2.4.1. Resumen del funcionamiento de InteractablePhysicDoor.....	113
5.3.4.3. Sistema de Eventos y control de flujo de juego	113
5.3.4.3.1. Blueprint TriggerOfEvents.....	113
5.3.4.3.1.1. Resumen del funcionamiento de TriggerOfEvents	116
5.3.4.4. Blueprint Enemigo	116
5.3.4.4.1. Resumen del funcionamiento de Enemigo	118
6. Conclusiones	120
7. Anexo: Arte	122
8. Bibliografía	146

Índice de Figuras

Figura 1. Imagen InGame de P.T.	22
Figura 2. Imagen InGame de Allison Road.....	23
Figura 3. Imagen InGame de Visage.....	24
Figura 4. Arte conceptual de Layers of Fear	24
Figura 5. Imagen InGame de Amnesia: The Dark Descent	25
Figura 6. Imagen InGame de SOMA	26
Figura 7. Imagen InGame de Alien: Isolation.....	27
Figura 8. Imagen InGame de Outlast	28
Figura 9. Gráfico de trello personal, utilizado para este proyecto.....	31
Figura 10. Lista de motores clasificados por licencias.....	35
Figura 11. Título y logo del videojuego.....	38
Figura 12. Clasificación PEGI de The Loudest Sound	41
Figura 13. Captura InGame	41
Figura 14. Controles mediante teclado y ratón.....	44
Figura 15. Controles mediante mando de juego.....	44
Figura 16. Captura Ingame desde la cámara del jugador	46
Figura 17. Interacción con un objeto InGame	47
Figura 18. Puerta abierta desde el punto de vista del jugador	48
Figura 19. Interacción con armario de la cocina, se puede observar un cajón ya abierto y otro a punto de ser abierto por el jugador.....	49
Figura 20. Radio que podemos encontrar dentro del videojuego	50
Figura 21. Objeto linterna, siendo interactuado InGame	51
Figura 22. Linterna siendo utilizada dentro de uno de los baños que podemos encontrar en el nivel	52
Figura 23. Enemigo Mom en uno de los pasillos de la casa	54
Figura 24. Ejemplo de Sound Cue	57
Figura 25. Nodos de modificación	57
Figura 26. Icono de Ambient Sound Actor	58
Figura 27. Atenuación del sonido InGame.....	59
Figura 28. Tipos de luces y su aspecto.....	60
Figura 29. Ejemplo de material Madera y sus nodos básicos	63
Figura 30. Relaciones del Framework.....	67
Figura 31. Imagen introductoria a los blueprints.	68
Figura 32. Paredes modulares dentro de 3ds Max.....	71
Figura 33. Unwraps de las paredes dentro de 3ds Max.....	72

Figura 34. Pared "Esquina Interior" y su colisión en color azul.	73
Figura 35. Modelado y texturizado de los 22 libros.....	74
Figura 36. Ejemplo libros colocados en estantería número 1.....	75
Figura 37. Ejemplo libros colocados en estantería número 2.....	75
Figura 38. Modelado gratuito (candelabro) antes de ser optimizado.	76
Figura 39. Modelado gratuito (candelabro) tras ser optimizado.	77
Figura 40. Tabla de optimización del modelo "candelabro".	77
Figura 41. Modelado gratuito (cortinas) antes de ser optimizado número 1.....	78
Figura 42. Modelado gratuito (cortinas) antes de ser optimizado número 2.....	78
Figura 43. Modelado gratuito (cortinas) tras ser optimizado número 1.....	79
Figura 44. Modelado gratuito (cortinas) tras ser optimizado número 2.	79
Figura 45. Tabla de optimización del modelo "cortinas".	80
Figura 46. Ejemplo mapa difuso.	80
Figura 47. Ejemplo mapa de normales.....	81
Figura 48. Ejemplo mapa de oclusión.....	81
Figura 49. Ejemplo mapa especular.	81
Figura 50. Ejemplo del texturizado de un libro.....	82
Figura 51. Libro con textura aplicada.	83
Figura 52. Ejemplo del resultado final de los materiales con los diferentes mapas de texturas.....	83
Figura 53. Opciones de Unreal sobre las Physical Surfaces.	85
Figura 54. Blueprint de la función <code>Footstep</code> , de la clase <code>FirstPersonCharacter</code>	86
Figura 55. Sound Cue de los pasos en alfombra.	86
Figura 56. Sound Cue de los pasos en madera.	87
Figura 57. Captura InGame de la cocina, donde escuchamos Ambient Sounds como la nevera o la luz del techo.	88
Figura 58. Planta baja, plano real.....	90
Figura 59. Planta baja, Unreal.....	91
Figura 60. Primera planta, plano real	92
Figura 61. Primera planta, Unreal	93
Figura 62. Parte del nivel en construcción, vista sin iluminación	94
Figura 63. Parte del nivel aún en construcción con algunas pruebas de iluminación y texturizado	95
Figura 64. Escaleras en construcción	96
Figura 65. Pruebas de iluminación en el pasillo de la primera planta.....	97
Figura 66. Comedor, nivel final InGame.	98
Figura 67. Baño, nivel final InGame.....	98
Figura 68. Componentes del blueprint <code>FirstPersonCharacter</code> listados.	99
Figura 69. Componentes del blueprint <code>FirstPersonCharacter</code> vistos desde el ViewPort.	100

Figura 70. Tabla de variables del blueprint <code>FirstPersonCharacter</code>	101
Figura 71. Tabla de funciones del blueprint <code>FirstPersonCharacter</code>	102
Figura 72. Interfaz de interacción con objetos.	102
Figura 73. Interacción con objetos InGame.	103
Figura 74. Componentes del blueprint <code>InteractableActor</code> listados y vistos desde el ViewPort.....	104
Figura 75. Tabla de variables del blueprint <code>InteractableActor</code>	105
Figura 76. Tabla de funciones del blueprint <code>InteractableActor</code>	105
Figura 77. Tabla de variables del blueprint <code>InteractablePickup</code>	107
Figura 78. Tabla de funciones del blueprint <code>InteractablePickup</code>	107
Figura 79. Componentes del blueprint <code>InteractableCabinet</code> listados y vistos desde el ViewPort. .	108
Figura 80. Tabla de variables del blueprint <code>InteractableCabinet</code>	109
Figura 81. Tabla de funciones del blueprint <code>InteractableCabinet</code>	109
Figura 82. Componentes del blueprint <code>InteractablePhysicDoor</code> listados.	110
Figura 83. Componentes del blueprint <code>InteractablePhysicDoor</code> vistos desde el ViewPort.	111
Figura 84. Tabla de variables del blueprint <code>InteractablePhysicDoor</code>	112
Figura 85. Tabla de funciones del blueprint <code>InteractablePhysicDoor</code>	112
Figura 86. Componentes del blueprint <code>TriggerOfEvents</code> vistos desde el ViewPort.....	114
Figura 87. Tabla de variables del blueprint <code>TriggerOfEvents</code>	115
Figura 88. Tabla de funciones del blueprint <code>TriggerOfEvents</code>	115
Figura 89. Componentes del blueprint <code>Enemigo</code> vistos desde el ViewPort.	116
Figura 90. Componentes del blueprint <code>Enemigo</code> listados.....	117
Figura 91. NavMesh del nivel de juego.	117
Figura 92. Tabla de variables del blueprint <code>Enemigo</code>	118
Figura 93. Tabla de funciones del blueprint <code>Enemigo</code>	118
Figura 94. Captura de The Loudest Sound InGame número 1.....	122
Figura 95. Captura de The Loudest Sound InGame número 2.....	123
Figura 96. Captura de The Loudest Sound InGame número 3.....	123
Figura 97. Captura de The Loudest Sound InGame número 4.....	124
Figura 98. Captura de The Loudest Sound InGame número 5.....	124
Figura 99. Captura de The Loudest Sound InGame número 6.....	125
Figura 100. Captura de The Loudest Sound InGame número 7.....	125
Figura 101. Captura de The Loudest Sound InGame número 8.....	126
Figura 102. Captura de The Loudest Sound InGame número 9.....	126
Figura 103. Captura de The Loudest Sound InGame número 10.....	127
Figura 104. Captura de The Loudest Sound InGame número 11.....	127
Figura 105. Captura de The Loudest Sound InGame número 12.....	128
Figura 106. Captura de The Loudest Sound InGame número 13.....	128

Figura 107. Captura de The Loudest Sound InGame número 14.....	129
Figura 108. Captura de The Loudest Sound InGame número 15.....	129
Figura 109. Captura de The Loudest Sound InGame número 16.....	130
Figura 110. Captura de The Loudest Sound InGame número 17.....	130
Figura 111. Captura de The Loudest Sound InGame número 18.....	131
Figura 112. Captura de The Loudest Sound InGame número 19.....	131
Figura 113. Captura de The Loudest Sound InGame número 20.....	132
Figura 114. Captura de The Loudest Sound InGame número 21.....	132
Figura 115. Captura de The Loudest Sound InGame número 22.....	133
Figura 116. Captura de The Loudest Sound InGame número 23.....	133
Figura 117. Captura de The Loudest Sound InGame número 24.....	134
Figura 118. Captura de The Loudest Sound InGame número 25.....	134
Figura 119. Captura de The Loudest Sound InGame número 26.....	135
Figura 120. Captura de The Loudest Sound InGame número 27.....	135
Figura 121. Captura de The Loudest Sound InGame número 28.....	136
Figura 122. Captura de The Loudest Sound InGame número 29.....	136
Figura 123. Captura de The Loudest Sound InGame número 30.....	137
Figura 124. Captura de The Loudest Sound InGame número 31.....	137
Figura 125. Captura de The Loudest Sound InGame número 32.....	138
Figura 126. Captura de The Loudest Sound InGame número 33.....	138
Figura 127. Captura de The Loudest Sound InGame número 34.....	139
Figura 128. Captura de The Loudest Sound InGame número 35.....	139
Figura 129. Captura de The Loudest Sound InGame número 36.....	140
Figura 130. Captura de The Loudest Sound InGame número 37.....	140
Figura 131. Captura de The Loudest Sound InGame número 38.....	141
Figura 132. Captura de The Loudest Sound InGame número 39.....	141
Figura 133. Captura de The Loudest Sound InGame número 40.....	142
Figura 134. Captura de The Loudest Sound InGame número 41.....	142
Figura 135. Captura de The Loudest Sound InGame número 42.....	143
Figura 136. Captura de The Loudest Sound InGame número 43.....	143
Figura 137. Captura de The Loudest Sound InGame número 44.....	144
Figura 138. Captura de The Loudest Sound InGame número 45.....	144

1. Introducción

Actualmente el mundo de los videojuegos está en uno de los mejores momentos. La sociedad está comenzando a aceptarlos dentro de su vida diaria y ya no se le considera raro a aquellos que encuentran diversión y entretenimiento en este mundo. Un buen ejemplo es la reciente salida al mercado de PokemonGO, un juego para smartphones que ha conseguido que millones de personas que no habían probado un videojuego nunca salgan a la calle a cazar pokemons. Lejos de haber alcanzado su madurez creativa, los videojuegos siguen siendo una nueva forma de arte que parece estar dando aún, 50 años después de su aparición, sus primeros pasos.

Con el paso del tiempo, los videojuegos han madurado y han surgido nuevos géneros orientados mayormente al público adulto, que se centra más en contar una historia que en el aspecto arcade de los videojuegos clásicos. Con este proceso de maduración surgió un nuevo género de terror que se centraba más en la exploración del escenario y en la sucesión de eventos programados.

The Loudest Sound se incluye dentro de éste nuevo género de terror, donde la ambientación y el diseño de nivel son los factores más importantes.

El documento está estructurado de una forma concreta para facilitar su lectura y comprensión.

Primeramente encontramos el *Marco teórico*, donde se analiza el concepto de videojuego y los juegos que han servido de referencia e inspiración para este proyecto.

A continuación encontraremos el apartado de *Objetivos*, donde se describen detalladamente los objetivos que se pretenden conseguir con este trabajo.

Posteriormente, el apartado de *Metodología* explica los métodos y herramientas que se han utilizado para gestionar el proyecto de una manera útil y sencilla.

En el apartado *Cuerpo del trabajo* podemos encontrar un análisis de los motores de videojuegos más recientes, además del documento de diseño (GDD) del videojuego y todo lo relacionado con su desarrollo e implementación, tanto en su apartado de mecánicas como en su aspecto de diseño gráfico y sonoro.

Después en la sección de *Conclusiones* aparecen mis opiniones acerca del desarrollo del proyecto, así como todo lo aprendido en su realización.

Por último, en el *Anexo: Arte* encontramos una gran cantidad de imágenes del resultado final del videojuego.

2. Marco teórico o Estado del arte

En este bloque se va a explicar el concepto básico de videojuego puesto que es necesario para comprender aquello que se pretende realizar. Además, se van a analizar múltiples videojuegos del género de terror, algunos de ellos creados por grandes empresas con calidad triple A, y otros creados por estudios independientes con un presupuesto mucho menor.

Podremos observar que muchos de estos videojuegos aún no están terminados o incluso han sido cancelados antes de su lanzamiento. Aun así, se puede observar un potencial enorme que aún no ha sido explotado en este género de videojuegos, con un mercado increíblemente grande.

2.1. Concepto de Videojuego

Para comprender correctamente que se está desarrollando, primero debemos comprender que es un videojuego, pues su definición ha cambiado inmensamente con la aparición de nuevas ideas por parte de los diseñadores de videojuegos.

La definición básica que podemos encontrar en multitud de webs como Wikipedia, Wikia, la propia RAE o páginas especializadas es la siguiente:

“Un videojuego o juego de vídeo es un software creado para el entretenimiento en general y basado en la interacción entre una o varias personas y un aparato electrónico que ejecuta dicho videojuego; este dispositivo electrónico puede ser una computadora, una máquina arcade, una videoconsola, un dispositivo *handheld* (un teléfono móvil, por ejemplo) los cuales son conocidos como “plataformas”.”

Hay videojuegos sencillos y otros más complejos, algunos son capaces de narrar historias y acontecimientos usando audio y video creados ex profeso, demostrando que el videojuego es otra manifestación del arte. Cuando los videojuegos comenzaron a estar disponibles al gran público estaban muy centrados en el género Arcade pero con el paso del tiempo se les encontró una utilidad mucho mayor, la capacidad de contar historias, y al igual que un buen libro o una buena película, un buen videojuego lograba influir en las emociones de los jugadores.

2.2. Esencia y referencias

El videojuego tendrá su propia esencia, pero para su realización voy a fijarme en algunos de los juegos más exitosos de este género. Algunos de estos juegos son antiguos y otros ni siquiera han llegado a salir a la venta, pero eso sí, todas estas obras han despertado el interés en su audiencia.

De estos videojuegos no sólo se ha estudiado las mecánicas de *gameplay*, sino que también se le ha dado importancia al arte y a la psicología dentro del videojuego que consigue asustar a los jugadores.

Los principales juegos que se han estudiado y que se van a comentar brevemente a continuación son:

- P.T.
- Allison Road
- Visage
- Layers of Fear
- Amnesia: The Dark Descent
- SOMA
- Alien: Isolation
- Outlast

2.2.1. P.T.

P.T. (siglas para “*Playable Teaser*”), fue un tráiler jugable para el videojuego *Silent Hills*. *P.T.* es un videojuego en primera persona de terror psicológico, que tiene como director a Hideo Kojima en colaboración con Guillermo Del Toro. *P.T.* fue cancelado antes de su salida, pero aun así, recibió grandes ovaciones por su dirección, apartado visual, complejidad de la historia y tensión sobrenatural. De igual manera, los puzzles del videojuego fueron aclamados por unos y criticados por otros.

Este ha sido la **principal fuente de inspiración del videojuego**, de hecho en el videojuego final se incluye un *Easter Egg*, o huevo de Pascua, que hace referencia a una de las escenas de este videojuego.



Figura 1. Imagen InGame de P.T.

P.T. utiliza una cámara en primera persona, siendo distinto a la cámara en tercera persona de los anteriores juegos de *Silent Hill*. Se centra en un protagonista desconocido que el jugador controla desde que se despierta en una casa encantada, donde ocurren experiencias paranormales. Las únicas áreas que se pueden explorar en la casa son un pasillo en forma de L y dos habitaciones adyacentes al mismo: un baño y una escalera que lleva a la habitación en la que el jugador comienza un bucle, una continua reaparición del mismo pasillo. Las únicas acciones que puede realizar el jugador es andar y hacer zoom. Para progresar, el jugador debe investigar eventos terroríficos y resolver puzzles crípticos. Cada vez que un bucle se completa satisfactoriamente, aparecen cambios en el pasillo. Además, el jugador puede encontrarse con un fantasma hostil llamado Lisa. Si ésta atrapa al protagonista, se le da un susto al jugador y vuelve a comenzar el bucle en el que se encontraba.

2.2.2. Allison Road

Allison Road es un videojuego de terror en primera persona realizado por Lilith Ltd y formalmente publicado por Team17. Era considerado el sucesor espiritual de *P.T.* hasta que fue cancelado en junio de 2016.

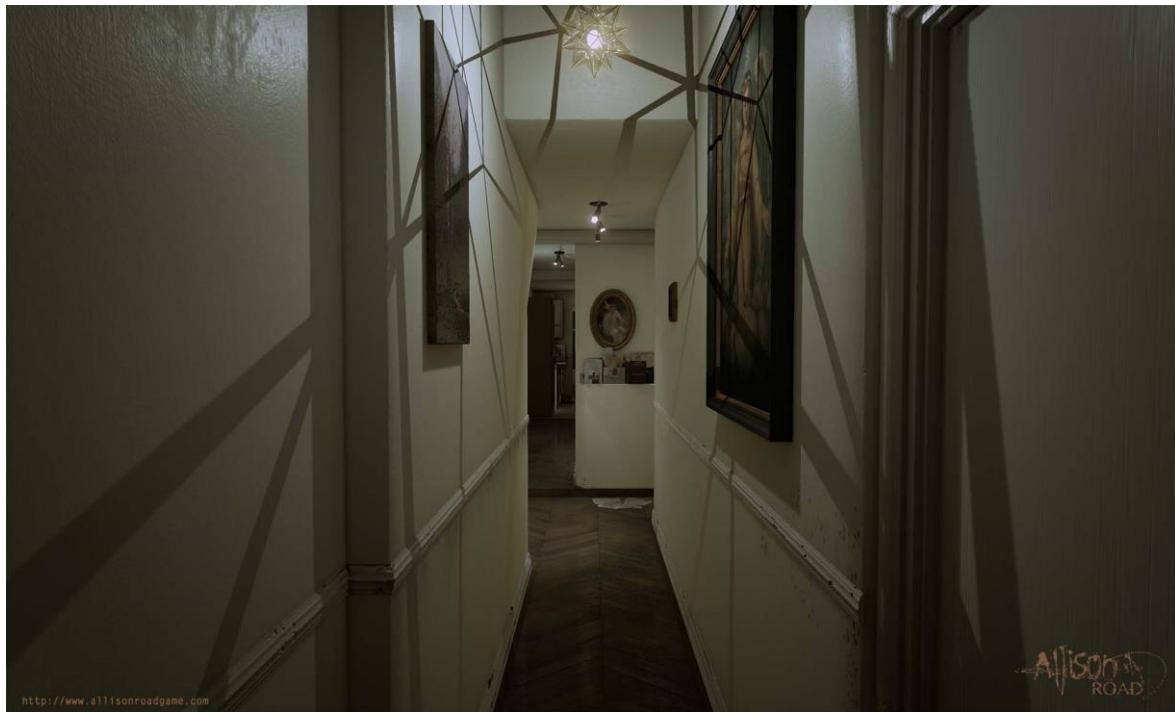


Figura 2. Imagen InGame de Allison Road

Allison Road ha sido la **segunda fuente de inspiración principal** para **The Loudest Sound**, incluyendo de nuevo algunos Easter Eggs relacionados con este videojuego.

El *gameplay* de *Allison Road* es similar al de *P.T.*, presentándonos una historia mínima que el jugador debe resolver. En este videojuego aparte de caminar y hacer zoom, también contamos con las mecánicas de interactuar con puertas y objetos.

2.2.3. Visage

Visage es un videojuego de terror desarrollado por un equipo independiente llamado SadSquare Studio. De forma similar a *Allison Road*, el videojuego es un sucesor a *P.T.* El videojuego actualmente está en campañas de Kickstarter y Steam Greenlight.



Figura 3. Imagen InGame de Visage

Este juego aún se encuentra en desarrollo y por ello no se sabe mucho de él.

2.2.4. Layers of Fear

Layers of Fear es un videojuego de terror psicológico desarrollado y publicado por Bloober Team.



Figura 4. Arte conceptual de *Layers of Fear*

En *Layers of Fear* el jugador controla a un pintor psicológicamente perturbado intentando completar su obra maestra, mientras se mueve por una mansión Victoriana, encontrando secretos perturbadores sobre el pintor. El *gameplay*, presentado principalmente por una perspectiva en primera persona, tiene una dirección altamente marcada por la historia y se centra en la resolución de puzzles y la exploración, pudiendo encontrar multitud de sustos en los niveles.

2.2.5. Amnesia: The Dark Descent

Amnesia: The Dark Descent es un juego de terror para PC desarrollado por Frictional Games, que desarrolló anteriormente la serie *Penumbra*. De manera similar a los anteriores juegos desarrollados por la compañía, *Amnesia* es un juego de aventura basado en la exploración y con una perspectiva en primera persona. El jugador no tiene acceso a armas, y debe utilizar su ingenio para evitar las truculentas criaturas que habitan el Castillo de Brennenburg o bien esconderse de ellos.



Figura 5. Imagen InGame de Amnesia: The Dark Descent

El juego además mantiene la interacción física con objetos utilizada en *Penumbra*, permitiendo puzzles físicos avanzados como apertura de puertas y arreglos de maquinaria.

2.2.6. SOMA

SOMA es un videojuego de aventura, ciencia ficción, terror y supervivencia. Tiene lugar en una instalación remota que se encuentra bajo el agua, donde las máquinas comienzan a tener características humanas.



Figura 6. Imagen InGame de SOMA

El videojuego se juega en una perspectiva en primera persona. El jugador encontrará varias criaturas hostiles que intentarán matarlo. *SOMA* principalmente utiliza elementos de terror psicológico en lugar de sustos convencionales como en la mayoría de los videojuegos del género.

2.2.7. Alien: Isolation

Alien: Isolation es un videojuego de terror, supervivencia y sigilo en primera persona, desarrollado por The Creative Assembly y publicado por Sega. Forma parte de la franquicia de las películas Alien.



Figura 7. Imagen InGame de *Alien: Isolation*

A diferencia de la mayoría de los demás juegos de la franquicia, en *Alien: Isolation* solo hay un xenomorfo en la mayor parte del juego, y este no puede ser matado, haciendo que el jugador deba utilizar tácticas de sigilo para sobrevivir. Aunque el juego cuenta con algunas armas, estas son sólo letales contra humanos y androides. En lugar de seguir un camino predeterminado, la inteligencia artificial del xenomorfo ha sido programada para cazar activamente al jugador utilizando su vista, audición y olor. La inteligencia artificial del alien fue programada con una compleja serie de diseños de comportamiento, que se liberan progresivamente a medida que encuentra al jugador para crear la ilusión de que el alien está aprendiendo de sus encuentros con él/ella, y hace ajustes a su estrategia de caza según dichos encuentros.

2.2.8. Outlast

Outlast es un videojuego de terror psicológico desarrollado y publicado por Red Barrels Games. *Outlast* se juega desde una perspectiva en primera persona. El protagonista, a diferencia de muchos protagonistas de juegos de terror de supervivencia, es incapaz de combatir, en su lugar el personaje es capaz de desplazarse, escalar u ocultarse en lugares de su entorno para eludir a los enemigos.



Figura 8. Imagen InGame de Outlast

Para desplazarse por el entorno, el personaje es capaz de escalar lugares altos, evitar obstáculos, arrastrarse, y deslizarse en espacios estrechos usando *parkour*. La resistencia del personaje es limitada y se puede recuperar progresivamente de las heridas si permanece un determinado tiempo sin ser atacado.

3. Objetivos

3.1. Objetivo principal del Trabajo de Fin de Grado

El objetivo de este proyecto es la creación de un videojuego 3D dentro del género de terror para PC. Se construirá con el motor de videojuegos Unreal Engine 4, y la programación completa de las mecánicas se llevará a cabo mediante el sistema de Scripting Visual Blueprints.

También se realizará un diseño previo del videojuego con un documento GDD, se analizarán juegos similares para conseguir un producto competente dentro del mercado actual, se analizarán las características que nos ofrece el motor Unreal Engine 4 y se aprovecharán al máximo, se crearán recursos para el mismo a partir de creaciones propias y de modificaciones de recursos obtenidos de manera gratuita con licencia Creative Commons.

3.2. Desglose de Objetivos

A continuación se muestran los objetivos del trabajo divididos en tareas más específicas:

1. Realizar el GDD (*Game Design Document*) de un videojuego.
2. Crear un videojuego que pueda ser jugado en otros ordenadores fácilmente.
3. Sonorizar un videojuego.
4. Crear modelados para un videojuego.
5. Texturizar con gran calidad escenarios y objetos de manera realista.
6. Crear un nivel realista y una ambientación extraordinaria con las herramientas que provee Unreal.
7. Importar y utilizar correctamente *Assets* creados mediante programas de terceros.
8. Implementar las mecánicas utilizando únicamente el sistema de Scripting Visual Blueprints.
9. Crear un videojuego y unos recursos fácilmente ampliables para una futura continuación del desarrollo.
10. Compatibilizar el videojuego con el dispositivo de realidad virtual Oculus Rift.
11. Presupuesto 0 €

4. Metodología

En este bloque se abordan los aspectos de Gestión de Proyectos y Metodología de Desarrollo. También se va a hacer un pequeño inciso sobre el control de versiones y repositorio de este proyecto.

4.1. Metodología de desarrollo

4.1.1. El método Kanban

En el desarrollo de software, utilizamos un sistema Kanban virtual para limitar el trabajo en curso. A pesar de que el nombre se origina del idioma japonés "Kanban", y se traduce aproximadamente como "tarjeta de señal", y hay tarjetas utilizadas en la mayoría de las implementaciones de Kanban en desarrollo de software, estas tarjetas no funcionan en realidad como señales para realizar más trabajo. Representan los elementos de trabajo. De ahí el término "virtual" porque no existe una tarjeta física.

El método Kanban formulado por David J. Anderson es una aproximación al proceso gradual, evolutivo y al cambio de sistemas para las organizaciones. Utiliza un sistema de extracción limitada del trabajo en curso como mecanismo básico para exponer los problemas de funcionamiento del sistema (o proceso) y estimular la colaboración para la mejora continua del sistema. Un ejemplo del sistema de extracción es el sistema Kanban, y es después de esta popular forma de trabajo en curso, que se ha considerado método.

4.1.2. ¿Por qué Kanban?

La metodología de desarrollo elegida para este proyecto ha sido la metodología ágil Kanban. Se ha optado por elegir esta metodología ya que se había trabajado con ella anteriormente y funciona muy bien para equipos pequeños. Además, dado que el desarrollo completo lo ha realizado una sola persona, el hecho de utilizar tarjetas con distintas finalidades, sin restricciones, poco estrictas y muy distintas entre sí, me permitía no acabar extremadamente cansado y poder dedicarle más tiempo diario.

4.2. Gestión del proyecto y Trello

El proyecto completo se ha gestionado mediante la herramienta gratuita Trello. Trello es un gestor de tareas que permite el trabajo de forma colaborativa mediante tableros (*board*) compuestos de columnas (llamadas listas) que representan distintos estados. Se basa en el método Kanban para gestión de proyectos, con tarjetas que viajan por diferentes listas en función de su estado.

Para comenzar se crearon y agregaron tareas seguras al *Backlog* (lista de tareas del proyecto) que posteriormente se irían cambiando a la lista de *Doing*, donde encontramos las tareas que están en progreso. Una vez se termina una tarea, abandona la lista *Doing* y se agrega a la lista *Testing*, donde se comprobarán que están terminadas y funcionan correctamente. Una vez que se ha producido esto, las tareas son movidas a *Done*, donde podemos encontrar todas las tareas ya terminadas.

Además de todo esto, durante el desarrollo surgen nuevas tareas que serán agregadas al *Backlog*, tareas que no estaban previstas al comienzo del proyecto pero necesarias actualmente.

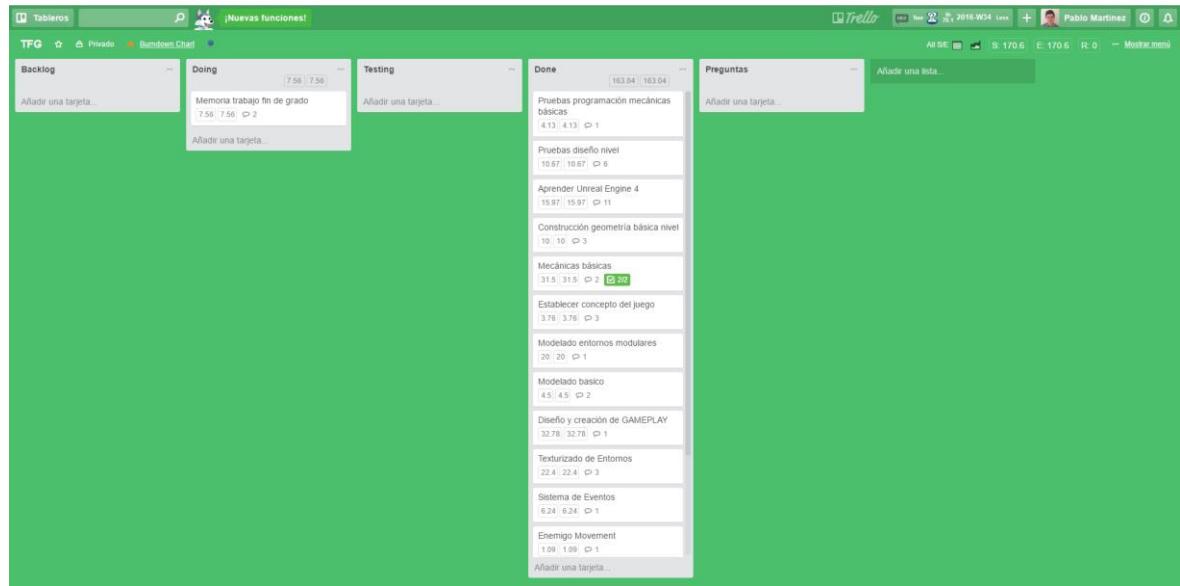


Figura 9. Gráfico de trello personal, utilizado para este proyecto

4.2.1. Plus for Trello

Además de la herramienta Trello, que se encuentra en nuestro navegador, se ha utilizado la herramienta Plus for Trello, una extensión del navegador que amplía la funcionalidad de Trello. Su característica básica y la que más uso ha tenido en este proyecto es la de contar el tiempo de trabajo.

Cuando comenzamos a trabajar debemos buscar la tarjeta en la que estamos trabajando y activar un contador. Cuando terminemos de trabajar deberemos parar este contador y se habrá guardado el tiempo que hemos estado trabajando en esa tarjeta.

Esta herramienta es muy útil para ponernos *deadlines* (fechas límite) y no dedicarle un tiempo excesivo a una tarea. Esto nos ayuda a terminar el proyecto a tiempo y a no quedarnos atascados en mitad del mismo.

4.3. Control de versiones y repositorio

Para este proyecto no se ha utilizado un control de versiones ni un repositorio por diversos motivos.

Este proyecto se basa en la realización de un videojuego de manera individual y ese posiblemente es el mayor motivo por el que no se ha utilizado, aun así, existen multitud de razones. El videojuego se ha desarrollado en Unreal Engine 4, los proyectos realizados con este motor pesan mucho y pueden surgir problemas si aparecen conflictos entre archivos con extensiones específicas de Unreal. Además, aunque no se haya utilizado control de versiones propiamente dicho, sí que se ha trabajado de una manera bastante segura para no perder información y trabajo ya realizado.

Unreal ya nos provee con una función de autoguardado por si surge algún problema, aun así, se han ido realizado múltiples clones del proyecto original para siempre poseer una versión anterior de repuesto por si sucediera algo inesperado. También se han ido subiendo a Google Drive múltiples versiones comprimidas del proyecto para más seguridad.

5. Cuerpo del trabajo

En este bloque se pretende describir con detalle el proceso completo de desarrollo del videojuego. Podemos encontrar tres partes diferenciadas dentro del bloque:

1. Una primera sección ha sido dedicada al análisis, explicación y elección de herramientas puesto que es una parte tan importante como el desarrollo mismo.
2. Una segunda sección ha sido dedicada a la elaboración del ***Game Design Document*** (GDD), donde se muestran todos los aspectos de diseño a tener en cuenta a la hora de crear un videojuego.
3. Una tercera y última sección ha sido dedicada puramente al **desarrollo e implementación** del videojuego. En esta sección no solo se incluye el desarrollo de Unreal, sino que también encontraremos apartados refiriéndose a la creación de modelados 3D como de sonidos.

5.1. Análisis, explicación y elección de herramientas

En este bloque se van a analizar los distintos motores de videojuegos más utilizados en la actualidad, dentro del contexto de los motores de videojuegos disponibles a la mayoría de desarrolladores, sin tener en cuenta los motores privados utilizados por grandes empresas como Electronic Arts o Blizzard.

Es necesario destacar que con Unreal Engine 4 se ha construido el videojuego, pero se han utilizado una gran cantidad de herramientas adicionales de edición de imágenes, creación y edición de modelados 3D, edición de audio, etc. para la creación de **Assets** (recursos), que a través del motor podrán ser utilizados en nuestro videojuego. Es por ello que también encontramos secciones reservadas a estos mismos.

5.1.1. Motores destinados a la producción de videojuegos

5.1.1.1. Los motores en su contexto

Antes de comenzar, cabe definir distintos conceptos importantes.

Framework: Un *framework* (estructura) define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. De igual manera, en el **desarrollo de software**, un framework o **infraestructura digital**, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Game Engine: Un *Game Engine* (motor de videojuegos) es un **framework** (estructura) diseñado para la creación y desarrollo de videojuegos. Los desarrolladores los utilizan para crear videojuegos para consolas, dispositivos móviles y ordenadores personales. La funcionalidad base proveída por los motores suele incluir: **motor de renderizado** para gráficos 2D o 3D, **motor de físicas**, detección de colisiones, **motor de sonido**, *scripting* (programación), animación, inteligencia artificial, *networking* (motor de red), administrador de memoria, *threading* (administración de procesos), etc.

5.1.1.2. Los motores de videojuegos actuales

Actualmente hay una cantidad inmensa de motores con los que los desarrolladores pueden crear su videojuego. Algunos de ellos son privados, mientras que otros están disponibles de forma gratuita o mediante contratos. Nombrarlos todos no es tarea fácil, pero Wikipedia nos muestra una lista con algunos de los más conocidos clasificados por licencias.

V · T · E	Game engines (list)		[hide]
Source port · First-person shooter engine (list) · Tile engine · Game engine recreation (list) · Game creation system			
Free and open-source	2D	Adventure Game Studio · Beats of Rage · Cocos2d · Flixel · Jogre · KiriKiri · libGDX · Moai · OpenFL · ORX · Pygame · Ren'Py · Stratagus · Thousand Parsec · VASSAL · Xconq	
	2.5D	Aleph One · Cube	
	3D	Away3D · Blender Game · Cafu · Crystal Space · Cube 2 · Delta3D · Dim3 · Goo Create · GLScene · Horde3D · HPL 1 · Irrlicht · id Tech 3 · id Tech 4 · JMonkey · OGRE · Open Wonderland · Panda3D · Papervision3D · Platinum Arts Sandbox Free 3D Game Maker · PlayCanvas · PLIB · Quake · Quake II · RealmForge · Retribution · Torque 3D	
	Mix	Allegro · Construct Classic · Godot · Lightweight Java Game Library · Spring · Wintermute Engine	
Proprietary	2D	Construct 2 · Corona · Clickteam Fusion · GameMaker: Studio · GameSalad · M.U.G.E.N · NScripter · RPG Maker · Southpaw · Stencyl · UbiArt Framework · Vicious · Virtual Theatre · V-Play · Zillions of Games	
	3D	4A · Amazon Lumberyard · Anvil · Bork3D · C4 · Chrome · Clausewitz Engine · Creation · CryEngine · Crystal Tools · Diesel · Dunia · EAGL · EGO · Elflight · Enforce · Enigma · Essence · Flare3D · Fox · Frostbite · Geo-Mod · GoldSrc · HeroEngine · HydroEngine · HPL 2-3 · id Tech 5 · id Tech 6 · Ignite · IW · Jade · Kinética · LS3D · Leadwerks · LithTech · Luminous Studio · LyN · Marmalade · Mizuchi · MT Framework · Outerra · Panta Rhei · PhyreEngine · Q · Real Virtuality · REDengine · Refractor · Riot · RAGE · SAGE · Serious · Shark 3D · ShiVa · Silent Storm · Source · Titan · TOSHI · Truevision3D · Unigine · Unity · Unreal · Vision · Visual3D · XnGine · X-Ray · YETI · Zero	
	Mix	Gamebryo · Hybrid Graphics · Kaneva Game Platform · Metismo	
Historical interest	BRender · Build · Dark · Doom · Game-Maker · GameMaker · Garry Kitchen's GameMaker · Genesis3D · Genie · Filmation · Freescape · INSANE · Jedi · MADE · Pie in the Sky · RenderWare · Sim RPG Maker · Sith · Voxel Space · Wolfenstein 3D		
Proprietary middleware (list)	Euphoria · Gameware · GameWorks · Havok · iMUSE · Kynapse · Quazal · SpeedTree · Xaitment · FaceGen		

Figura 10. Lista de motores clasificados por licencias.

Nosotros nos vamos a centrar en los más utilizados por desarrolladores actuales, sobre todo *indies* (independientes), puesto que pertenecemos a ese conjunto. Estos motores son: Unreal Engine 4, Unity y CryEngine.

5.1.1.2.1. Unreal Engine

Unreal Engine 4 (UE4) es el nuevo motor liberado de Epic Games abierto al gran público recientemente, y es el sucesor de UDK. *UE4* tiene algunas capacidades gráficas increíbles, incluyendo cosas como capacidades avanzadas de iluminación dinámica y un nuevo sistema de partículas que puede manejar hasta un millón de partículas en una escena a la vez. Cosa que a cualquier artista 3D o de juego le hará caer la baba, su motor gráfico es el más potente de la comparativa.

Aunque el *Unreal Engine 4* es el sucesor del UDK, es importante tener en cuenta que ha habido algunos cambios muy drásticos en su motor. Estos cambios no son malos; sin embargo, la facilidad de uso de *UE4* respecto a UDK hace que sea mucho más atractivo para los nuevos desarrolladores de juegos. De todos modos *UE4* no es un software sencillo dado sus múltiples posibilidades de desarrollo en entornos profesionales y la potencia de su motor gráfico.

Un cambio notable ha sido en el lenguaje de scripting para *UE4*. Como es sabido, Unreal Engine siempre ha utilizado UnrealScript. En esta versión, UnrealScript ha sido completamente sustituido por C++, y Kismet ha sido sustituido por el sistema Blueprint más intuitivo.



5.1.1.2.2. Unity

Unity es un motor gráfico históricamente asociado a los juegos para dispositivos móviles, pero el lanzamiento de *Unity 5* y su nuevo sistema de renderizado suponen nuevas y potentes capacidades para crear juegos con gráficos realistas, que ofrezcan una gran inmersión al jugador para dispositivos cada vez más potentes.



Es evidente que en 2015-16 estamos entrando, por fin, en la guerra *next-gen* con *Unity* intentando enfrentarse cara a cara a los motores gráficos más potentes como CryENGINE y Unreal Engine 4.

Una de las claves de *Unity* es que el juego puede ser portado a 21 plataformas diferentes *empleando* un solo código, lo cual facilita mucho a los desarrolladores crear una versión de su juego para varias plataformas.

5.1.1.2.3. CryEngine

CryEngine es un motor gráfico extremadamente potente, desarrollado por Crytek e introducido con su primer Far Cry. El motor está diseñado para usarse en juegos de PC y consolas, incluyendo PlayStation 4 y Xbox One. Las capacidades y potencia de este motor sobrepasan de largo a motores como Unity, tan solo Unreal Engine 4 puede competir en aspectos como las físicas, los sistemas de animación de modelos y la capacidad de iluminar en tiempo real las escenas. *CryEngine* es un motor gráfico fantástico, pero tiene una gran curva de aprendizaje, por lo que tendrás que invertir bastantes horas ante de comenzar a ser productivo con él.



CRYENGINE®

5.1.1.3. Elección del motor

La elección del motor es una de las decisiones más importantes a la hora de desarrollar un videojuego. Influye en todos sus aspectos, *Look&Feel*, posibilidades de mecánicas, dificultades técnicas, iluminación, tiempo de aprendizaje, tiempo de desarrollo, etc.

Como desarrollador independiente y estudiante, he basado mi decisión en distintos aspectos:

- Precio
- Documentación
- Comunidad de usuarios activa
- Utilidades distintas que aprender

Tras estudiar detenidamente todas mis opciones se ha optado por utilizar Unreal Engine 4 por ser el que más se adapta al proyecto en mente. Unreal Engine 4 junto con todas sus herramientas es 100% gratuito, tan sólo debemos pagar un 5% de nuestros beneficios al vender un videojuego hecho con su motor si superamos una cifra concreta. Además, Unreal Engine 4 tenía una gran fama de tener una muy buena y grande comunidad de desarrolladores a sus espaldas, cosa que he podido confirmar

mientras desarrollaba el videojuego. He podido preguntar en los foros de Unreal de manera muy sencilla y a las pocas horas recibí la ayuda de uno de los propios miembros de la compañía. Además, me puse en contacto con múltiples personas por Twitter y por Youtube, personas que se han interesado en el proyecto y han ofrecido su ayuda.

Unreal Engine 4 también contaba con una característica la cual tenía mucha curiosidad, la programación visual mediante Blueprints, por la que teóricamente se puede construir un juego completo sin escribir código.

El desarrollo del videojuego se ha basado al completo en el sistema de Blueprints para comprobar que esto es cierto y para aprender a utilizar este nuevo sistema que resulta muy interesante.

5.2. Documento de Diseño del Videojuego (GDD)

En este bloque se puede encontrar todo lo relacionado al diseño del videojuego en su apartado teórico. Esto incluye diseño de mecánicas y *gameplay*, diseño sonoro, diseño de nivel, diseño gráfico, historia y detalles técnicos.

El nombre elegido para el videojuego es **The Loudest Sound** y su logo es el siguiente:



Figura 11. Título y logo del videojuego

5.2.1. El juego en términos generales

5.2.1.1. Resumen del argumento

Nos despertamos en una casa de la cual no recordamos nada y tenemos un gran dolor de cabeza. Mientras vamos investigando el escenario, nos damos cuenta de que no nos sentimos cómodos, algo raro pasa en esta casa...

Conforme avancemos en la historia, sucederán eventos paranormales e iremos descubriendo que el lugar donde nos encontramos era la antigua residencia de una feliz familia, que fue trágicamente asesinada hace unos pocos días.

5.2.1.2. Conjunto de características

Las principales características de este videojuego, y presentes en una gran cantidad de juegos de terror exitosos son las siguientes:

- Gran ambientación que nos pondrá los pelos de punta
- Buen diseño sonoro
- Buen diseño de nivel y de escenarios
- Gráficos de la actual generación de videojuegos
- Nivel perfectamente explorable, lleno de **detalles**
- Mecánicas sencillas pero esenciales
- Desarrollado con Unreal Engine 4
- Fácilmente accesible a personas nuevas en el mundo del videojuego

5.2.1.3. Género

El género elegido para este videojuego es Horror o Terror.

The Loudest Sound es una Aventura *Survival Horror*, en primera persona. Al igual que en la mayoría de los videojuegos de este género, el protagonista vive una aventura en la que debe de escapar de situaciones típicas de una película de terror (uir de un asesino, escapar de una casa llena de *zombies*, etc.). Un factor muy importante en este videojuego es el terror psicológico, ayudado de una buena ambientación y apartado sonoro.

El **horror o terror** es un género que se define por la sensación que causa: **miedo**. Nöel Carroll en su libro *The Philosophy of Horror* explica que la característica más importante del género horror es el efecto del que se causa en la audiencia, el horror necesariamente debe provocar miedo en el o al espectador.

Wikipedia

Algunos videojuegos del mismo género son: P.T, Allison Road o el nuevo Resident Evil 7.

5.2.1.4. Audiencia

Como bien dice PEGI en su web oficial, la clasificación por edades es un sistema destinado a garantizar que el contenido de los productos de entretenimiento, como son las películas, los vídeos, los DVD y los juegos de ordenador, sea etiquetado por edades en función de su contenido.

5.2.1.4.1. Clasificación PEGI del videojuego

PEGI 18:

La clasificación de adulto se aplica cuando el nivel de violencia alcanza tal grado que se convierte en representación de violencia brutal o incluye elementos de tipos específicos de violencia. La violencia brutal es el concepto más difícil de definir, ya que en muchos casos puede ser muy subjetiva pero, por lo general, puede definirse como la representación de violencia que produce repugnancia en el espectador.

Los descriptores que aparecen en el reverso de los estuches indican los motivos principales por los que un juego ha obtenido una categoría de edad concreta. Existen ocho descriptores: violencia, lenguaje soez, miedo, drogas, sexo, discriminación, juego y juego en línea con otras personas.

- **Violencia:** El juego contiene representaciones violentas.
- **Lenguaje soez:** El juego contiene palabrotas.
- **Miedo:** El juego puede asustar o dar miedo a niños.
- **Drogas:** El juego hace referencia o muestra el uso de drogas.
- **Sexo:** El juego contiene representaciones de desnudez o / y comportamientos sexuales o referencias sexuales.
- **Discriminación:** El juego contiene representaciones discriminatorias, o material que puede favorecer la discriminación.
- **Juego:** El juego fomenta el juego de azar y apuestas o enseña a jugar.
- **Juego en línea:** El juego puede jugarse en línea.

La clasificación de **The Loudest Sound** es la siguiente:



Figura 12. Clasificación PEGI de *The Loudest Sound*

5.2.1.5. Apariencia del juego

Tanto la apariencia, ambientación y experiencia de juego deben ser terroríficas pero realista. Es por ello que el videojuego está basado en una casa. El escenario a simple vista no tiene nada especial, es simplemente la casa de alguien, pero no la nuestra, y ese es uno de los factores que más inquietan al ser humano, el estar dentro de un ambiente familiar pero a la vez peligroso y lleno de desconocimiento. El diseño de nivel es especialmente cargado, muy detallista. La casa está diseñada de forma que desprenda vida, hay o habrá hace muy poco alguien viviendo ahí, pues encontramos platos de comida sucios, manchas en las paredes, objetos rotos, libros desordenados, cosas tiradas por el suelo, notas y documentos del trabajo o dibujos de niños pequeños.



Figura 13. Captura InGame

Toda la historia sucede por la noche, y la única iluminación con la que contamos son las luces de la casa y una linterna que encontraremos a lo largo del juego. Esto se traduce en un ambiente muy oscuro, lleno de misterio y peligros. Además, el jugador nunca se sentirá seguro, pues no está en sus manos en mantener las luces encendidas, pues en cualquier momento pueden fallar las luces de la casa o nuestra linterna y quedarnos en completa oscuridad, sin poder ver las amenazas que nos acechan.

5.2.2. Jugabilidad y mecánicas

En este apartado se describe tanto los aspectos de jugabilidad, en el que se incluyen los controles de juego, tanto como las mecánicas de juego, sistema de guardado, etc.

5.2.2.1. Jugabilidad

La jugabilidad es un término empleado en el diseño y análisis de juegos que describe la calidad del juego en términos de sus reglas de funcionamiento y de su diseño como juego. Se refiere a todas las experiencias de un jugador durante la interacción con sistemas de juegos.

La jugabilidad es una serie de decisiones interesantes.

SID MEIER, Game Designer

Como todo videojuego de terror, la jugabilidad no es el punto fuerte del videojuego, pues de manera similar a un videojuego de Aventura gráfica, una novela o una película, para transmitir la experiencia deseada no necesita que el jugador realice una gran cantidad de acciones. La clave para este tipo de entretenimiento es la inmersión del participante en el mundo que se le plantea.

5.2.2.1.1. Objetivo del videojuego

Nos despertamos en una casa que desconocemos y al poco tiempo nos damos cuenta de que algo extraño está sucediendo. Nuestro objetivo será responder a las siguientes preguntas mientras avanzamos por el nivel y exploramos nuestros alrededores:

- **¿Quién somos?**
- **¿Cómo hemos llegado ahí?**
- **¿De quién es esta casa?**
- Y por último, y la pregunta más importante, **¿Cómo salimos de aquí?**

El videojuego ha sido diseñado de manera que haya una mínima interfaz, por ello todas estas preguntas se irán respondiendo poco a poco, contadas por los escenarios o los eventos que sucedan dentro de la casa.

5.2.2.1.2. Progresión

En el videojuego progresamos la mayoría del tiempo de manera inconsciente. Se ha pretendido crear un flujo de juego completamente dinámico en el que no nos comunicamos directamente con el jugador para presentarle un objetivo, como podría ser por ejemplo: “Ve a la cocina”. Este tipo de progresión se puede observar sobre todo en los videojuegos del género RPG (*Role Playing Game*), donde aceptamos una **misión** que nos manda a un **lugar** a lograr un **objetivo**.

El videojuego **The Loudest Sound** posee un sistema de progresión basado en eventos que se explicará más detalladamente en el siguiente punto.

5.2.2.1.3. Libertad y estructura de eventos

En **The Loudest Sound** el jugador se puede mover **libremente** por el escenario, siempre que las puertas no estén cerradas o bloqueadas por algún motivo. Esto le permite sentir una cierta libertad con la cual explorar y dar rienda suelta a todo tipo de teorías cuando vamos encontrando nuevos elementos relevantes para la historia.

Esta libertad está controlada dentro del flujo de juego mediante un sistema de eventos.

Todos los eventos heredan desde un mismo Blueprint padre, **TriggerOfEvents**, que contiene los elementos mínimos necesarios en todo evento. Posteriormente, cada evento posee sus propias características que lo hacen único y funcional para la situación indicada.

5.2.2.1.4. Acciones del personaje

Nuestro personaje se controla como en cualquier juego de acción en primera persona. Podemos movernos libremente por el escenario e interactuar con los objetos que vayamos encontrando. Además, también podemos acercar la cámara como segunda forma de interacción. Estas mecánicas se explicarán más detalladamente en el apartado **Mecánicas**.

5.2.2.1.5. Controles del juego

El videojuego funciona correctamente tanto con teclado y ratón como con mando de juego. Los controles se han diseñado de manera que sean sencillos e intuitivos, con ello se pretende ser accesible a aquellas personas que no estén familiarizadas con el mundo de los videojuegos.

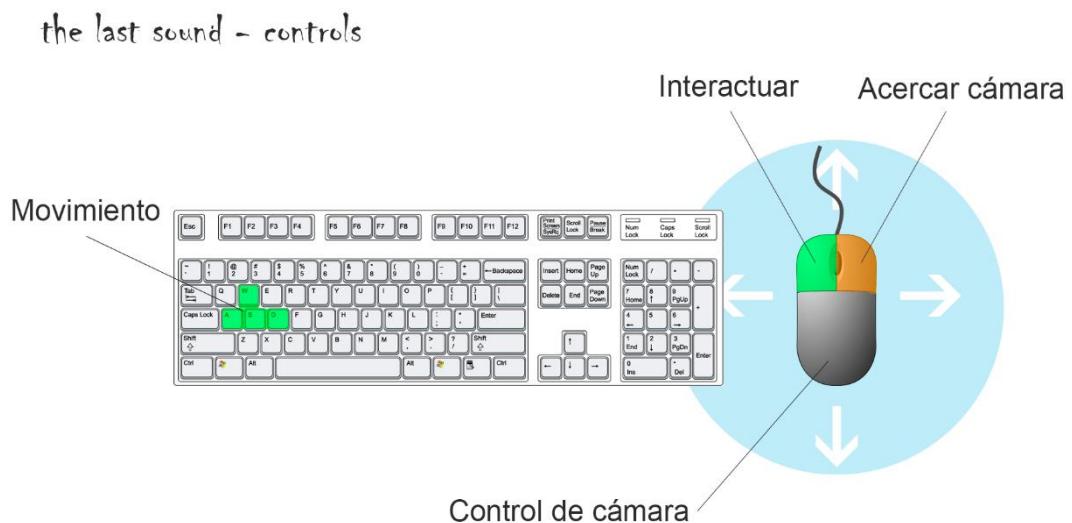


Figura 14. Controles mediante teclado y ratón.



Figura 15. Controles mediante mando de juego.

5.2.2.2. Mecánicas

Las mecánicas de los videojuegos son una serie de reglas o métodos diseñados para la interacción dentro de la ejecución del juego, creando *gameplay*. Todos los videojuegos tienen mecánicas, sin embargo, también son muy importantes la historia y el estilo del videojuego. En general, el proceso de estudio que se lleva a cabo por el *Game Designer* crea mecánicas que permiten a los jugadores engancharse al videojuego; sin embargo, estas mecánicas no siempre son una experiencia divertida.

La interacción de varias mecánicas dentro de un juego determina la complejidad y el nivel de interacción por parte del jugador dentro del juego. Algunos tipos de mecánicas de juego se han usado en los videojuegos durante mucho tiempo aunque otras son relativamente nuevas, habiendo sido inventadas en la última década.

La complejidad de las mecánicas de juego no debería ser confundida con la profundidad o incluso el realismo del juego. Esto quiere decir, que incluso con una mecánica muy simple se pueden crear videojuegos largos y tediosos que enganchen a los jugadores durante años.

En **The Loudest Sound** las mecánicas, aunque pocas, son tan importantes y utilizadas que sin ellas el videojuego no tendría sentido.

5.2.2.2.1. Movimiento y cámara en primera persona

Se trata de un videojuego tridimensional en tercera persona. La velocidad de movimiento del personaje que controlamos es la velocidad a la que anda una persona, es decir, no vamos corriendo ni trotando, puesto que es un juego en el que nos movemos por entornos cerrados y debemos explorar el escenario que nos rodea.



Figura 16. Captura Ingame desde la cámara del jugador

Nuestro movimiento es acompañado por un ligero movimiento de cámara, simulando el movimiento que realiza la cabeza de una persona al andar. Este tipo de detalles que presenta el juego se van sumando para crear una gran inmersión por parte del jugador y presentar un aspecto similar a los “videojuegos triple A”, o de grandes empresas con mucho presupuesto.

5.2.2.2.2. Interacción con objeto

En el escenario de juego podemos encontrarnos con objetos con los que podemos interactuar. Cuando interactuamos con un objeto éste se acerca a la cámara y tenemos la habilidad de rotarlo. El videojuego no se pausa cuando cogemos un objeto. Esta mecánica sirve para inspeccionarlos más de cerca y dar pie a la imaginación del jugador. Por ejemplo: nos encontramos un papel con un dibujo, interactuamos con él y al rotarlo vemos escrito “*I will find you*”.



Figura 17. Interacción con un objeto InGame

Esta mecánica funcionará de igual manera con las notas, el juego no se pausará cuando las estemos leyendo, esto crea un ambiente muy tenso y realista.

5.2.2.2.3. Abrir y cerrar puertas

Contamos con la habilidad de abrir y cerrar puertas. No las abrimos pulsando un botón, sino que al hacer clic es como si pusiéramos la mano en el pomo y empujáramos la puerta para pasar. Esta puerta posee físicas y podemos empujarla con nuestro propio cuerpo. Esto introduce al jugador dentro del nivel, otorgando la sensación de realmente tener importancia dentro del entorno que está explorado, no es una simple cámara.



Figura 18. Puerta abierta desde el punto de vista del jugador

También nos encontraremos con puertas bloqueadas o cerradas con llave, estas últimas se podrán abrir con llaves específicas que iremos encontrando.

5.2.2.2.4. Abrir y cerrar cajones o similares

De igual manera que funcionan las puertas, también podremos abrir cajones u objetos similares para encontrar objetos, notas, etc.



Figura 19. Interacción con armario de la cocina, se puede observar una puerta ya abierta y otra a punto de ser abierta por el jugador

5.2.2.2.5. Interacción mediante enfoque de cámara

Con una tecla podremos también hacer zoom, esta mecánica también favorece a crear un buen ambiente puesto que disminuye nuestro campo de visión. Imitando a la vida real, podremos “centrarnos” más en un objeto o zona pensando que hemos visto algo u observar los detalles de un objeto.



Figura 20. Radio que podemos encontrar dentro del videojuego

Por ejemplo, dentro del videojuego cuando encontramos la radio de la imagen superior, podemos centrar nuestra vista en ella. Esto hará que se ejecute un evento que tocará música en la radio.

5.2.2.2.6. Coger y guardar objetos

Algunos objetos con los que interactuemos no volverán a su posición una vez hemos terminado de examinarlos, sino que nos los guardaremos y utilizaremos automáticamente cuando sea necesario. Algunos ejemplos *InGame* son la linterna, la cual una vez la cogemos se utilizará automáticamente en entornos con poca luz.

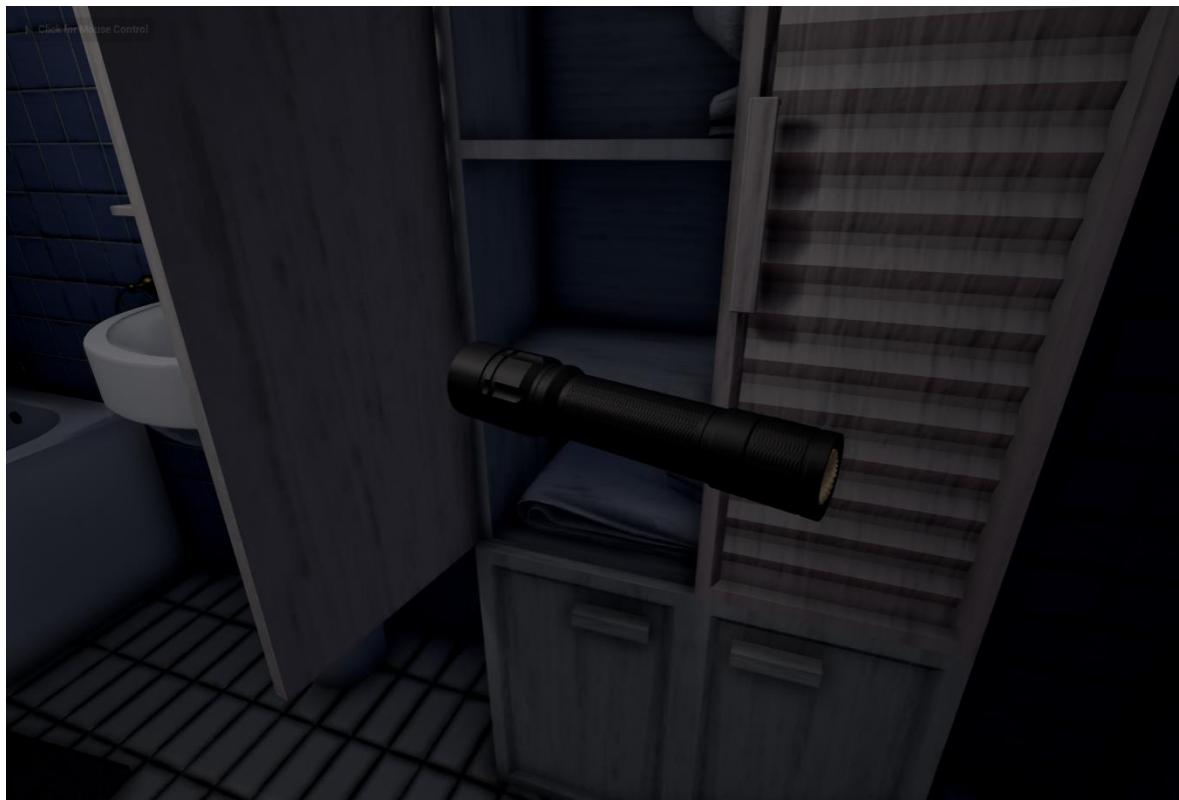


Figura 21. Objeto linterna, siendo interactuado *InGame*

Otro ejemplo son las llaves que encontramos por el escenario. Cada llave se utiliza automáticamente en la puerta correspondiente una vez la intentamos abrir (habiendo cogido previamente la llave).

5.2.2.2.7. Linterna

Dentro del juego podremos encontrar una linterna. Una vez la hayamos cogido funcionará automáticamente en entornos con poca luz, aunque siempre puede fallar...



Figura 22. Linterna siendo utilizada dentro de uno de los baños que podemos encontrar en el nivel

5.2.3. Historia, características y personajes

A pesar de que este trabajo se ha centrado en realizar tan sólo una pequeña parte de lo que sería el videojuego completo, parte de la historia y los personajes han sido trabajados, aunque de manera provisional.

5.2.3.1. Historia

La historia la vamos conociendo conforme avanzamos por el juego, mediante notas y eventos. Ningún narrador ni texto se ha utilizado pues el objetivo es que el jugador dé rienda suelta a la imaginación y cree teorías propias. De todas formas, la historia esencial que la mayoría de personas podría deducir al final del juego es la siguiente:

Nos levantamos en una casa que desconocemos, posiblemente un caso de amnesia es el responsable de esto. Se trata de la casa de una familia, al menos los dos padres y una hija, puesto que hemos encontrado numerosos dibujos de una niña pequeña llamada Lucy. Los eventos paranormales que suceden en la casa son causados por una mujer, posiblemente la madre de la familia, que intenta vengarse de algo, o de alguien. Finalmente, descubrimos que nosotros éramos el padre de esa niña y el marido de aquella mujer, ahora en forma de tenebroso fantasma. Tras muchos días sin dormir, problemas escuchando una sucesión infinita de números en la radio y múltiples visitas a un psicólogo, una noche, la locura se apoderó de nosotros. Asesinamos vilmente a nuestra mujer y a nuestra hija, y todo esto nos costó la poca cordura que nos quedaba. Esta serie de acontecimientos fue la que causó la amnesia, y una vez recordamos la triste historia, nos suicidamos para dar fin a nuestro sufrimiento.

5.2.3.2. Personajes

5.2.3.2.1. Dad

Nosotros somos el padre de familia de la casa. Una persona realista, con imperfecciones, del cual se apoderó la locura y masacró a su familia.

5.2.3.2.2. Mom

Fantasma de la madre de la familia. Nos persigue y nos atormenta por lo que hicimos. Es el principal enemigo del juego.



Figura 23. Enemigo Mom en uno de los pasillos de la casa

5.2.4. Nivel de juego

The Loudest Sound tan sólo presenta un nivel donde transcurren todos los eventos. Este nivel se trata de una clásica casa americana con un estilo clásico fácil de notar en las paredes y muebles. Se trata de una casa con una iluminación pobre donde se nota que vive gente actualmente, o hace poco tiempo al menos. Esto dentro del nivel se transmite mediante manchas en las paredes, platos sin fregar, objetos por el suelo, cosas sin recoger, etc.

Aunque todo esto no parezca importante desde el punto de vista del jugador, lo es desde el punto de vista psicológico. Al ser humano no le gusta encontrarse en lugares donde pueden vivir desconocidos, donde creemos estar solos pero no lo estamos. Igualmente el nivel ha sido diseñado de manera que incluya una gran cantidad de detalles para que el jugador se sienta abrumado.

Muchas de las habitaciones son amplias aunque repletas de elementos que explorar, y contamos con muchos pasillos estrechos que causan una sensación de agobio en el usuario.

El nivel de juego es interactivo en muchos de sus elementos, desde coger elementos para investigarlos o utilizarlos en otros lugares hasta abrir cajones en busca de una solución.

La creación del nivel está descrita en la sección de Desarrollo e Implementación.

5.2.5. Sistema de música y sonido

En **The Loudest Sound** el sonido y la música es uno de los elementos que tienen que estar más trabajados, pues es una de las claves de una buena ambientación. El sonido del videojuego presenta diversas características:

- Sonido y música cambian dependiendo de los eventos y de las decisiones del jugador.
- Alto volumen y calidad de los pasos del personaje.
- Sonidos ambiente espacializados realistas como por ejemplo: lluvia en las ventanas, sonido nevera, etc.
- Sonidos muy cuidados, de alta calidad, con una buena espacialización y atenuación.

Algunos sonidos serán sonidos ambientales que se encontrarán en todo momento dentro del nivel y se podrán parar o reanudar. Otros sonidos serán simplemente música de fondo que se tocará dados eventos específicos. Por último, algunos sonidos serán simplemente creados en el nivel por el código cuando sea necesario.

5.2.6. Inteligencia Artificial

El videojuego cuenta con una inteligencia artificial extremadamente sencilla ya que el control del flujo del juego lo lleva el Sistema de Eventos comentado anteriormente.

El enemigo que nos encontramos en el nivel posee una inteligencia artificial mínima, una vez *spawneado* (creado) dentro del nivel por el código, podremos mandarle perseguir a nuestro jugador en distintas velocidades y posiciones: andando, arrastrándose, corriendo, etc.

También tiene la habilidad de matar a nuestro jugador cuando se choca contra él.

5.3. Desarrollo e implementación

Dentro de este apartado se ha aprovechado la oportunidad para hablar de la arquitectura de Unreal Engine 4 puesto que es uno de los elementos que más influyen en el aspecto final de un videojuego, de hecho, muchos expertos saben diferenciar el motor con el que está hecho un videojuego tan sólo jugando al mismo durante unos minutos.

Además, se destacarán en mayor profundidad las utilidades y características que ofrece Unreal Engine 4, puesto que ha sido el motor elegido para el desarrollo del videojuego que se está documentando.

5.3.1. La arquitectura de Unreal Engine 4 y el sistema de scripting visual Blueprints

A continuación se van a introducir brevemente algunos de los conceptos y características más importantes que presenta Unreal. Posteriormente se indagará en el sistema de scripting visual Blueprints que incorpora Unreal y de su funcionamiento básico.

5.3.1.1. Audio y sonido

El sonido es una de las partes más importantes en la creación de escenarios inmersivos, sobre todo en los videojuegos de terror, donde el sonido es probablemente igual o incluso más importante que el resto de las características técnicas. Desde sonidos ambiente en el nivel, a sonidos interactivos de vehículos o armas, hasta el diálogo de los personajes, el audio del videojuego puede crear o destruir

la experiencia de usuario, uno de los aspectos en los que más se ha trabajado en este proyecto. Hacer que el sonido de un videojuego suene como debería de sonar, es una tarea muy complicada. El sistema de audio de Unreal Engine 4 nos provee de herramientas y utilidades para amoldar los sonidos en el juego hasta conseguir la sensación deseada. Esto es muy importante porque significa que una versión limpia del sonido se puede producir en una aplicación externa, en este caso Audacity, ser importado, y posteriormente modificado con el motor para conseguir el resultado apropiado.

5.3.1.1.1. Sound Cue Editor

El comportamiento de la reproducción de audio en Unreal Engine 4 se define mediante *Sound Cues*. El *Sound Cue Editor* es un editor basado en nodos que se utiliza para trabajar con el audio.

La **salida de audio** originada por la combinación de nodos creada en el *Sound Cue Editor* se guarda como un **Sound Cue**.

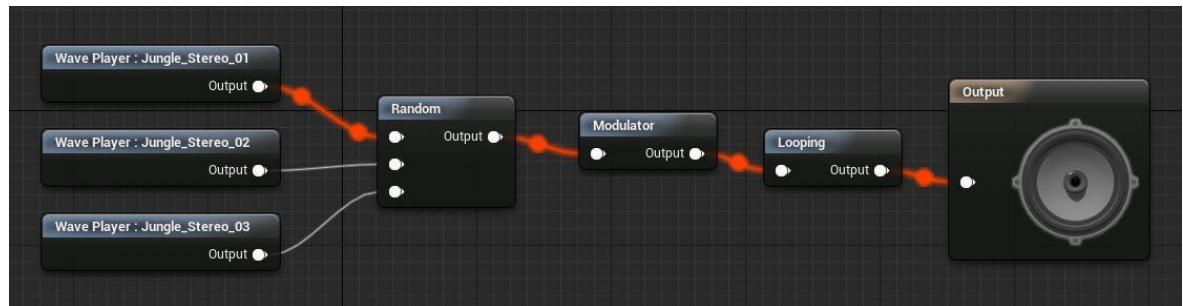


Figura 24. Ejemplo de Sound Cue

El *Audio Node Graph* muestra la señal de audio de izquierda a derecha, con nodos interconectados representando módulos de control de audio y archivos de audio. El nodo Output, que posee la imagen de un altavoz, representa el audio final que se escuchará dentro del juego y siempre está posicionado lo más a la derecha posible en el gráfico. Los **archivos de audio originales (Sound waves)** estarán siempre posicionados lo más a la izquierda posible en el gráfico. Los cables se utilizan para conectar los nodos.

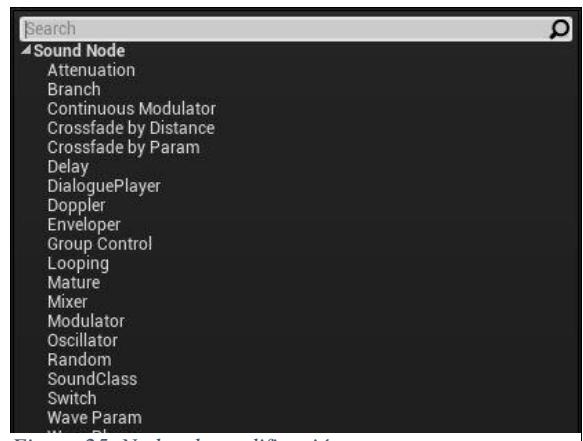


Figura 25. Nodos de modificación

Podemos seleccionar nodos de modificación del sonido de una amplia lista. Mediante estos nodos podemos modificar el audio e incluso hacer que el audio sea distinto cada vez que se ejecute.

5.3.1.1.2. Ambient Sound Actor



Figura 26. Icono de Ambient Sound Actor

Unreal Engine 4 agiliza el proceso por el cual puedes producir o modificar sonidos ambiente mediante el uso del *Ambient Sound Actor*. Cuando un *Sound Wave* o *Sound Cue* se coloca en el nivel, se crea un *Ambient Sound Actor* con ese sonido. El *Ambient Sound Actor* tiene muchas propiedades que modificarán como recibe el jugador dicho sonido.

El *Ambient Sound Actor* puede utilizarse para muchas tareas tanto si el sonido se ejecuta en bucle como si no lo hace.

Generalmente, los *Ambient Sound Actor* se comportan como los sonidos del mundo real, cuanto más cerca te encuentras del sonido, más fuerte se escucha.

5.3.1.1.3. Sound Attenuation

Sound Attenuation es básicamente la habilidad de un sonido de bajar de volumen conforme el jugador se aleja de él. Funciona utilizando dos radios, *MinRadius* y *MaxRadius*. Mientras te mueves desde la localización origen del sonido y su *MinRadius*, el volumen del sonido está al 100%. Cuando te mueves entre el *MinRadius* y el *MaxRadius* el volumen desaparece progresivamente entre 100% y silencio. La manera en la que el audio desaparece viene dada por el *Distance Algorithm* que se decida utilizar, algoritmos que proveen distintos tipos de curvas que controlan el volumen entre ambos radios.

Además de estos radios, también encontramos *Attenuation Shapes* que se pueden utilizar para elegir la forma que tendrá el propio *Attenuation* volumen pudiendo elegir entre esfera, cápsula, caja y cono.



Figura 27. Atenuación del sonido InGame

En la imagen superior podemos observar la atenuación del sonido de las gotas de lluvia golpeando la ventana.

5.3.1.2. Rendering and Graphics

El sistema de renderizado de Unreal Engine 4 es nuevo y soporta herramientas avanzadas de DirectX11 y 12 incluyendo sombreado diferido, iluminación global, transparencia con iluminación y post procesado como simulación de partículas mediante la GPU utilizando campos vectoriales.

En esta sección hay muchísimas cosas de las que se podrían hablar pero me voy a centrar en las herramientas que más he utilizado en este proyecto.

5.3.1.2.1. Lighting Basics

En Unreal Engine 4 contamos con 4 tipos distintos de luces: *Directional*, *Point*, *Spot* y *Sky*.

Las *Directional lights* son principalmente utilizadas como nuestra luz exterior principal o cualquier luz que necesite parecer que llega extremadamente lejos o hasta el infinito.

Las *Point lights* son las clásicas bombillas, emiten luz en todas las direcciones desde un punto.

Las *Spot lights* emiten luz desde un punto, pero su luz está limitada por unas formas cónicas en lugar de esferas.

Por último, las *Sky lights* capturan tu escena y aplican luz a tus modelados como una luz general.

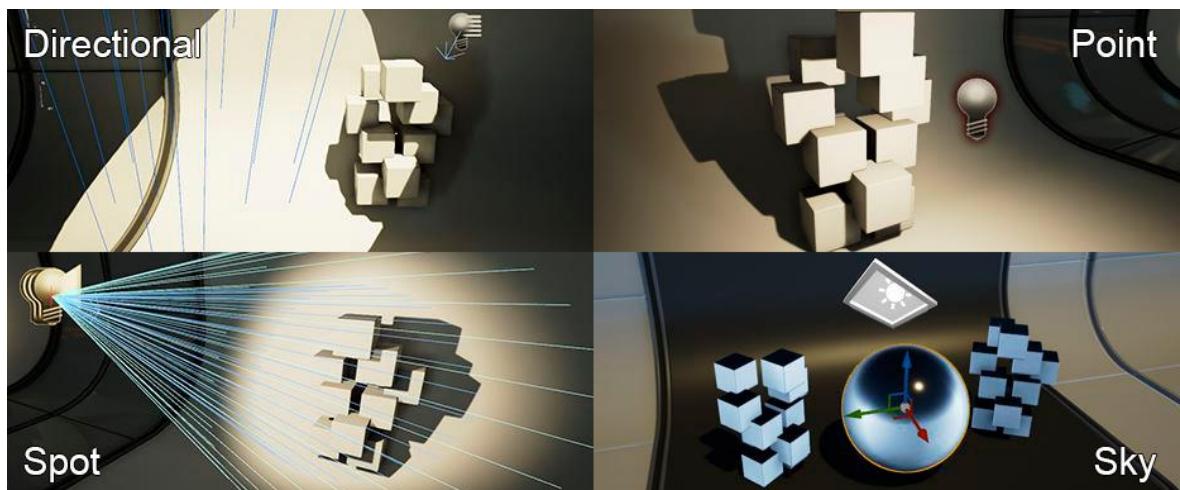


Figura 28. Tipos de luces y su aspecto

Además de estos cuatro tipos de luz, en todas las luces podemos encontrar una propiedad llamada *Mobility*. Existen tres tipos de *Mobility*: *Static*, *Stationary* y *Movable*. Cada una de estas cambia dramáticamente la manera en que funciona la luz y su impacto en el rendimiento.

5.3.1.2.1.1. Static lights

Antes de comenzar a ver este tipo de luces primero debemos comprender el concepto de *lightmap*.

Un *lightmap* es una estructura de datos usada en el *lightmapping*, una forma de cacheo de superficies en el cual el brillo de las superficies es precalculado y guardado en mapas de texturas para su posterior uso. Los *lightmaps* comúnmente se aplican a los objetos estáticos en aplicaciones 3D en tiempo real,

como los videojuegos, para proveer de efectos de iluminación avanzada como la iluminación global a un coste de computación muy bajo.

Las luces *Static* son luces que no se pueden cambiar o mover de ninguna manera durante el juego. Son calculadas tan sólo dentro de los *lightmaps*, y una vez procesadas, no tienen impacto en el rendimiento. Los objetos dinámicos no se benefician de las luces *Static*, por ello tienen límites claros.

De los tres tipos de *Mobility*, las luces *Static* tienen **calidad media**, la **mutabilidad más baja** y el **coste de rendimiento más bajo**.

Dado que las *Static lights* sólo utilizan *Lightmaps*, las sombras son calculadas antes del *gameplay*. Esto significa que no admiten sombras dinámicas. Sin embargo, cuando los objetos iluminados son también estáticos, tienen la habilidad de producir sombras de área. También debe hacerse notar que las superficies que reciban sombras suaves seguramente deberán ajustar la resolución de su *Lightmap* para que las sombras se vean con buena calidad.

5.3.1.2.1.2. Stationary lights

Las luces de *Mobility Stationary* son luces que tienen que mantenerse en una posición, pero son capaces de cambiar de otras maneras, como por ejemplo su brillo y su color. Esta es la principal característica que las diferencia de las luces *Static*, que no pueden cambiar de ninguna manera durante el *gameplay*. De todas maneras, cabe notar que los cambios durante el tiempo de ejecución tan sólo afectan a la iluminación directa. La iluminación indirecta (rebotada y calculada por la iluminación global), puesto que es precalculada anteriormente, no cambiará.

De los tres tipos de *Mobility*, las luces *Stationary* tienen la **mayor calidad**, una **mutabilidad media** y un **coste de rendimiento medio**.

Toda la iluminación indirecta y el sombreado de las *Stationary lights* se guarda en el *Lightmap*. Las luces directas se guardan dentro del *Shadowmap*. Estas luces utilizan el *Distance Field Shadows*, lo

que significa que sus sombras pueden permanecer afiladas incluso con una resolución del *Lightmap* bastante baja en objetos ya iluminados.

5.3.1.2.1.3. Movable lights

Las *Movable lights* producen iluminación y sombras completamente dinámicas, pueden cambiar su posición, rotación, color, brillo, desvanecimiento, radio y todas las propiedades que posee una luz. Ninguna de la iluminación que producen es calculada dentro de los *lightmaps* y no admiten iluminación indirecta.

De los tres tipos de *Mobility*, las luces *Movable* tienen una **buenas calidad**, la **mutabilidad más alta** y un **alto coste de rendimiento**.

5.3.1.2.2. Material conceptos básicos

Un material es un *asset* (recurso) que puede ser aplicado a un *mesh* (modelo) para controlar el aspecto visual de la escena. A un nivel más alto, probablemente es más fácil pensar un material como la pintura que es aplicada a un objeto. Pero incluso eso puede ser un poco incorrecto, pues un material literalmente define el tipo de superficie del que tu objeto aparenta estar construido. Se puede definir su color, su brillo, si puedes ver a través del objeto y mucho más.

Los materiales son uno de los aspectos más críticos para conseguir la apariencia que se desea conseguir en los objetos y niveles.

Lo más importante que hay que saber de los materiales es que no están construidos mediante código, sino mediante una red de scripting visual similar a los Blueprints, formados por nodos llamados Material Expressions dentro del Material Editor. Cada nodo contiene parte de código HLSL (*High-Level Shading Language*), diseñado para llevar a cabo una tarea específica. Esto significa que cuando construimos un material estamos creando código HLSL mediante scripting visual.

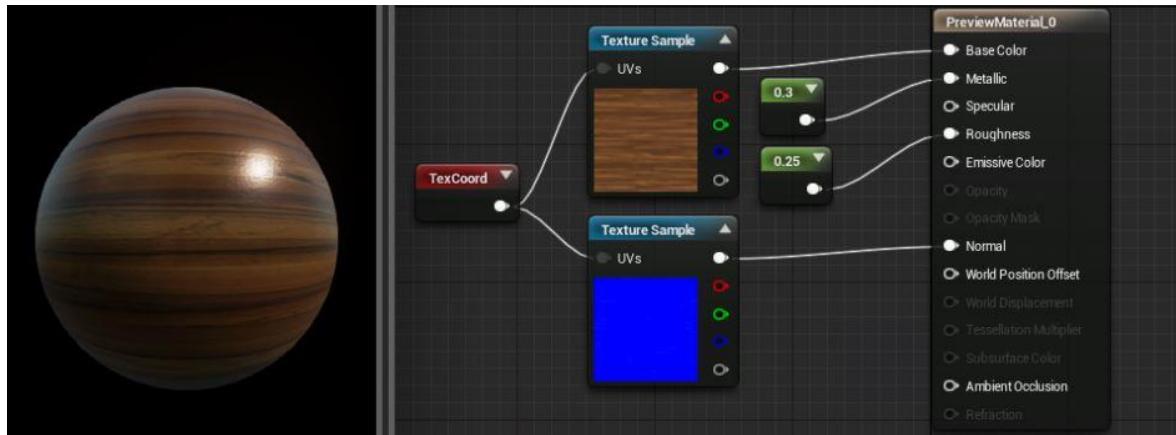


Figura 29. Ejemplo de material Madera y sus nodos básicos

En la figura 29 podemos observar una red muy simple definiendo un suelo de madera. Sin embargo, las redes de *Material Expression* o materiales no tienen que ser simples. Es común tener algunos materiales que tienen muchas docenas de nodos *Material Expression* dentro de ellos.

5.3.1.2.3. PostProcessVolume conceptos básicos

El *PostProcessVolume* es un tipo especial de volumen que puede ser añadido al nivel para cambiar los parámetros del post procesado. En Unreal Engine 4, cada *PostProcessVolume* es básicamente un tipo de capa que se superpone. Con este volumen se permite a los artistas y a los diseñadores modificar la apariencia y ambientación general de la escena. Algunos ejemplos de elementos y efectos incluyen *Bloom*, oclusión ambiental o cambio de tono entre muchos.

5.3.1.3. Conceptos básicos de la arquitectura

El objetivo de este bloque es hacer una breve introducción a las características más importantes del motor e introducir algunos de los conceptos base para comprender el resto del proyecto.

5.3.1.3.1. UObjects y Actors

Los Actores o *Actors* son instancias de clase que derivan de la clase AActor; la clase base de todos los objetos que pueden ser colocados en el mundo del juego. Los Objetos u *Objects* son instancias de clases que heredan de la clase UObject; la clase base de todos los objetos de Unreal Engine, incluyendo a los Actores. Así que, en realidad, todas las instancias de Unreal Engine son Objetos; sin embargo, el término Actor es comúnmente utilizado para referirnos a las instancias de clases que derivan de AActor en su jerarquía, mientras que el término Objeto es utilizado para referirnos a instancias de clases que no heredan de la clase AActor. La mayoría de las clases se crearán heredarán de AActor en algún punto de su jerarquía.

En general, los Actores pueden pensarse como objetos o entidades completas, mientras que los Objetos son partes más especializadas. Los actores comúnmente utilizan Componentes, que son objetos especiales, para definir algunos aspectos de su funcionalidad o mantener valores para una colección de propiedades. Por ejemplo imaginemos un coche. El coche en sí mismo es un Actor, mientras que las partes del coche, como las puertas y las ruedas, serían Componentes de ese Actor.

5.3.1.3.2. Gameplay y las clases del Framework

Las clases básicas del *gameplay* incluyen funcionalidad para representar Jugadores, Aliados y Enemigos, de igual manera, para controlar estos avatares mediante Inputs del Jugador o lógica de inteligencia artificial. También encontramos clases para crear *Heads-Up Displays* (HUD o interfaz *InGame*) y cámaras para los jugadores. Finalmente, las clases del *gameplay* como *GameMode*, *GameState* y *PlayerState* imponen las reglas del juego, y rastrean como está progresando el juego y los jugadores.

Estas clases crean tipos de Actores, que pueden o bien ser colocadas dentro del nivel o generadas cuando se necesite.

5.3.1.3.3. Representando Jugadores, Aliados y Enemigos en el Mundo

5.3.1.3.3.1. Pawn

Un *Pawn* es un Actor que puede ser un “agente” dentro del mundo. Los *Pawns* pueden ser poseídos por un *Controller*, están programados para aceptar input fácilmente, y pueden hacer varias acciones como haría otro Jugador. Cabe notar que un *Pawn* no tiene por qué ser humanoide.

5.3.1.3.3.2. Character

Un *Character* es un *Pawn* de estilo humanoide. Por defecto posee un *CapsuleComponent* (cápsula) para la colisión y un *CharacterMovementComponent* (movimiento). Puede hacer movimientos humanoides, puede replicar sus movimientos suavemente en la red y tiene alguna funcionalidad relacionada con la animación.

5.3.1.3.4. Controlando Pawns con Player Input o lógica IA

5.3.1.3.4.1. Controller

Un *Controller* es un Actor que es responsable de dirigir un *Pawn*. Típicamente viene de 2 formas, *AIController* y *PlayerController*. Un *Controller* puede “poseer” un *Pawn* y tomar control de él.

5.3.1.3.4.2. PlayerController

Un *PlayerController* es la interfaz entre el *Pawn* y el jugador humano controlándolo. El *PlayerController* esencialmente representa la voluntad del jugador humano.

5.3.1.3.4.3. AIController

Un *AIController* es lo que suena, una “voluntad” simulada que controla un *Pawn*.

5.3.1.3.5. Mostrando información a los Jugadores

5.3.1.3.5.1. HUD

Un HUD es un “*Heads-Up Display*”, o esa interfaz 2D que se muestra encima de la pantalla que es común en muchos videojuegos. Un ejemplo sería la vida, munición, retícula del arma, etc. Cada *PlayerController* o Jugador típicamente posee un HUD.

5.3.1.3.5.2. Cámara

El *PlayerCameraManager* es el “ojo” para un jugador y controla cómo se comporta. Cada *PlayerController* típicamente también tiene uno de estos.

5.3.1.3.6. Estableciendo y Rastreando las reglas del juego

5.3.1.3.6.1. GameMode

El concepto de Juego está dividido en 2 clases. El *GameMode* es la definición del juego, incluyendo cosas como las reglas del juego y las condiciones de victoria. Tan solo existe en el servidor. Típicamente no debería de haber muchos datos que cambien durante la partida y definitivamente no debe contener datos que los jugadores necesiten conocer.

5.3.1.3.6.2. GameState

El *GameState* contiene el estado del juego, que puede incluir cosas como la lista de jugadores conectados, la puntuación, donde están las piezas en un juego de ajedrez, o la lista de misiones que has completado en un juego de mundo abierto. El *GameState* existe en el servidor y en todos los clientes y puede ejecutarse libremente para mantener todas las máquinas actualizadas.

5.3.1.3.6.2. PlayerState

Un *PlayerState* es el estado de uno de los participantes en el juego, como un jugador o un robot que esté simulando ser un jugador. La IA que existe como parte del juego que no representa jugadores humanos no posee un *PlayerState*. Por ejemplo, algunos datos que serían apropiados en un videojuego estilo MOBA, donde varios jugadores compiten online en una arena, y que estarían dentro del *PlayerState* serían el nombre del jugador, puntuación, nivel en la partida, o por ejemplo en una partida de capturar la bandera en un FPS el *PlayerState* sabría si el jugador está actualmente llevando la bandera. Los *PlayerState* para todos los jugadores están en todas las máquinas (no como los *PlayerControllers*) y pueden ejecutarse libremente para mantener todo actualizado.

5.3.1.3.7. Las relaciones de las clases del Framework

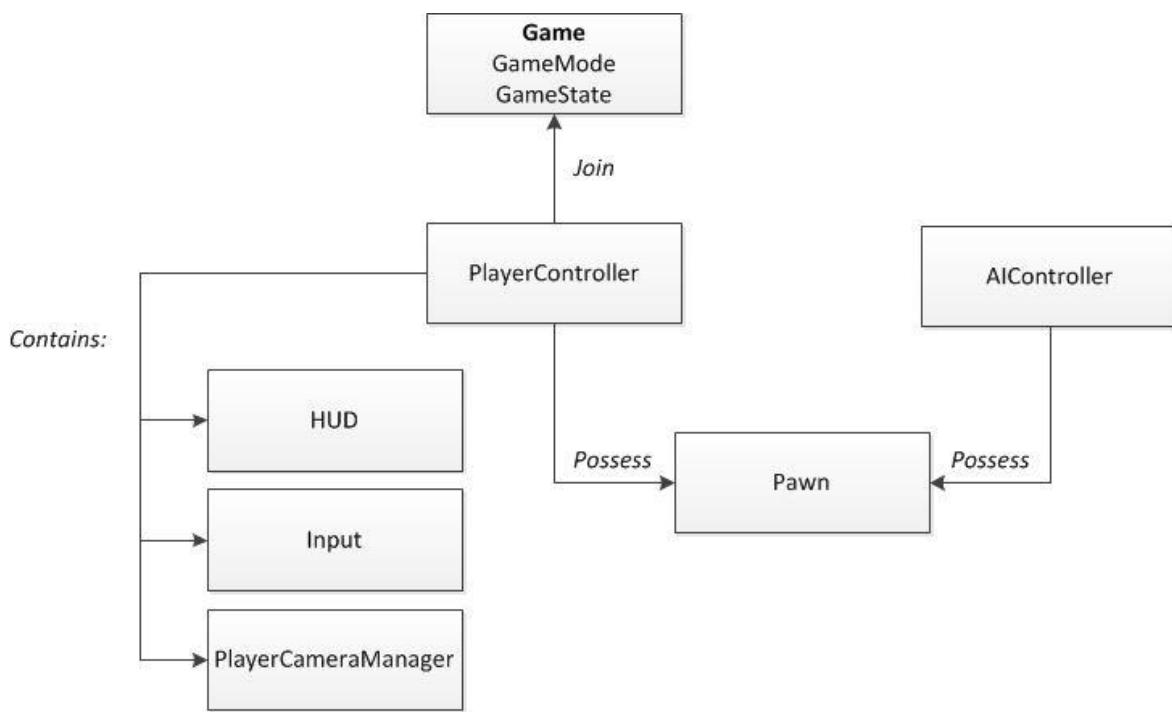


Figura 30. Relaciones del Framework

El gráfico de arriba muestra como las distintas clases base del *gameplay* se relacionan entre ellas. Un juego está construido por un *GameMode* y un *GameState*. Los jugadores humanos que se unen a la partida están asociados con los *PlayerController*. Estos *PlayerController* permiten a los jugadores poseer *Pawns* dentro del juego para que puedan tener representación física dentro del nivel. Los

PlayerControllers también dan a los jugadores controles mediante Input, un *Heads-Up Display* o *HUD* y un *PlayerCameraManager* para controlar la visión.

5.3.1.4. Scripting visual Blueprints

El sistema de scripting visual Blueprints en Unreal Engine es un sistema completo de programación de juego basado en el concepto de usar una interfaz basada en nodos para crear elementos del *gameplay* desde dentro del propio editor Unreal. Como en muchos de los lenguajes de programación, se utiliza para definir clases orientadas a objetos dentro del motor. Posteriormente, nos referiremos a la mayoría de objetos creados mediante Blueprint directamente como “Blueprints”. Este sistema es extremadamente flexible y poderoso ya que provee a los diseñadores de la habilidad de utilizar una gran cantidad de conceptos y herramientas generalmente tan solo disponibles a los programadores. En adición, un sistema disponible en la implementación de C++ de Unreal Engine permite a los programadores crear bases generales que pueden ser utilizadas posteriormente por los diseñadores.

5.3.1.4.1. ¿Cómo funcionan?

En su forma básica, los Blueprints son adiciones al juego programadas visualmente. Conectando **Nodos**, **Eventos**, **Funciones** y **Variables** con **Cables** es posible crear mecánicas y elementos complejos del juego.



Figura 31. Imagen introductoria a los blueprints.

Los Blueprints funcionan usando gráficos de Nodos para múltiples objetivos – construcción de objetos, funciones individuales y eventos generales del *gameplay* – que son específicos para cada instancia del Blueprint de manera que podemos crear otros comportamientos y funcionalidades.

Hay que tener en cuenta que pueden haber blueprints **padre** y blueprints **hijo** que hereden de padres. Cuando un blueprint hereda de otro, todas las funciones y variables del padre también son heredadas. También hay que tener en cuenta que si existe una función con el mismo nombre tanto en el padre como en el hijo, la del hijo sobrescribirá a la del padre.

5.3.1.4.2. Tipos de Blueprints

Los tipos de Blueprints más comunes con los que se suele trabajar son los Level Blueprints y los Blueprint Classes. Estos son los dos tipos de Blueprints, que también incluyen Blueprint Macros y Blueprint Interfaces.

5.3.1.4.2.1. Level Blueprint

Un Level Blueprint es un tipo especializado de Blueprint que actúa como un gráfico de eventos global que comprende el nivel completo. Cada nivel en el proyecto tiene su propio Level Blueprint creado por defecto que puede ser editado dentro del Editor Unreal, sin embargo, no se pueden crear nuevos Level Blueprint desde la interfaz del editor.

Los eventos que pertenecen al nivel o instancias específicas actores de dentro del nivel son utilizados para ejecutar secuencias de acciones en forma de llamadas a Funciones u operaciones de control de flujo.

5.3.1.4.2.2. Blueprint Class

Un Blueprint Class, comúnmente acortado Blueprint, es un recurso que permite a los creadores de contenido añadir nuevas funcionalidades fácilmente encima de clases del juego ya existentes. Los Blueprints son creados de forma visual dentro del editor de Unreal, en lugar de escribir código y ser guardados como recursos en un paquete de contenido. Este tipo de Blueprint esencialmente define una nueva clase o tipo de Actor que puede ser colocado dentro de mapas como instancias que se comportan de igual manera que lo haría otro Actor.

5.3.1.4.2.3. Data-Only Blueprint

Un Data-Only Blueprint es un Blueprint Class que contiene solo el código (en forma de gráficos de nodos), variables y componentes heredadas de su padre. Estos Blueprints permiten cambiar y modificar esas propiedades heredadas, pero no se pueden añadir nuevos elementos. Son esencialmente un reemplazo para los arquetipos y pueden ser utilizados por los diseñadores para cambiar propiedades o crear elementos con variaciones.

5.3.2. Diseño artístico

En esta sección se comenta detalladamente todo lo relacionado con el diseño artístico del videojuego, tanto visual como sonoro. Cabe destacar que el diseño artístico del videojuego ha sido altamente influenciado por los videojuegos que se han utilizado de referencia y que tuvieron tanto éxito cuando aparecieron.

5.3.2.1. Diseño y arte visual

El diseño gráfico del videojuego ha sido otra de las partes más complicadas del videojuego, pues crear un ambiente realista con una gran limitación en el número de polígonos de los modelados ha sido una tarea complicada. De igual manera, los materiales y texturizados debían ser de gran calidad, tanto de iluminación, reflejos e imágenes de gran calidad.

Cabe destacar que no todos los elementos que se han utilizado son de fuente propia, sino que algunos de ellos se han obtenido de manera gratuita con licencia **Creative Commons**.

5.3.2.1.1. Modelados y optimizaciones

Como ya se ha comentado anteriormente, algunos de los modelados han sido realizados por mí, mientras que otros se han descargado de forma gratuita y presentando licencia Creative Commons. Es por ello que en este apartado también se habla de las optimizaciones, pues la gran mayoría de los modelados que han sido descargados de manera gratuita no estaban orientados al mundo de los

videojuegos. Es por ello que contaban con una cantidad de polígonos increíblemente alta y algunos de ellos con Unwraps prácticamente imposibles de realizar.

Puesto que se han realizado muchos modelos tan sólo se van a comentar aquellos que tengan algo distinto, interesante o importante. De igual manera, tan sólo se comentará como se han optimizado los modelos descargados gratuitamente. Si se desean ver más imágenes de modelados se pueden observar en el anexo “Arte”.

5.3.2.1.1.1. Entornos modulares

Dadas las características de este proyecto y su ambición de ser altamente ampliado y comercializado en un futuro se ha tomado la decisión de crear los entornos de manera modular. Esto quiere decir que los elementos se han creado como si fueran “piezas” de un puzzle infinito.

Tan sólo se han modelado de forma modular los elementos arquitectónicos de la casa, esto incluye paredes, esquinas, columnas, puertas y ventanas entre otros.

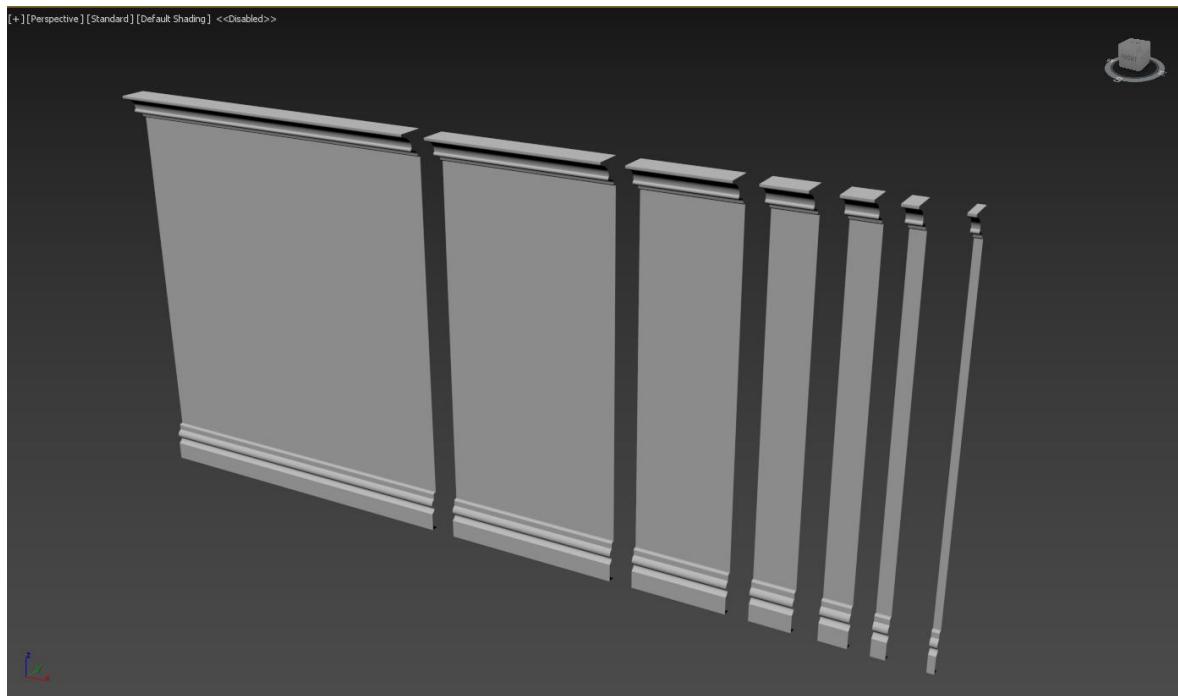


Figura 32. Paredes modulares dentro de 3ds Max.

Todos estos elementos tienen unas medidas exactas, múltiplos de 5, para ser fácilmente manejadas en el editor de niveles que incorpora Unreal.

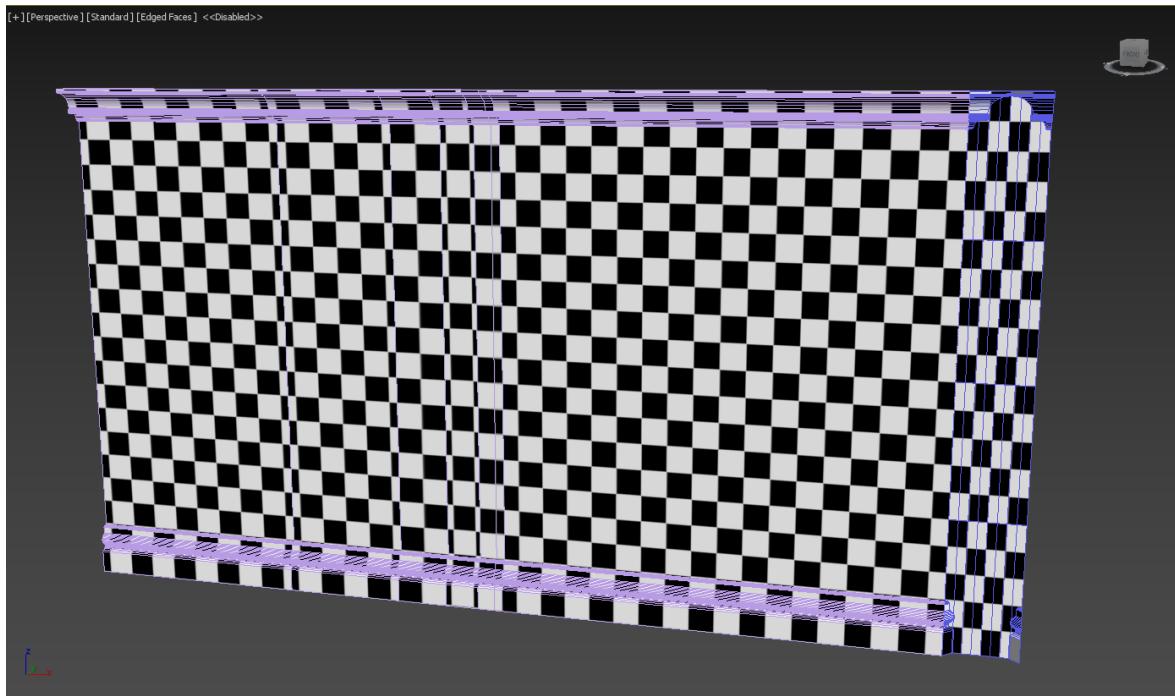


Figura 33. Unwraps de las paredes dentro de 3ds Max.

Además, en todos se han realizado los Unwraps (“aplanar” el modelo 3D para su posterior texturizado) de manera que tengan una misma escala y representen de manera muy similar las texturas aplicadas.

Además de las paredes, crear los entornos de manera modular obligó a crear también las esquinas interiores y exteriores, incluyendo molduras específicas para dichas zonas. Además, la colisión de estas partes también tuvo que ser modificada dentro de 3ds Max para que se comportaran de una manera correcta.

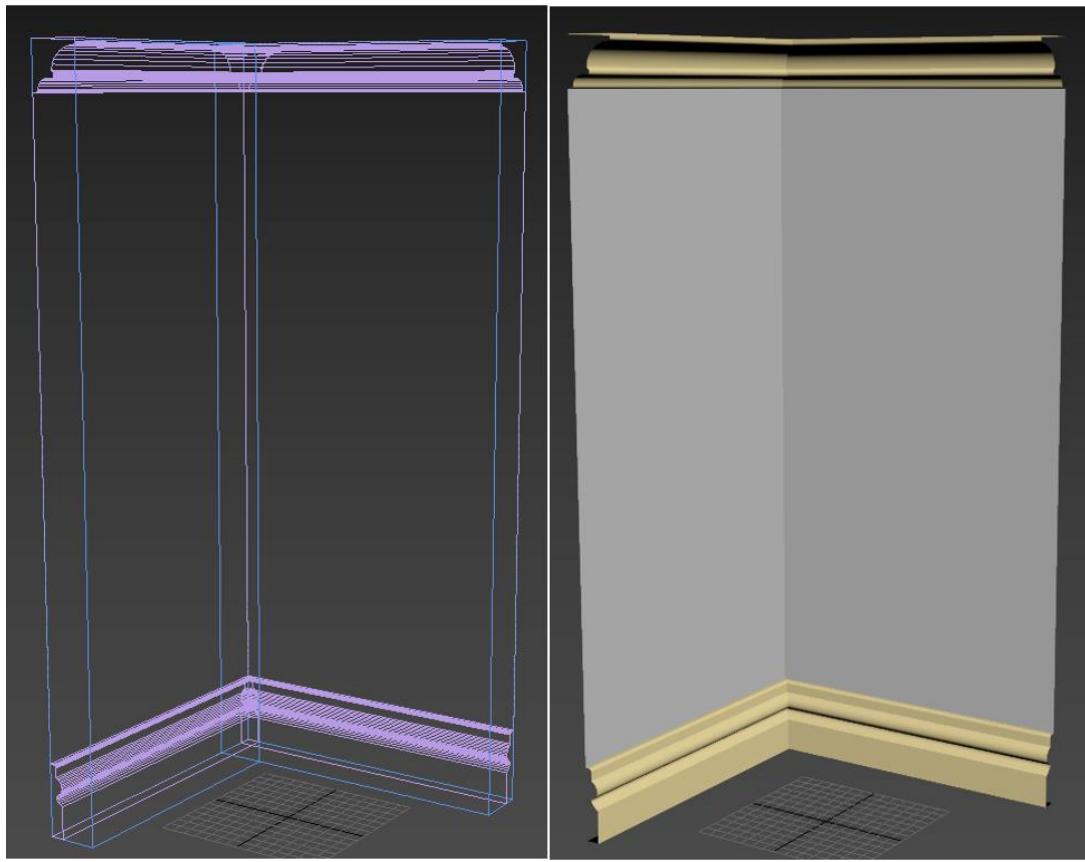


Figura 34. Pared "Esquina Interior" y su colisión en color azul.

Gracias a haber utilizado esta técnica, una vez construidas las piezas, el nivel se construyó muy rápidamente y se pudo modificar conforme se probaba el *gameplay*.

5.3.2.1.1.2. Libros y script FillMyBookshelves

Los libros fueron una parte interesante del modelado puesto que al querer hacer los entornos con un alto nivel de detalle se requería tener bastantes libros distintos tanto en forma, tamaño y textura. Es por ello que se modelaron y texturizaron un total de 22 libros.

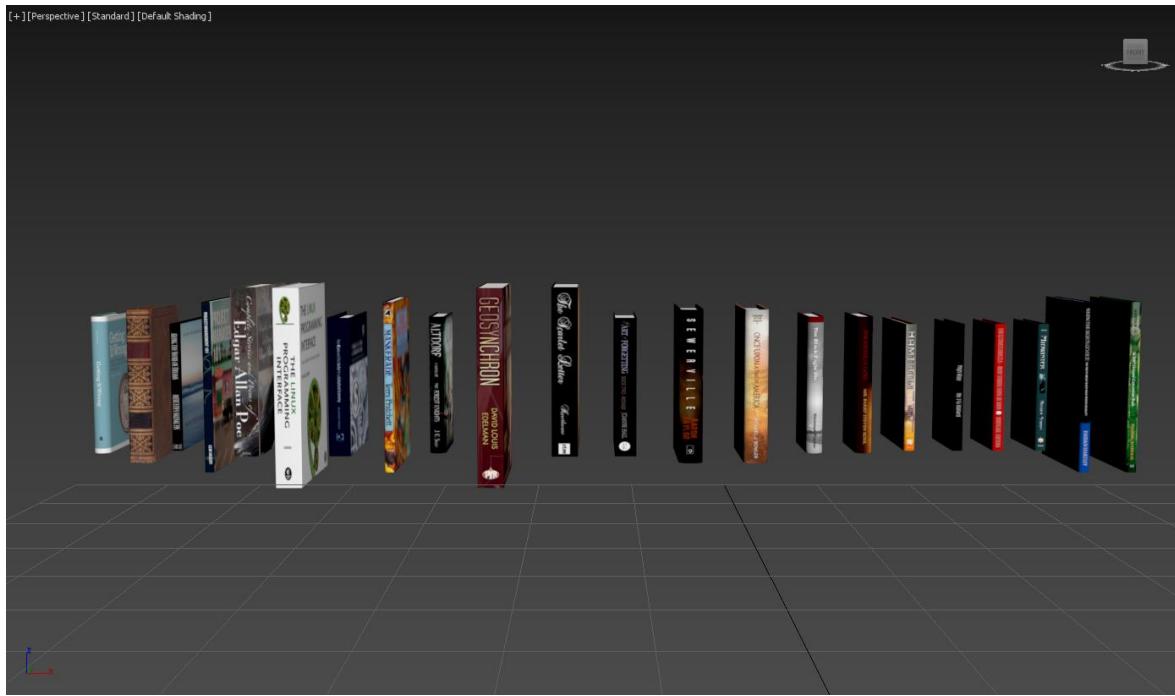


Figura 35. Modelado y texturizado de los 22 libros.

Estos libros aparte de poder encontrarse libremente en el nivel y poder interactuar con algunos de ellos, también debían de encontrarse en las estanterías y eso era un problema puesto que colocarlos a mano podría requerir horas de tiempo, sobre todo si se querían hacer distintos tipos de ordenación y conseguir un aspecto realista. En este momento es en el que entra en escena el script *FillMyBookshelves* para 3ds Max.

Investigando acerca de este problema encontré este script gratuito que permite establecer una serie de objetos “libro” y posteriormente darle unos valores al script para ordenarlos en una estantería automáticamente. El resultado es muy satisfactorio como podemos comprobar:



Figura 36. Ejemplo libros colocados en estantería número 1.



Figura 37. Ejemplo libros colocados en estantería número 2.

Gracias al número distinto de libros que se han creado, dentro del videojuego es sencillo ignorar la repetición y sentirnos inmersos en la experiencia que se plantea.

5.3.2.1.1. Optimizaciones

Como se ha comentado anteriormente, el 90% de los modelados descargados tenía una cantidad de polígonos demasiado alta para los requerimientos del videojuego. Es por ello que se han tenido que optimizar con las herramientas de 3ds Max.

Este trabajo normalmente lleva bastante tiempo, pues cuando las herramientas de optimización de 3dsMax actúan, suelen romper el modelado y se tiene que reparar manualmente. Tras mucho esfuerzo se pueden conseguir resultados decentes como se puede ver en las siguientes imágenes:



Figura 38. Modelado gratuito (candelabro) antes de ser optimizado.

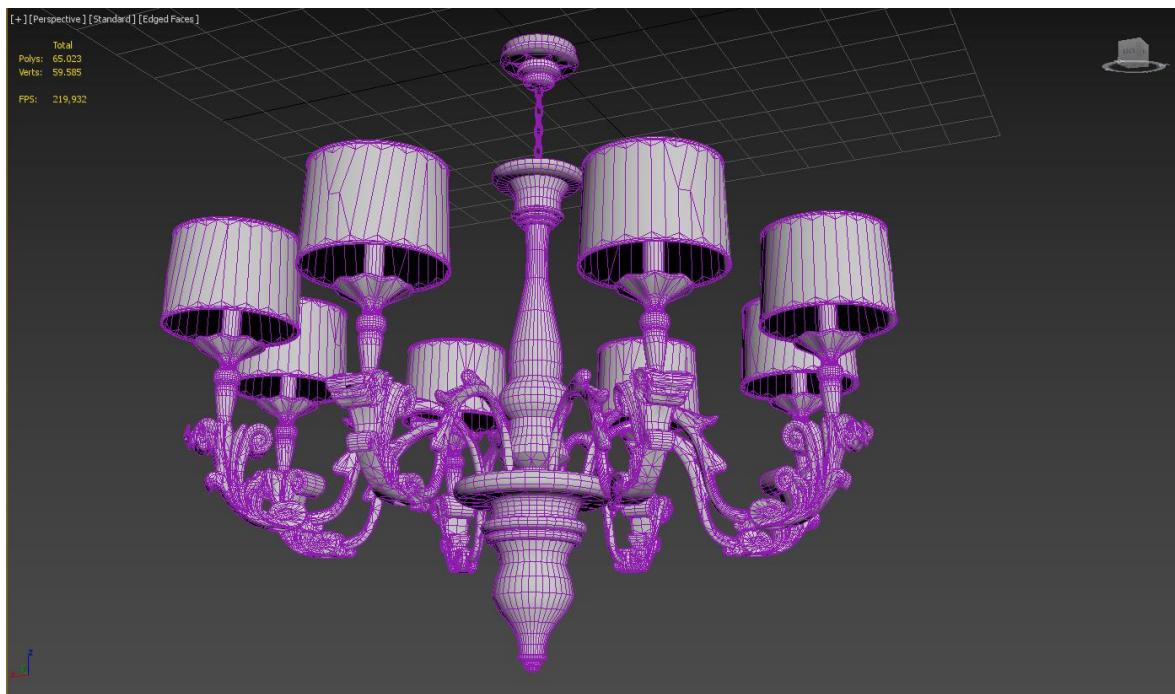


Figura 39. Modelado gratuito (candelabro) tras ser optimizado.

Modelo Candelabro	
Número de polígonos inicial	196.344
Número de polígonos final	65.023
Número de vértices inicial	99.216
Número de vértices final	59.585

Figura 40. Tabla de optimización del modelo "candelabro".

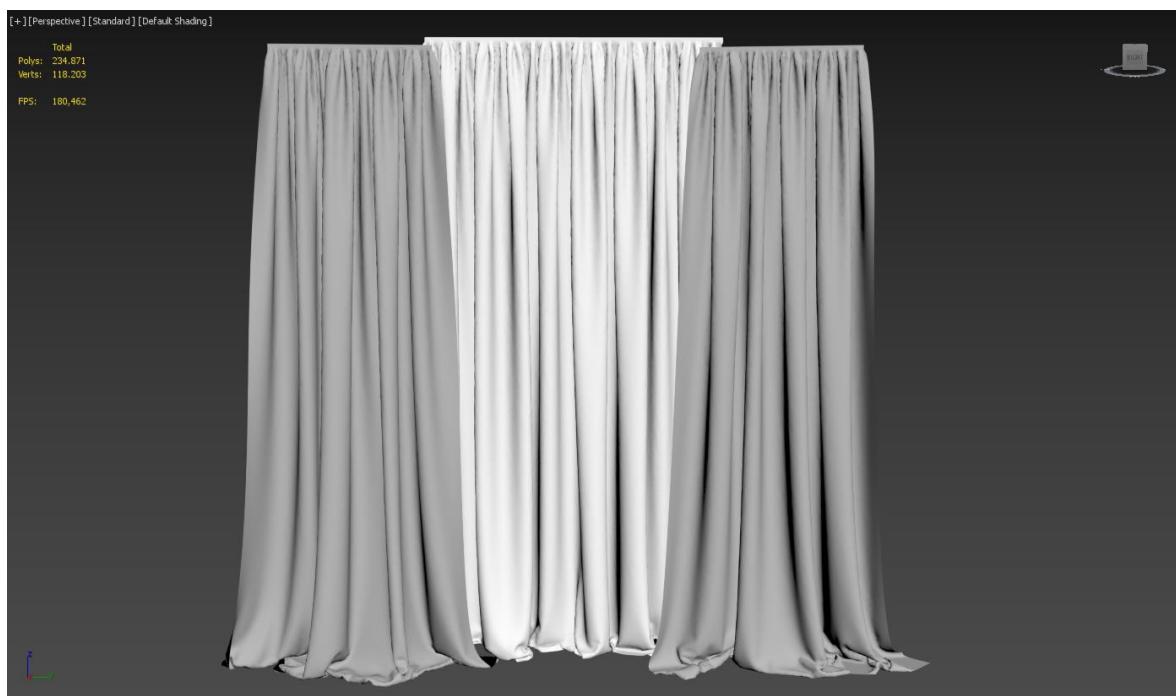


Figura 41. Modelado gratuito (cortinas) antes de ser optimizado número 1.

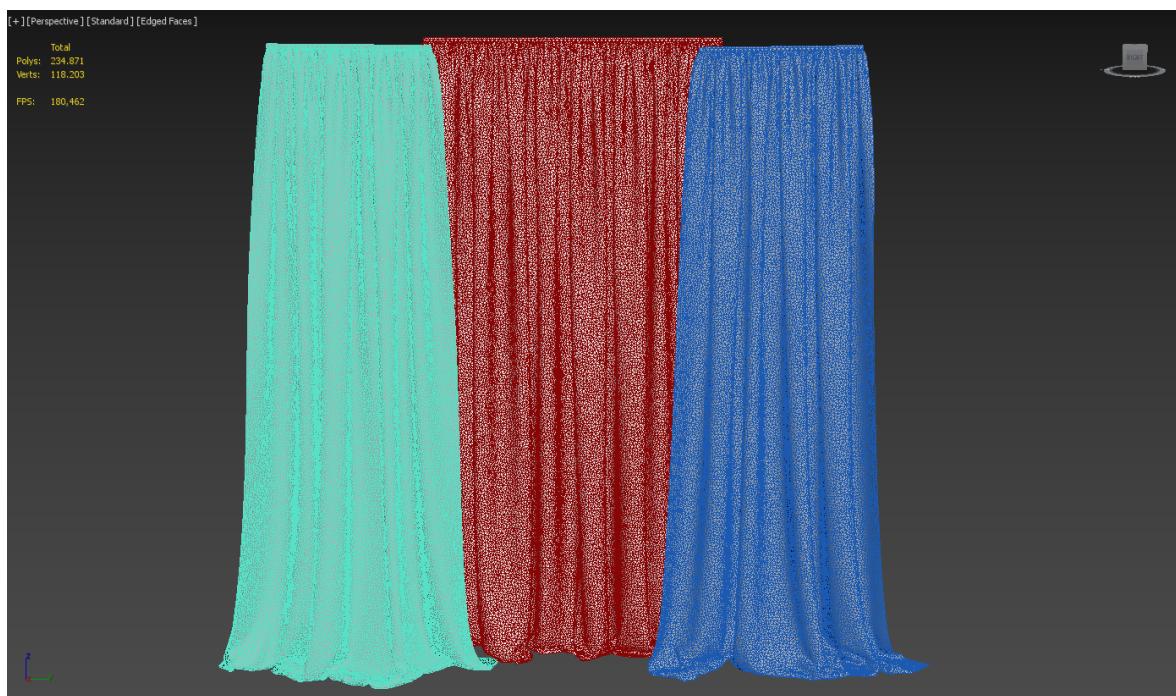


Figura 42. Modelado gratuito (cortinas) antes de ser optimizado número 2.

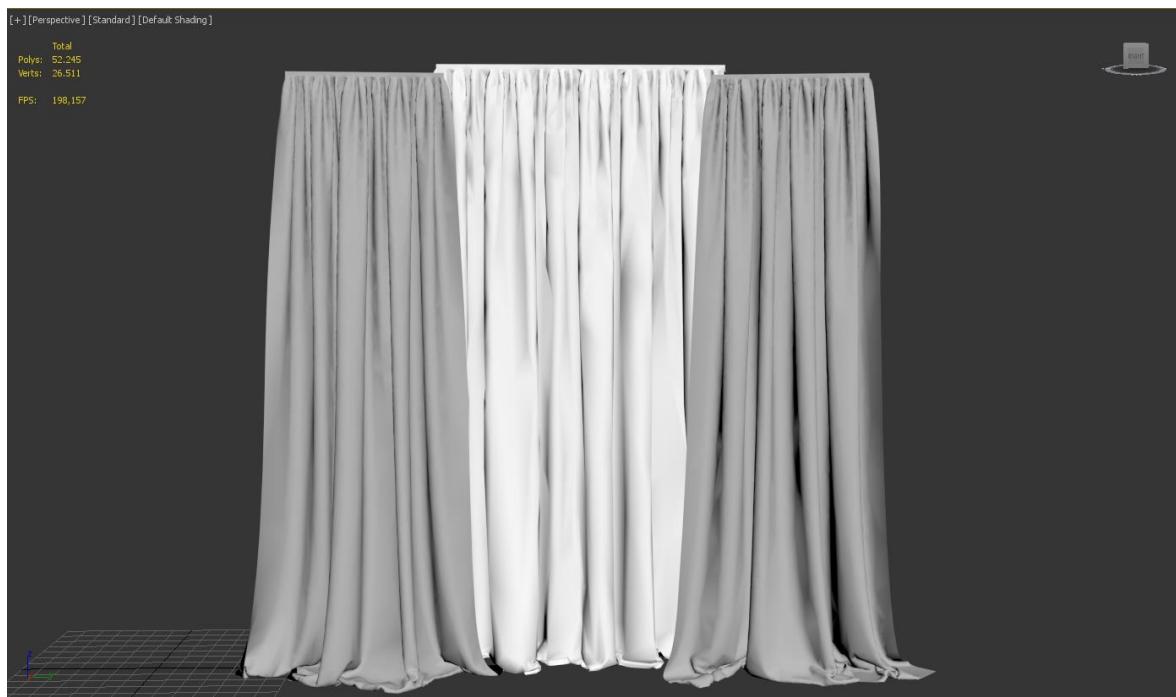


Figura 43. Modelado gratuito (cortinas) tras ser optimizado número 1.



Figura 44. Modelado gratuito (cortinas) tras ser optimizado número 2.

Modelo Cortinas	
Número de polígonos inicial	234.871
Número de polígonos final	52.245
Número de vértices inicial	118.203
Número de vértices final	26.511

Figura 45. Tabla de optimización del modelo "cortinas".

Estos tan sólo son dos ejemplos, pero se puede observar claramente que a pesar de perder un poco de calidad visual, es necesario realizar las optimizaciones pues se reduce el coste de renderizado y por lo tanto mejora el rendimiento considerablemente.

5.3.2.1.2. Materiales y texturizado

Antes de comenzar cabe distinguir entre los distintos tipos de mapas de texturas más comunes que podemos encontrar normalmente en el texturizado de modelados 3D:

- **Mapa difuso:** esencialmente son los colores que posee el modelo. Coloca la imagen 2D de la textura en el modelo 3D.



Figura 46. Ejemplo mapa difuso.

- **Mapa de normales:** simulan la impresión de una Superficie 3D, es decir, de relieve. Pero este relieve no va a proyectar ninguna sombra y no obstruirá a otros objetos. Si el ángulo de cámara es rasante en relación a la superficie, nos daremos cuenta de que la superficie no tiene relieve en realidad.

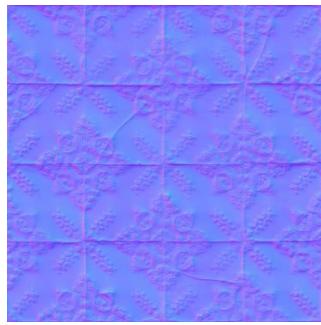


Figura 47. Ejemplo mapa de normales.

- **Mapa de oclusión:** los mapas de oclusión se utilizan para informar qué partes del modelo deberían recibir alta o baja iluminación indirecta.



Figura 48. Ejemplo mapa de oclusión.

- **Mapa especular:** los mapas especulares son los que definen el brillo y color de las partes iluminadas de un objeto.

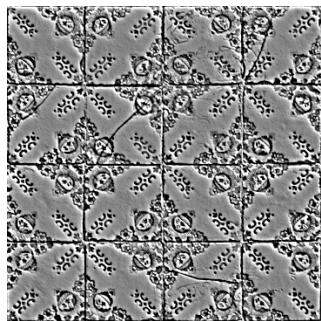


Figura 49. Ejemplo mapa especular.

En **The Loudest Sound**, todos estos mapas han sido creados gracias a la versión gratuita de la herramienta *CrazyBump*, que puede encontrarse en el siguiente link: <http://www.crazybump.com/>.

El texturizado y los materiales presentes en todos los modelos del videojuego han sido una de las partes más costosas y también trabajadas del proyecto. Un material y una textura realista consiguen darle un aspecto magnífico a los modelados con pocos polígonos.

Antes de crear las texturas y los materiales se ha hecho un Unwrap adecuado a los objetos. Un ejemplo muy claro de una textura sencilla es la de los libros:

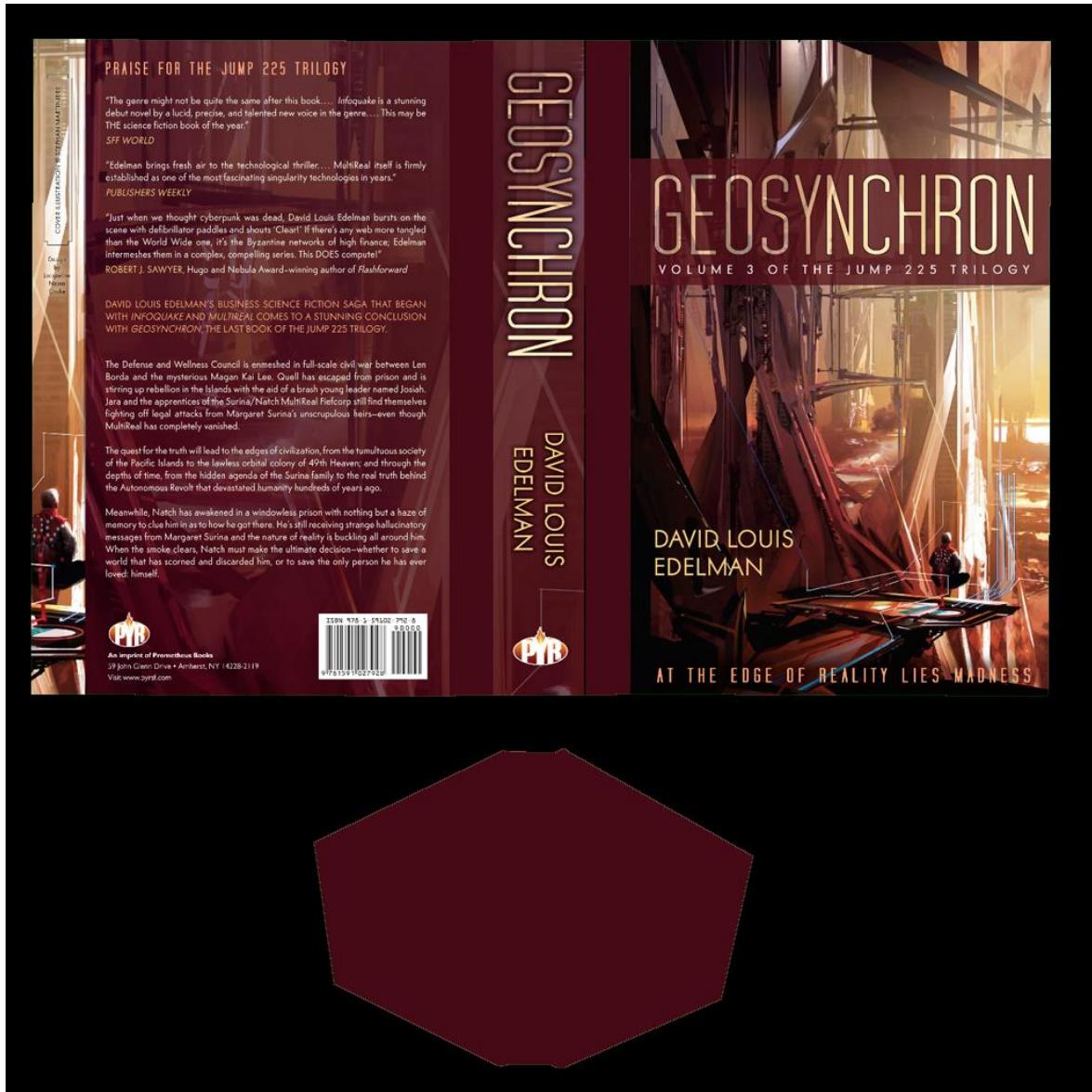


Figura 50. Ejemplo del texturizado de un libro.

Donde cada una de las partes del libro se ha separado para poder aplicar la textura en 2D al modelado en 3D.

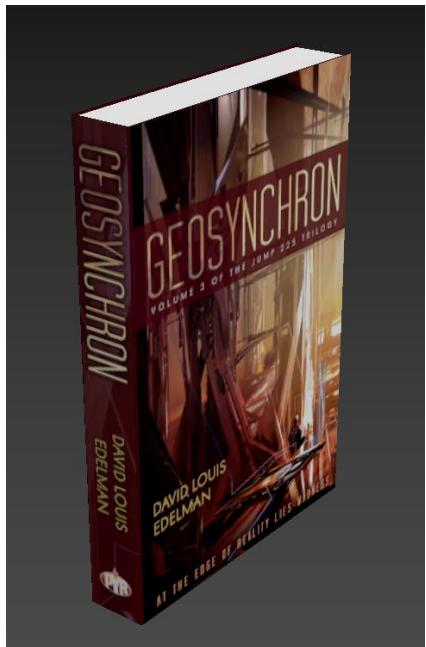


Figura 51. Libro con textura aplicada.

Como podemos observar en la figura 48, la textura del libro capturada desde 3dsMax tan sólo contiene el mapa de imagen aplicado al modelo.

Algunos tipos de textura como esta no necesitan distintos tipos de mapas que crean la ilusión de volumen como puede ser un mapa de normales, pero en otros tipos de texturizados sí que necesitan este tipo de mapeados.

Por ejemplo, en el caso de las paredes de la cocina donde se necesita mostrar el relieve de las baldosas se han tenido que utilizar las imágenes mostradas anteriormente en el ejemplo de cada uno de los mapas. Específicamente estamos hablando de las figuras 43, 44, 45, y 46.

El resultado de la combinación de estas texturas es muy realista y de muy buena calidad, como podemos ver en la siguiente imagen:



Figura 52. Ejemplo del resultado final de los materiales con los diferentes mapas de texturas.

5.3.2.2. Diseño y arte sonoro

Todos los sonidos de este videojuego han sido obtenidos de manera gratuita con licencia Creative Commons. Para aplicar estos sonidos al videojuego se han tenido que modificar previamente para crear la experiencia deseada.

A continuación, se mostrarán algunos de los sonidos con características interesantes que han sido utilizados, tanto su modificación en Audacity como su funcionamiento dentro de Unreal.

5.3.2.2.1. Sonido llantos fantasma

Los llantos que escuchamos del fantasma son los llantos de una mujer que han sido transformados en Audacity con varias herramientas hasta conseguir convertirlo en un sonido extremadamente depresivo y desagradable.

Dentro del videojuego simplemente se trata de un *AmbientSound Actor* que se mueve con el enemigo y posee propiedades de atenuación y espacialización del sonido para saber de dónde viene y lo lejos que se encuentra.

5.3.2.2.2. Sonido puerta evento baño

El sonido de la puerta del evento del baño ha sido uno de los más complicados de crear puesto que ha requerido la mezcla de distintos sonidos modificados en un tiempo exacto.

Posteriormente dentro de Unreal se ha tenido que animar tanto el movimiento como la manivela de la puerta para corresponder al sonido exactamente. Esto ha requerido trabajar dentro de una animación marcada por el tiempo con una precisión extremadamente alta, pues el conjunto de rotaciones y movimiento es complicado de realizar.

El resultado final ha sido satisfactorio y la experiencia que surge del evento ha merecido la pena.

5.3.2.2.3. Sonido de pasos y el sistema de detección de superficie

El sonido de los pasos también ha resultado interesante de construir pues uno de los requisitos es que fuera distinto en cada superficie que pisáramos, además de que era necesario añadir variaciones entre cada paso para no crear la sensación de repetición en el jugador.

Para ello, se creó un blueprint que realiza un *raycast* desde nuestra cabeza hacia abajo hasta que choca con el primer objeto, que se tratará del suelo (esto incluye alfombras). Cada uno de los modelados del suelo posee un material distinto dependiendo de la superficie que tratemos. Por ejemplo, la madera posee un material con apariencia de madera, el granito un material que representa el granito, etc. Aunque esto es obvio dado que no todas las partes de la casa van a poseer el mismo suelo, sirve para introducir una nueva herramienta que incorpora Unreal, se trata de las *Physical Surfaces*.

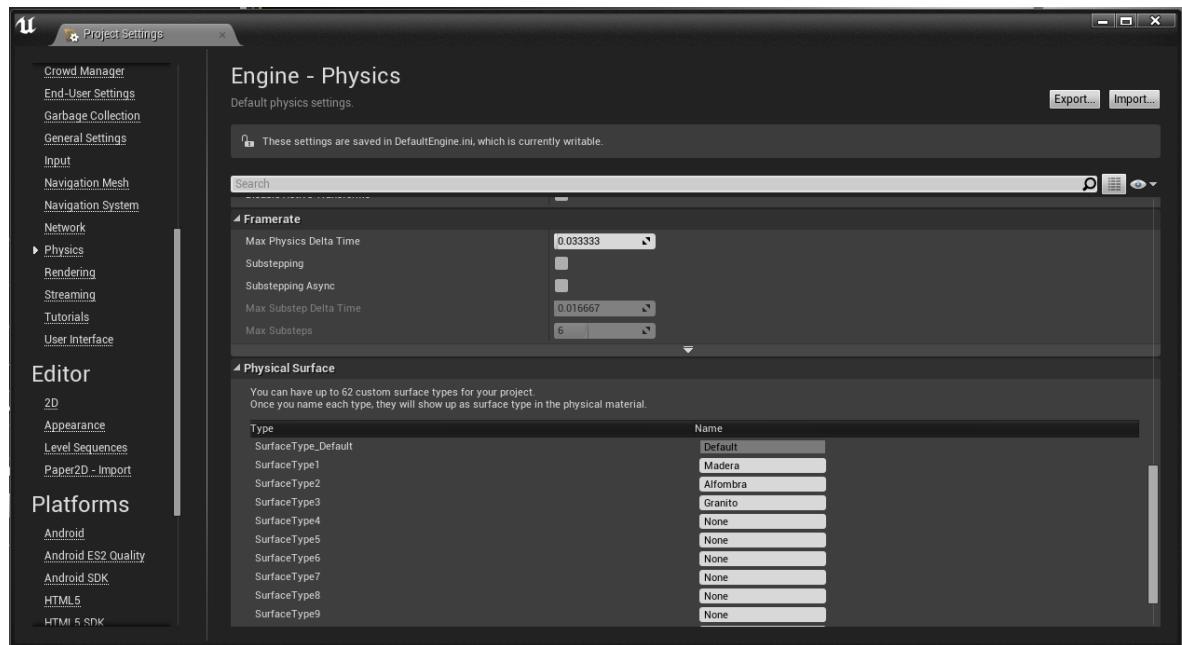


Figura 53. Opciones de Unreal sobre las Physical Surfaces.

Unreal nos permite crear distintas *Physical Surfaces*, esto se trata de una propiedad que se le puede aplicar a los materiales que permite cambiar su deslizamiento, peso, etc., pero nosotros tan sólo lo utilizamos como si fueran grupos. De esta manera, a todos los materiales que al caminar sobre ellos deban sonar como madera les asignaremos la *Physical Surface* “Madera”. Repetiremos lo mismo para los distintos materiales.

Una vez hecho esto, el script de detección de superficies funcionará perfectamente.

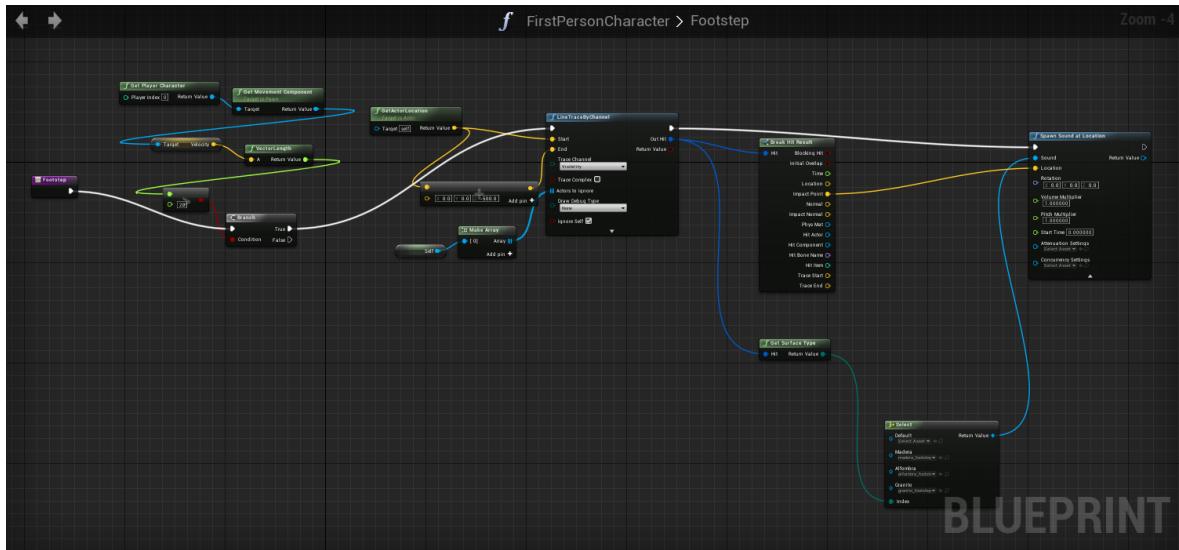


Figura 54. Blueprint de la función `Footstep`, de la clase `FirstPersonCharacter`.

Además de esto, también se han creado distintas *Sound Cues*, una para cada sonido. Por ejemplo, la alfombra tiene un *Sound Cue* muy sencillo:

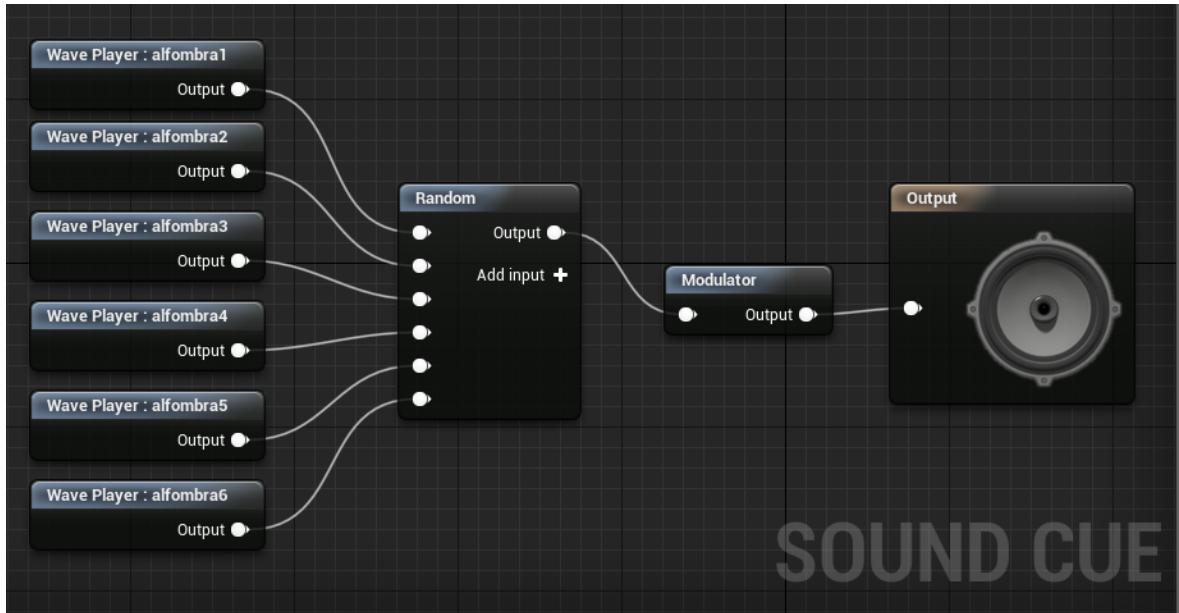


Figura 55. Sound Cue de los pasos en alfombra.

Simplemente cogemos cada uno de los sonidos de los pasos en alfombra creados en Audacity y los conectamos a un nodo *Random*. Esto hará que cada vez que llamemos a esta *Sound Cue* se ejecute uno de los sonidos de manera aleatoria.

Posteriormente, la salida del nodo *Random* se conecta a un *Modulator* para introducir de manera fácil una pequeña variación en el tono y volumen del sonido ejecutado.

Otros sonidos de pasos como los de la madera son más complicados.

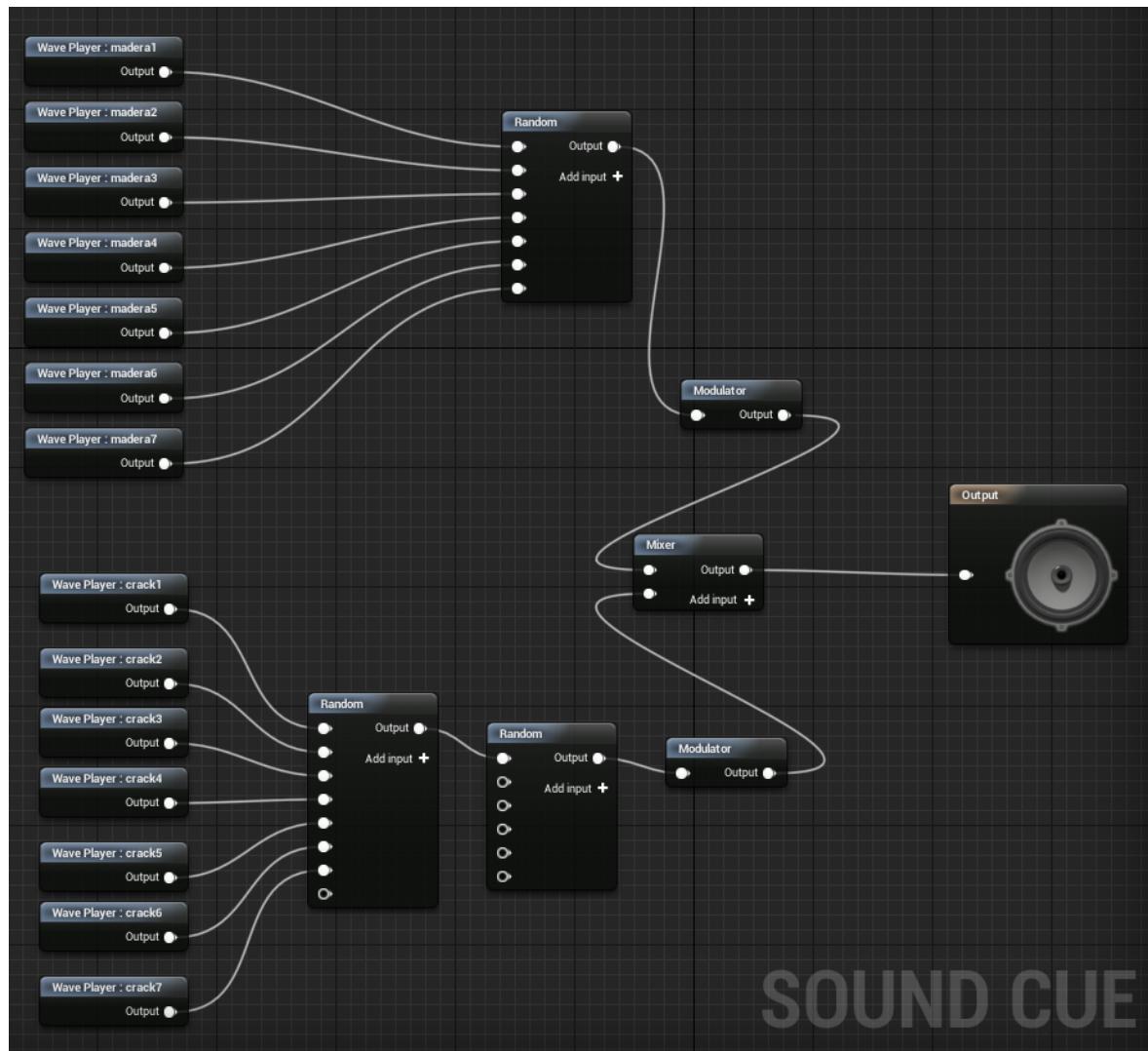


Figura 56. Sound Cue de los pasos en madera.

Funciona de manera similar al sonido de los pasos en la alfombra pero con un añadido. Queremos que de vez en cuando suenen los típicos “cracks” que se producen al caminar sobre un suelo de madera, es por ello que con cada paso hay una posibilidad de que suene uno de estos sonidos, de igual manera seleccionados aleatoriamente entre un amplio repertorio creado en Audacity.

5.3.2.2.4. Sonidos ambiente y la atenuación del sonido

Dentro del videojuego podemos encontrar dos tipos de sonidos ambiente, unos sin atenuación del sonido que suenan al mismo volumen en todo el nivel, y otros con atenuación que tan sólo podemos escuchar si nos acercamos a la fuente del sonido.

Los sonidos sin atenuación dentro de Unreal son llamados sonidos 2D, puesto que no están espacializados y funcionan como una canción de fondo que está constantemente activa hasta que termina, o en caso de que se seleccione la opción bucle, sonará indefinidamente hasta que decidamos pararla. Un ejemplo claro de este tipo de sonido es la tormenta que escuchamos de fondo, también obtenida gracias a la gran comunidad de FreeSound.



Figura 57. Captura InGame de la cocina, donde escuchamos Ambient Sounds como la nevera o la luz del techo.

Los sonidos con atenuación son más específicos y añaden un nivel de detalle muy alto al videojuego. Estos sonidos aportan un realismo espectacular ya que poseen una atenuación del sonido dependiente de la distancia desde el sonido al jugador. El volumen del sonido se controla por el proceso

algorítmico que nosotros decidimos dentro de las opciones del actor *Ambient Sound*. Algunos ejemplos de este sonido en **The Loudest Sound** son el sonido de las gotas de lluvia que podemos escuchar al acercarnos a las ventanas, o el sonido que produce la nevera cuando nos encontramos en la cocina.

5.3.3. Creación de nivel de juego

En esta sección se va a describir con detalle cómo se ha construido el nivel, todos los pasos que se han llevado a cabo y los cambios que ha experimentado el nivel conforme se probaba el *gameplay*.

5.3.3.1. Resumen

La casa en la que nos encontramos es el único nivel del videojuego. Se trata de una clásica casa americana con una grandísima cantidad de detalles. El objetivo de crear un entorno realista y tan detallado era uno de los objetivos principales del videojuego, pues en este tipo de géneros es extremadamente importante.

5.3.3.2. Idea, adquisición de planos y primeros cimientos

Al comienzo del desarrollo se optó por crear el nivel libremente, pero tras varias pruebas, no se conseguía dar la apariencia de un lugar realista, daba la sensación de ser falso, como montado por piezas sin mucho sentido. Es por ello que se procedió a buscar planos de casas americanas.

Una vez se encontró una casa que cumplía con nuestras expectativas, se imprimieron los planos y se recreó la casa a medida dentro de Unreal, con algunos pequeños cambios destinados al *gameplay*, utilizando geometría básica incorporada por defecto en Unreal.



Figura 58. Planta baja, plano real



Figura 59. Planta baja, Unreal

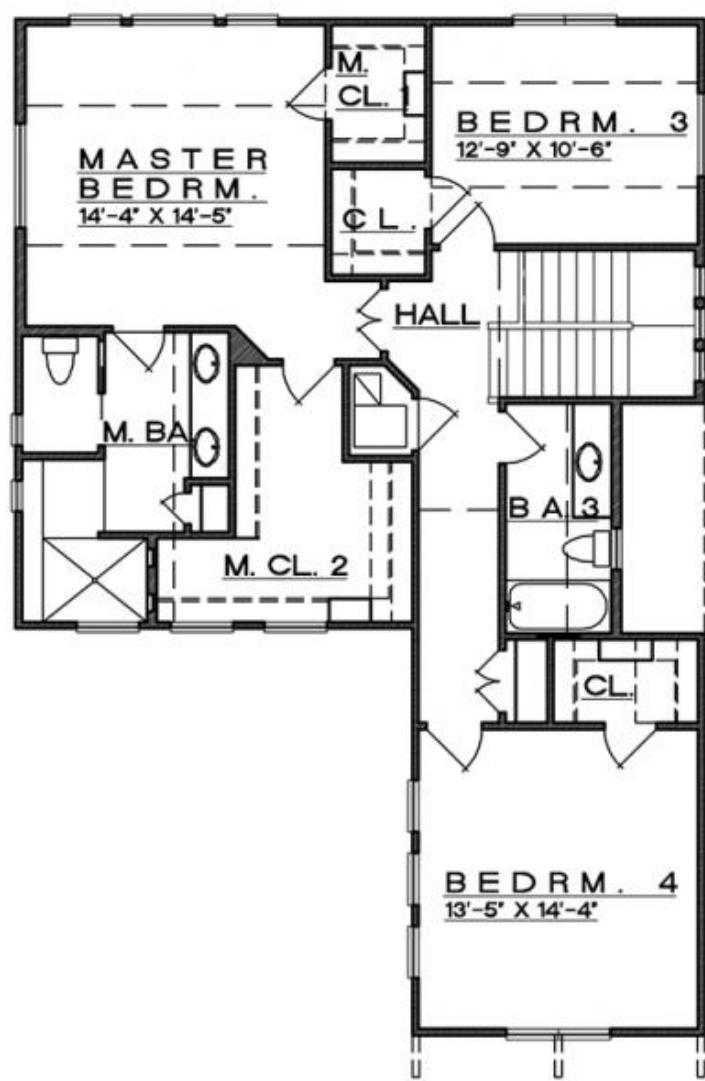


Figura 60. Primera planta, plano real



Figura 61. Primera planta, Unreal

5.3.3.3. Cimientos con entornos modulares

Una vez estaban creados todos los entornos modulares, incluyendo esquinas y paredes de distintos tamaños, se comenzó a construir de nuevo el nivel, pero esta vez utilizando los distintos tipos de módulos para las paredes, columnas, techos, etc.

La creación de los **entornos modulares** está detallada en la sección “**Modelados y optimizaciones**”.

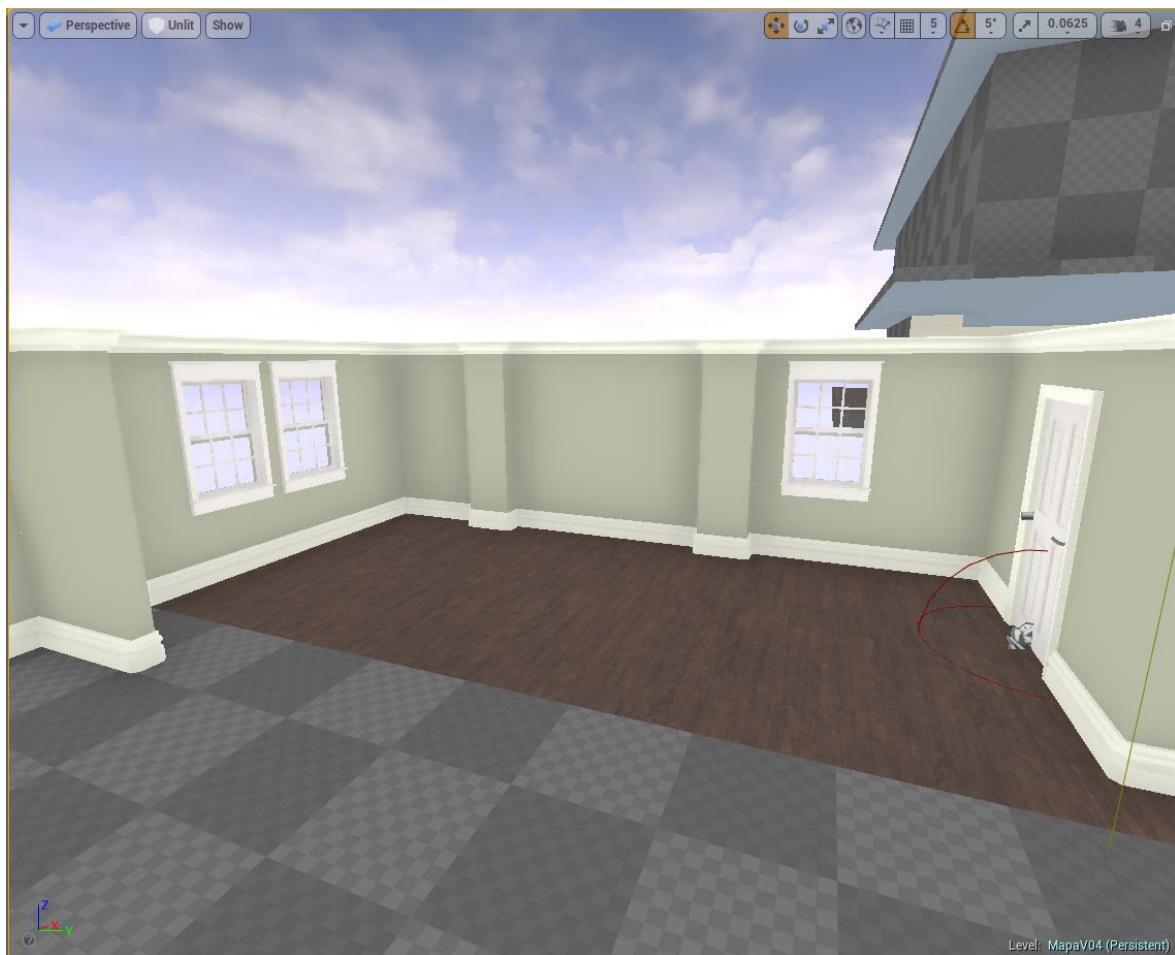


Figura 62. Parte del nivel en construcción, vista sin iluminación

Como se puede ver en la imagen, también se crearon módulos para las ventanas y las puertas. También se iban probando diferentes texturas en los modelados para comprobar que su Unwrap estaba realizado correctamente.

En esta fase además se realizaron algunas pruebas sencillas de iluminación y de texturizados dinámicos o Decals, de los cuales tenemos un ejemplo en la siguiente imagen. Los Decals de los que hablamos son las manchas de la pared.

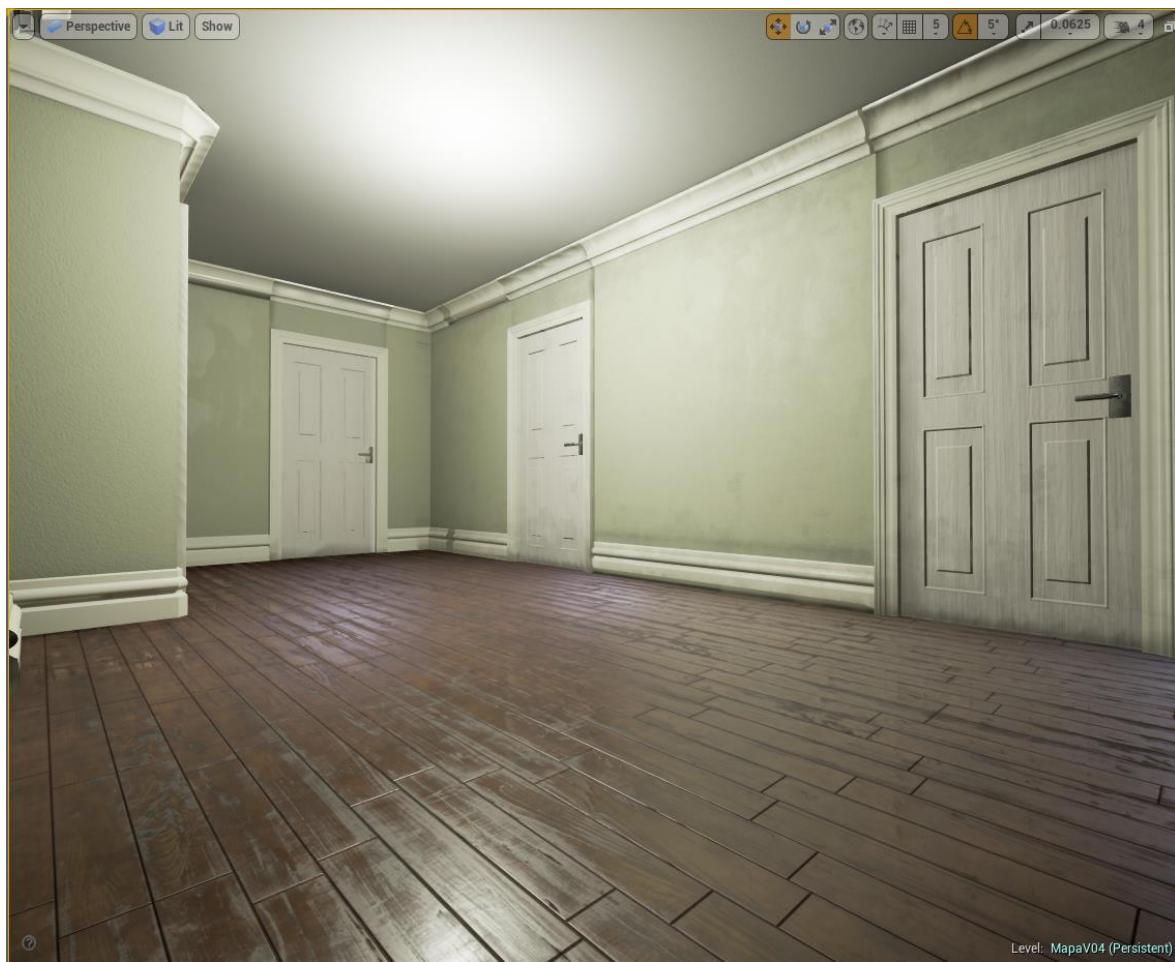


Figura 63. Parte del nivel aún en construcción con algunas pruebas de iluminación y texturizado

5.3.3.4. Cimientos provisionales para el desarrollo

En esta fase se completaron los cimientos de la casa al completo, con las secciones de la misma que estaban pensadas para el *gameplay*, colisiones correctas y modelos con un buen Unwrap. Este nivel se utilizó para proseguir con el desarrollo de mecánicas y juego en general.

Podemos ver algunos modelados nuevos como la escalera.



Figura 64. Escaleras en construcción

Así como los cimientos del piso final, con alguna iluminación, en ese momento dinámica.



Figura 65. Pruebas de iluminación en el pasillo de la primera planta

5.3.3.5. Nivel final

El nivel final se ha completado, modificado, arreglado y mejorado numerosas veces para crear la experiencia de usuario que se buscaba. Muchas zonas han disminuido de tamaño por decisión de diseño de juego. Las texturas se han trabajado mucho (sección Texturizado), así como la iluminación y los modelados (sección Modelados y optimizaciones).

En este nivel también se han añadido una grandísima cantidad de elementos nuevos, entre ellos incluidos los eventos y los sonidos ambiente, de una importancia altísima.

Algunas capturas del nivel final:



Figura 66. Comedor, nivel final InGame.



Figura 67. Baño, nivel final InGame.

5.3.4. Desarrollo de mecánicas de juego

Todas las mecánicas del videojuego se han realizado utilizando la programación visual que nos ofrece Unreal, o su llamado sistema de Blueprints. Ha sido una gran experiencia aprender a utilizar este sistema, el cual ha resultado ser altamente intuitivo y fácilmente adaptable a mis necesidades.

Pueden parecer pocas mecánicas a primera vista, pero una vez dentro del juego conectan muy bien entre ellas y permiten una experiencia muy gratificante. Cabe destacar que estas mecánicas han sido elegidas escrupulosamente en el proceso de diseño del videojuego y no se incluyen más porque el videojuego demanda no tener más.

5.3.4.1. Blueprint First Person Character

El blueprint **First Person Character** se trata del blueprint principal, el que contiene la mayor cantidad de variables, funciones y referencias entre otros blueprints. Es por ello que es el primero que se va a explicar, ya que posteriormente muchos blueprints accederán a variables o se comunicarán con este blueprint.

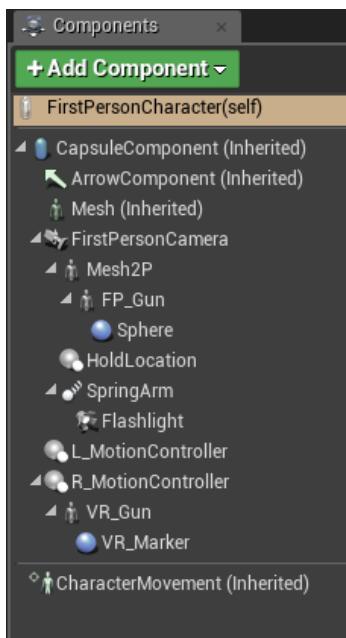


Figura 68. Componentes del blueprint **FirstPersonCharacter** listados.

Tiene muchos componentes, muchos de ellos añadidos automáticamente por defecto en Unreal. Los más importantes son: **CapsuleComponent**, **CharacterMovement**, **FirstPersonCamera**, **HoldLocation**, **SpringArm** y **Flashlight**.

CapsuleComponent controla la colisión de nuestro personaje y tiene una forma de cápsula con un radio bastante pequeño, pues dado el tipo de videojuego y el escenario, nos interesaba ser “delgados” y poder movernos por los pequeños recovecos que encontramos en la intrincada casa.

El componente **CharacterMovement** es creado automáticamente por Unreal. Este componente es el que nos permite manejar a nuestro jugador mediante controladores, ya sea ratón y teclado o un mando. Cabe destacar que cuando nos movemos, realmente se mueve el **CapsuleComponent**, pero como el resto de componentes son hijos de este elemento, se desplazan con él.

El elemento **FirstPersonCamera** es un *Camera Component*. Se trata de la cámara por la que observa el jugador. Se decidió utilizar un FOV (Field Of View o campo de visión) de 90 grados, pues la naturaleza del videojuego pide que la visión sea limitada y no podamos observar qué tenemos detrás de nosotros. Por ejemplo, en el caso de un videojuego de disparos, nos interesa tener un campo de visión mayor, para poder ver a un número mayor de enemigos.

HoldLocation simplemente es un *Scene Component*, que viene a ser un componente vacío, pero que cuenta con valores de posición, rotación y escalado. La posición de *HoldLocation* se utiliza en el blueprint *InteractablePickup*, pues se trata del lugar donde debe moverse el objeto cuando lo inspeccionemos (delante de nuestro personaje).

Por último, **SpringArm** y **Flashlight** se les considera que van juntos puesto que tienen un mismo propósito: ser la linterna de nuestro personaje. Flashlight es simplemente una *SpotLight* dinámica con un radio y distancia de atenuación determinados. *SpringArm* es un tipo de componente especial cuya finalidad en este caso es la de crear una especie de retardo entre el movimiento de la linterna y el movimiento de la cámara del jugador.

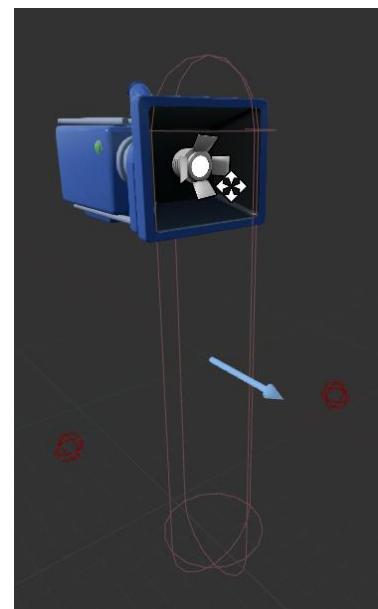


Figura 69. Componentes del blueprint *FirstPersonCharacter* vistos desde el ViewPort.

FirstPersonCharacter Variables		
Nombre	Tipo	Descripción
BaseTurnRate	<i>Float</i>	Controla la velocidad de la cámara en el eje X.
BaseLookUpRate	<i>Float</i>	Controla la velocidad de la cámara en el eje Y.
UsingMotionController	<i>Boolean</i>	Controla si se están utilizando mandos especiales para realidad virtual.
FOVinitial	<i>Float</i>	Variable que guarda el FOV de la cámara natural, para volver a ponerlo después de hacer zoom.
DeltaTime	<i>Float</i>	Hace independiente el movimiento de los FPS.

AvailableInteractions	<i>InteractableActor</i> ARRAY	Variable que contiene todos los <i>InteractableActor</i> disponibles en un instante determinado.
HighlightedInteractions	<i>InteractableActor</i>	Variable que contiene el <i>InteractableActor</i> con el que se interactuará si se hace click.
InteractionDotThreshold	<i>Float</i>	Umbral que se utiliza para destacar un objeto si estamos mirando hacia él.
BloquearMovimiento	<i>Boolean</i>	Permite denegar o permitir el movimiento del jugador.
EstaMirando	<i>Boolean</i>	Permite saber si el jugador está utilizando la mecánica de hacer zoom en un momento específico.
AnimacionMuerte	<i>Boolean</i>	Controla la ejecución de la animación de muerte del personaje.
Enemigo	<i>Enemigo</i>	Referencia al enemigo del nivel.
GrupoEventosActual	<i>Integer</i>	Número que indica el grupo de eventos en el que nos encontramos. Va cambiando conforme jugamos.
DefaultPostprocessSettings	<i>Post Process Settings</i>	Variable que guarda los valores del volumen de PostProcesado inicial.

Figura 70. Tabla de variables del blueprint *FirstPersonCharacter*.

<i>FirstPersonCharacter Functions</i>	
Nombre	Descripción
<i>ConstructionScript</i>	Constructor de la clase. Establece el FOV de la cámara y la velocidad del movimiento del jugador.
<i>UpdateCharacter</i>	Comprueba las interacciones posibles a su alrededor.
<i>CheckInteractions</i>	Comprueba el número de interacciones disponibles y posteriormente llama a las funciones para elegir la mejor.
<i>GetBestInteraction</i>	De entre las interacciones disponibles, determina la mejor.
<i>SetCurrentInteraction</i>	Convierte una “mejor interacción” en la interacción actual y muestra su interfaz, permite hacer clic, etc.
<i>AttemptInteraction</i>	Cuando hacemos clic, intenta interactuar con un objeto.
<i>AddInteraction</i>	Añade una interacción a la lista de interacciones disponibles.

RemoveInteraction	Elimina una interacción de la lista de interacciones disponibles.
Footstep	Se encarga de comprobar el material sobre el que estamos pisando y de hacer sonar la pisada.
Death	Animación de muerte del jugador.
ApagarLuces	Se encarga de apagar las luces de la casa y de cambiar los valores del volumen de post procesado.

Figura 71. Tabla de funciones del blueprint **FirstPersonCharacter**.

5.3.4.1.1. Resumen del funcionamiento de FirstPersonCharacter

Podemos llamar al blueprint **FirstPersonCharacter** el blueprint principal, puesto que es donde se guardan la mayoría de variables y funciones, además de ser ampliamente referenciado por el resto de blueprints.

Principalmente se encarga del movimiento y control de cámara del jugador, pero además se incluye el control de las interacciones, el grupo de eventos actual y referencias para cambiar el post procesado del nivel entero así como el control de la mayoría de los blueprints.

5.3.4.2. Objetos interactivos

Todos los objetos con los que se pueden interactuar heredan de una misma clase padre, el blueprint **Interactable Actor**. Esto permite que todos los objetos cuenten con una misma interfaz común y que puedan ser “apuntados” con el ratón e interactuados con el clic izquierdo.

Podremos reconocer todos los objetos interactivos al acercarnos a ellos y ver la siguiente imagen que aparece cuando los miramos:



Figura 72. Interfaz de interacción con objetos.



Figura 73. Interacción con objetos InGame.

A pesar de poseer la misma interfaz, los elementos pueden funcionar de manera completamente distinta. A continuación se explica el funcionamiento básico de cada tipo de objeto interactivo.

5.3.4.2.1. Blueprint Interactable Actor

Este es el blueprint principal del que heredan el resto de objetos interactivos. Contiene todo aquello que es esencial en todo objeto interactivo.

Está compuesto por un *Billboard Component (Icon)* que nos permite manejarlo más fácilmente al utilizarlo en el editor de niveles. Dentro de éste podemos encontrar un *Widget Component (InteractionUI)*, una *Sphere Collision (InteractionTrigger)* y un *Static Mesh Component (InteractionMesh)*.

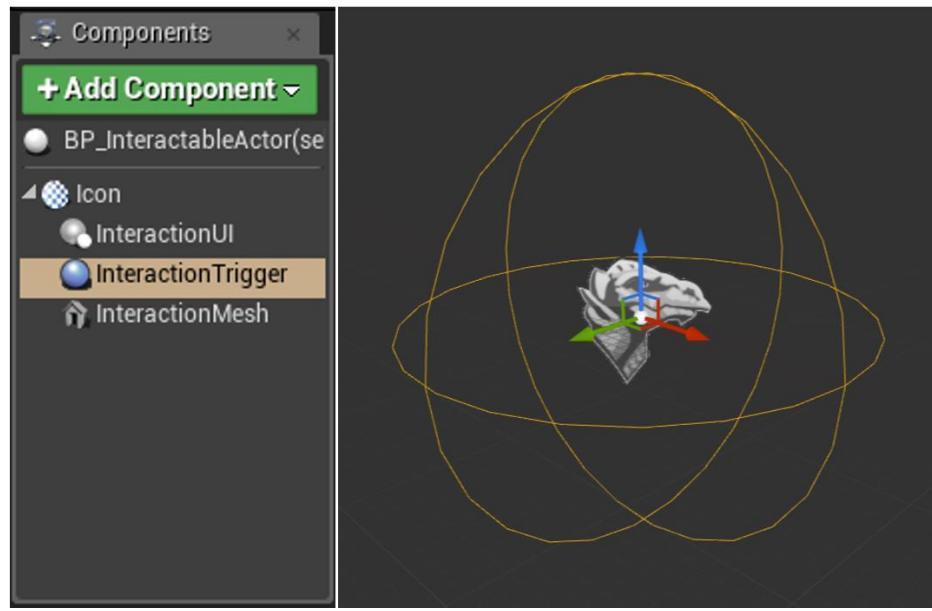


Figura 74. Componentes del blueprint `InteractableActor` listados y vistos desde el ViewPort.

InteractionUI se trata de un *widget* y nos servirá para controlar la interfaz que se dibuja (la mano que vemos al acercarnos a un objeto interactivo). **InteractionTrigger** será el volumen en el cual podemos interactuar con dicho objeto. Por último, **InteractionMesh** será el modelado de dicho objeto interactivo, aunque este *mesh* no es utilizado en alguno de los objetos interactivos por tener que realizar algunas funciones demasiado específicas, como las puertas.

InteractableActor Variables		
Nombre	Tipo	Descripción
TriggerOffset	<i>Vector</i>	Modifica la posición del TriggerSphere.
TriggerRadius	<i>Float</i>	Cambia el radio del TriggerSphere.
UIOffset		Modifica la posición del UI.
Player	<i>First Person Character</i>	Referencia al jugador para tener acceso a sus funciones y variables.
UIDescription	<i>Text</i>	Opcionalmente se puede utilizar esta variable para añadir texto a la interfaz a parte de la imagen de la mano.
ActorMesh	<i>Static Mesh</i>	Se utiliza para cambiar el modelo del <code>InteractableActor</code> .
MeshOffset	<i>Vector</i>	Modifica la posición del StaticMesh.
MeshScale	<i>Vector</i>	Modifica la escala del StaticMesh.

SePuedeInteractuar	<i>Boolean</i>	Variable que se utiliza para permitir o denegar la interacción con dicho objeto.
---------------------------	----------------	--

Figura 75. Tabla de variables del blueprint *InteractableActor*.

<i>InteractableActor Functions</i>	
Nombre	Descripción
ConstructionScript	Constructor de la clase. Establece el UI, el SphereTrigger y el StaticMesh.
ToggleHighlight	Cambia la visibilidad de la interfaz.
Interact	No hace nada. Es sobrescrita dentro de cada hijo realizando la función específica que se desee en cada caso.
SetupInteractionUI	Crea la interfaz en la posición y con los valores indicados.
SetupUIText	Si se dispone de texto, crea el texto en la interfaz.
SetupInteractionTrigger	Crea el SphereTrigger en la posición y con los valores indicados.
SetupMesh	Crea el StaticMesh en la posición y con los valores indicados.
ToggleInteractuable	Cambia la variable SePuedeInteractuar a su contrario
ChangeSePuedeInteractuar	Se encarga de cambiar la variable SePuedeInteractuar correctamente, eliminando referencias en otras clases, y utilizando la función <i>ToggleInteractuable</i> .

Figura 76. Tabla de funciones del blueprint *InteractableActor*.

5.3.4.2.1.1. Resumen del funcionamiento de *InteractableActor*

InteractableActor puede colocarse en el nivel aunque en ningún momento se utiliza de esta manera, puesto que no contiene funcionalidad. Se utiliza para crear el mismo sistema de interfaz y detección de la interacción más apropiada de los objetos interactivos.

La interacción más apropiada se decide por dos reglas:

1. El jugador se encuentra dentro del volumen de la *TriggerSphere*
2. El jugador está mirando hacia ese objeto, donde se seleccionará el objeto más cercano a su línea de visión en el caso de que hayan varios objetos.

Una vez un objeto es apropiado, la interfaz de interacción aparece encima de éste y al hacer clic se ejecuta la función ejecutar, que será sobrescrita en cada uno de los hijos de *InteractableActor*.

5.3.4.2.2. Blueprint Interactable Pickup

Este blueprint hereda de `InteractableActor` y es el responsable de aquellos objetos que podemos coger, inspeccionar y volver a dejar en su sitio dentro del juego. Este blueprint es la mecánica que hemos visto en el GDD “Interacción con objeto”.

Los componentes de este grupo son exactamente los heredados de la clase `InteractableActor`, no incluye ningún nuevo puesto que no fuese necesario.

InteractablePickup Variables		
Nombre	Tipo	Descripción
Inspeccionando	<code>Boolean</code>	Determina si se está inspeccionando un objeto en un instante determinado.
PosInicial	<code>Vector</code>	Posición inicial del objeto.
RotInicial	<code>Rotator</code>	Rotación inicial del objeto.
Animando	<code>Boolean</code>	Se utiliza para animar el movimiento del objeto desde su posición inicial hasta una posición enfrente del jugador.
LastPosHold	<code>Vector</code>	Posición global del objeto enfrente del jugador.
AnimandoOut	<code>Boolean</code>	Se utiliza para animar el movimiento del objeto desde enfrente nuestra hasta su posición inicial.
LastRot	<code>Rotator</code>	Rotación del objeto al soltarlo.
NuevoGrupoEventos	<code>Integer</code>	Si es igual a 0, no sucede nada. Si es otro número, al ejecutarse el evento y en el momento que nosotros decidamos se llamará a la función <code>NuevoGrupoDeEventos</code> .
EventosAComprobar	<code>TriggerOfEvents ARRAY</code>	Esta variable contiene los elementos del tipo <code>TriggerOfEvents</code> que deben de comprobarse al ejecutarse la función <code>NuevoGrupoDeEventos</code> .
SeCoge	<code>Boolean</code>	Determina si el objeto debe volver a su posición o si nuestro personaje se lo guarda.

UnlinkCabinet	<i>Boolean</i>	Variable auxiliar que sirve para desconectar un objeto interactivo de un cajón (si estaba vinculado a uno).
Cabinet	<i>InteractableCabinet</i>	En caso de estar conectado a un cajón, referencia a dicho cajón.

Figura 77. Tabla de variables del blueprint *InteractablePickup*

<i>InteractablePickup Functions</i>	
Nombre	Descripción
ConstructionScript	Constructor de la clase. Guarda la posición y rotación inicial del objeto dentro del nivel.
Pickup	Lleva a cabo toda la función de coger el objeto, ocultar el HUD, ponerlo delante nuestra, etc. Igualmente controla que vuelva a su posición inicial cuando dejamos de inspeccionarlo.
NuevoGrupoDeEventos	Función que se encarga de subir un nivel en el grupo de eventos actual.

Figura 78. Tabla de funciones del blueprint *InteractablePickup*.

5.3.4.2.2.1. Resumen del funcionamiento de InteractablePickup

InteractablePickup es la clase responsable de los objetos que se pueden coger e investigar en el nivel. Utilizando el sistema de interfaz y selección de *InteractableActor*, es capaz de mover el objeto desde su posición en el nivel hasta una posición delante de nuestro personaje, animándolo en el proceso. Una vez lo hemos cogido, podemos rotarlo utilizando los controles de movimiento de cámara, y dejarlo cuando hayamos terminado de inspeccionarlo. Cuando dejamos un objeto, este vuelve automáticamente a su posición y rotación iniciales.

Esta es una mecánica muy importante, pues aparte de permitir que el escenario se sienta vivo y que el jugador tiene importancia en él, es una de las principales mecánicas para contar la historia, cogiendo notas y descubriendo secretos escondidos en los objetos.

Esta mecánica también permite coger objetos y guardárnoslos, como las llaves y la linterna, aparte de tener la capacidad de ejecutar eventos.

5.3.4.2.3. Blueprint Interactable Cabinet

Los cajones de los armarios utilizan el blueprint *InteractableCabinet*. Este blueprint hereda de *InteractableActor* y funciona con una animación simple basada en el tiempo, sin tener en cuenta el motor de físicas del mundo.

A parte de los elementos heredados de *InteractableActor*, disponemos de otros dos elementos. Un *StaticMesh* que representará el modelado del cajón y, como hijo de éste último, un *SceneComponent* (componente vacío) para establecer la localización del tirador del cajón.

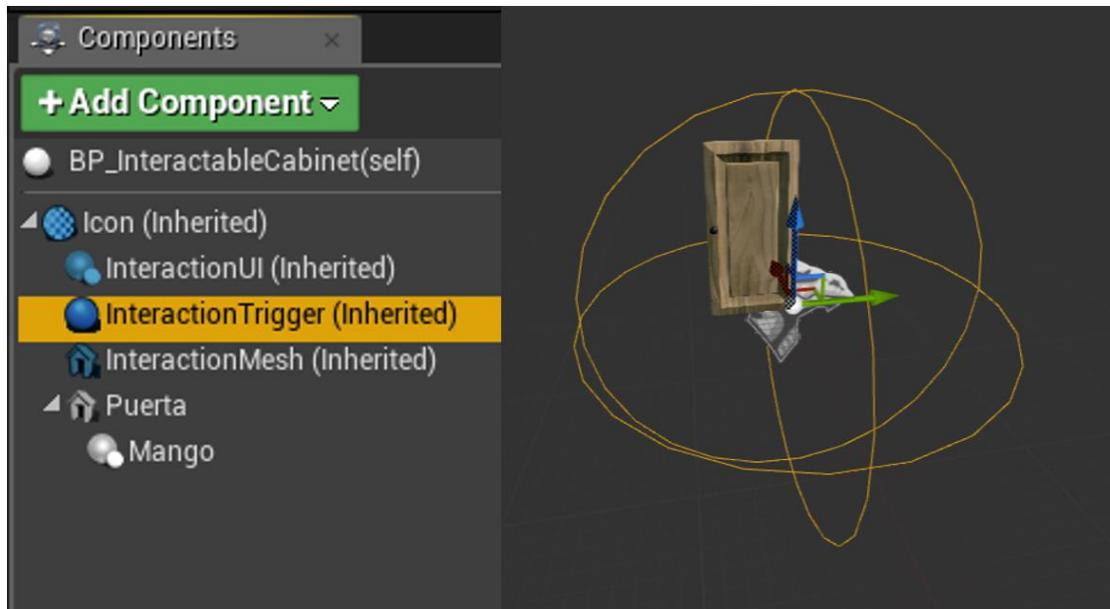


Figura 79. Componentes del blueprint *InteractableCabinet* listados y vistos desde el ViewPort.

InteractableCabinet Variables		
Nombre	Tipo	Descripción
Animando	<i>Boolean</i>	Se utiliza para animar el movimiento de la puerta.
RotFinal	<i>Rotator</i>	Rotación final que debe tener la puerta.
RotInicial	<i>Rotator</i>	Rotación inicial de la puerta en el nivel.
RotacionPuerta	<i>Float</i>	Ángulo que debe girar la puerta.
CabinetMesh	<i>StaticMesh</i>	Modelado del cajón.
MangoOffset	<i>Vector</i>	Posición del mango dentro del nivel.

SelectSound	<i>Boolean</i>	Decide si la puerta debe ejecutar el sonido de abrir o el de cerrar.
SoundAlreadyPlayed	<i>Boolean</i>	Indica si el sonido ya ha sonado.
SonidoAbrir	<i>Sound Cue</i>	Sonido que debe reproducirse al abrir el cajón.
SonidoCerrar	<i>Sound Cue</i>	Sonido que debe reproducirse al cerrar el cajón.
InteractablesDeDentro	<i>InteractablePickup</i> <i>ARRAY</i>	Referencia a los <i>InteractablePickup</i> que se encuentran tras el cajón.

Figura 80. Tabla de variables del blueprint *InteractableCabinet*.

<i>InteractableCabinet Functions</i>	
Nombre	Descripción
<i>ConstructionScript</i>	Constructor de la clase. Establece la rotación inicial, final, modelo y escala del cajón, y la posición de la interfaz.
<i>ToggleDoor</i>	Cambia entre abrir y cerrar la puerta.
<i>UpdateLocationUIandTrigger</i>	Reposiciona el SphereTrigger y la interfaz en el mango del cajón cuando este se mueve.
<i>ClearInteractablesDeDentro</i>	Elimina toda conexión con los Pickups que se encuentran tras el cajón.

Figura 81. Tabla de funciones del blueprint *InteractableCabinet*.

5.3.4.2.3.1. Resumen del funcionamiento de *InteractableCabinet*

InteractableCabinet está presente en todos los armarios que encontramos en el videojuego y corresponde a cada una de las puertas de dichos armarios. También hereda de la clase *InteractableActor*.

Funciona de manera sencilla pues no tiene en cuenta el motor de físicas del mundo. Simplemente mueve las puertas de los cajones mediante una animación basada en el tiempo y reproduce sonidos cuando se abre y se cierra.

Posee una peculiaridad, que se trata de cambiar la interactividad de los objetos que se encuentran dentro del armario, de forma que no podemos acceder a un objeto atravesando la puerta del cajón.

5.3.4.2.4. Blueprint InteractablePhysicDoor

Las puertas del videojuego funcionan mediante este blueprint, que también hereda del blueprint *InteractableActor*. Las puertas en **The Loudest Sound** funcionan de una manera especial, pues no siguen una animación fija que nos aleja del realismo cuando nos chocamos con ella y es como si fuera una pared, o peor aún, no tiene colisión. Las puertas en **The Loudest Sound** funcionan con el motor de físicas que incorpora Unreal, mediante bisagras y la aplicación de fuerzas. Es por ello que ha sido una tarea difícil programarlas.

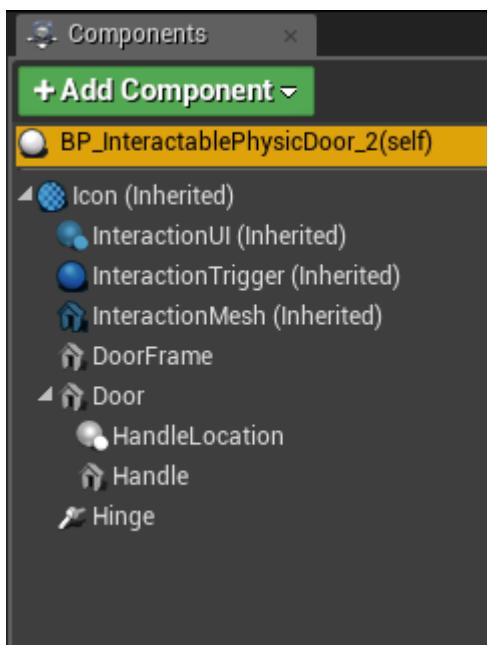


Figura 82. Componentes del blueprint *InteractablePhysicDoor* listados.

Si vamos a la pestaña del viewport comprobaremos que contiene bastantes componentes además de aquellos heredados de la clase *InteractableActor*.

DoorFrame es un StaticMesh que básicamente contiene el modelo de la pared y el marco de la puerta, que deben ser distintos al StaticMesh Door, que representa la puerta, puesto que ésta debe moverse dinámicamente.

Dentro de Door, podemos ver HandleLocation, que es un Scene Component (componente vacío) que sirve de auxiliar para la colocación de la UI. Handle es otro StaticMesh del modelo de la manivela de la puerta.

Por último, Hinge es el elemento que hace de bisagra entre la puerta y el marco de la puerta.



Figura 83. Componentes del blueprint `InteractablePhysicDoor` vistos desde el ViewPort.

InteractablePhysicDoor Variables		
Nombre	Tipo	Descripción
MovingDoor	<i>Boolean</i>	Indica si la puerta ha sido interactuada recientemente.
FuerzaPuerta	<i>Vector</i>	Indica la dirección y fuerza a aplicar a la puerta.
IsInVolume	<i>Boolean</i>	Indica si el jugador se encuentra dentro del TriggerSphere de la puerta.
RotInicial	<i>Rotator</i>	Rotación inicial de la puerta en el nivel.
FuerzaVuelta	<i>Vector</i>	Fuerza que se le debe aplicar a la puerta cuando debe cerrarse.
FuerzaAAPLICAR	<i>Vector</i>	Fuerza que se le debe aplicar a la puerta cuando debe abrirse.
Cerrando	<i>Boolean</i>	Indica si la puerta está abierta y debe cerrarse.
DuracionAbrir	<i>Float</i>	Tiempo que se aplica la fuerza de abrir la puerta.

DuracionCerrar	<i>Float</i>	Tiempo que se aplica la fuerza de cerrar la puerta.
GirarMango	<i>Boolean</i>	Indica si se está girando la manivela de la puerta.
RotMangoInicial	<i>Rotator</i>	Rotación inicial de la manivela en el nivel.
GirarIda	<i>Boolean</i>	Variable auxiliar para la animación del mango
Locked	<i>Boolean</i>	Indica si la puerta se encuentra cerrada o abierta, y la bloquea o no dependiendo del valor.
MarcoPuertaModel	<i>Static Mesh</i>	Modelo del marco de la puerta
MatPared	<i>Material</i>	Material de la pared del marco de la puerta.
OpenWithoutHandle	<i>Boolean</i>	Indica si la puerta debe abrirse sin animar la manivela de la puerta.

Figura 84. Tabla de variables del blueprint *InteractablePhysicDoor*.

<i>InteractablePhysicDoor Functions</i>	
Nombre	Descripción
ConstructionScript	Constructor de la clase. Establece la rotación inicial de la puerta y la manivela, además de los modelos de la puerta, el marco y el material de la pared.
GrabDoor	Función que establece los valores y comienza la animación de abrir la puerta.
CloseDoor	Función que establece los valores y comienza la animación de cerrar la puerta.
EngancharPuerta	Se encarga de “enganchar” la puerta al marco cuando ésta se encuentra cerca de cerrarse. Esta función es necesaria, pues el cerrado de la puerta no es automático al moverse mediante físicas.
RelocalizarTriggerYUI	Reposiciona el SphereTrigger y la interfaz en la manivela de la puerta cuando este se mueve.

Figura 85. Tabla de funciones del blueprint *InteractablePhysicDoor*.

5.3.4.2.4.1. Resumen del funcionamiento de *InteractablePhysicDoor*

Las *InteractablePhysicDoor* son uno de los elementos que más podemos encontrar dentro del nivel, y se trata de todas las puertas dentro de la casa. Este blueprint también es un hijo del blueprint *InteractableActor*.

Estas puertas son más complicadas que las de los armarios, pues funcionan con el motor de físicas que incorpora Unreal. Nuestro personaje puede chocarse contra ellas para cerrarlas, igual que empujarlas para abrirlas. El manillar de la puerta se mueve mediante animaciones basadas en tiempo y los sonidos están adecuadamente sincronizados con el movimiento de la puerta.

Las puertas pueden estar cerradas además de tener otras características como la fuerza con la que se abre o cierra la puerta, la duración del empuje, etc.

5.3.4.3. Sistema de Eventos y control de flujo de juego

Como ya se ha comentado en el GDD, el progreso del videojuego viene marcado por el sistema de eventos. Éste nos permite progresar por el nivel y controla internamente las situaciones paranormales a las que nos enfrentamos.

Los eventos se ejecutan siguiendo un orden determinado por el diseñador del videojuego y su funcionamiento está explicado a continuación.

5.3.4.3.1. Blueprint TriggerOfEvents

TriggerOfEvents es el blueprint del que heredan todos los eventos del videojuego. Es bastante sencillo pues tan sólo contiene lo básico que necesita cualquier evento.

Físicamente tan sólo posee dos componentes. Un *triggerbox* y un *billboard*. El *triggerbox* será la zona de efecto del evento y el *billboard* se utiliza simplemente para que sea más fácil de identificar y manipular el blueprint en el editor de niveles de Unreal.

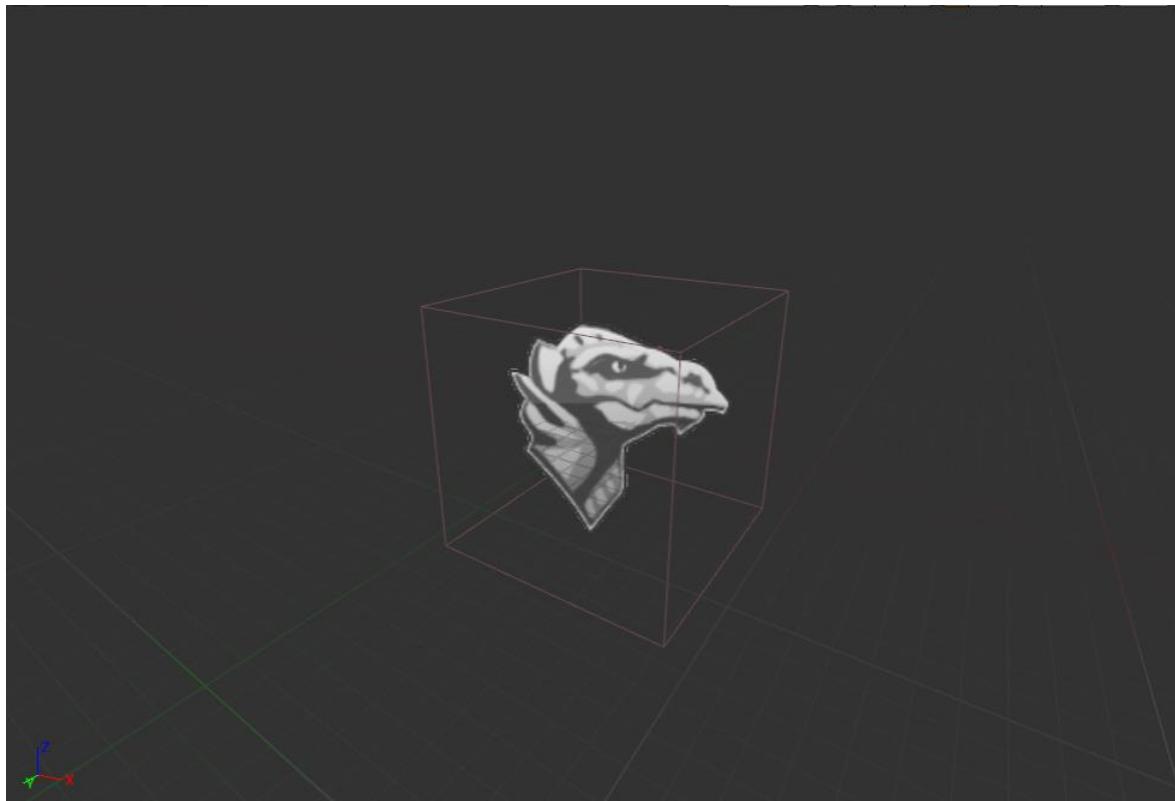


Figura 86. Componentes del blueprint `TriggerOfEvents` vistos desde el ViewPort.

Podemos encontrar múltiples variables dentro del Blueprint `TriggerOfEvents` que heredan todos los eventos. Antes de definirlas se debe hacer un pequeño inciso para poder comprenderlo adecuadamente.

En la clase (o Blueprint) más importante del juego y que es la misma que controla al jugador, `FirstPersonCharacter`, podemos encontrar una variable llamada **GrupoEventosActual** del tipo `Integer`. Esta variable es igual a 0 cuando comenzamos la partida y determina el grupo de eventos que se puede ejecutar, es decir, qué eventos reaccionarán con la interacción del jugador.

Esto se realiza de esta manera puesto que todos los eventos son *triggerbox* (volumen con forma de cubo) que detectan cuando el jugador entra en ellos. Puesto que queremos que los eventos se ejecuten en un orden concreto y que la historia tenga un orden particular, necesitamos que tan sólo actúen los eventos cuya variable **GrupoEventos** sea igual a la variable del jugador **GrupoEventosActual**.

Ahora sí, las variables y funciones que podemos encontrar dentro de la clase `TriggerOfEvents` son las siguientes:

TriggerOfEvents Variables		
Nombre	Tipo	Descripción
GrupoEventos	<i>Integer</i>	Se trata de la variable más importante de los eventos. Determina el grupo de eventos al que pertenece dicho evento, controlando su ejecución dentro del juego.
CanRepeat	<i>Boolean</i>	Determina si el evento se puede repetir más de una vez.
AlreadyPlayed	<i>Boolean</i>	Indica si el evento ya ha sido ejecutado una vez.
Player	<i>First Person Character</i>	Referencia al jugador para tener acceso a sus funciones y variables.
NuevoGrupoEventos	<i>Integer</i>	Si es igual a 0, no sucede nada. Si es otro número, al ejecutarse el evento y en el momento que nosotros decidamos se llamará a la función <i>NuevoGrupoDeEventos</i> .
EventosAComprobar	<i>TriggerOfEvents ARRAY</i>	Esta variable contiene los elementos del tipo <i>TriggerOfEvents</i> (otros eventos) que deben de comprobarse al ejecutarse la función <i>NuevoGrupoDeEventos</i> .

Figura 87. Tabla de variables del blueprint *TriggerOfEvents*.

TriggerOfEvents Functions	
Nombre	Descripción
ConstructionScript	Constructor de la clase. No hace nada.
Ejecutar	No hace nada. Es sobrescrita dentro de cada hijo realizando la función específica que se desee en cada caso.
ComprobarJugador	Comprueba si el jugador se encuentra dentro del evento en ese preciso instante.
NuevoGrupoDeEventos	Intenta cambiar a un nuevo grupo de eventos.

Figura 88. Tabla de funciones del blueprint *TriggerOfEvents*.

Existe una extensa cantidad de hijos del blueprint *TriggerOfEvents*, puesto que cada evento es prácticamente único y necesita unas variables y funciones específicas para hacerlo funcionar. Por ejemplo: si queremos hacer que suene un sonido al introducirnos en un *Trigger* tan sólo necesitaríamos el blueprint Padre y añadir una variable *Sound Cue* que refiera al sonido. Por el

contrario, si quisiéramos hacer cosas más complejas como abrir una puerta a la vez que aparece un enemigo, necesitaremos variables y funciones muy distintas.

Dada la enorme cantidad de variantes que hay y que su implementación no es muy diferente de lo comentado anteriormente, no se van a analizar más a fondo.

5.3.4.3.1.1. Resumen del funcionamiento de TriggerOfEvents

TriggerOfEvents se coloca en el nivel como un cubo invisible que detecta la colisión del jugador, y cuando este se encuentra dentro del volumen. Permite las funciones básicas que necesita todo evento, como llamar a un nuevo grupo de eventos, ejecutar la funcionalidad de dicho evento y comprobar si el jugador está dentro del volumen.

El resto de eventos heredarán de esta clase.

5.3.4.4. Blueprint Enemigo



Figura 89. Componentes del blueprint **Enemigo** vistos desde el ViewPort.

El enemigo que podemos encontrar en este videojuego se trata de un fantasma y utiliza este blueprint. Es la única IA que podemos encontrar en el videojuego y es bastante sencilla, dado que el miedo que transmite está basado en los eventos en los que aparece y no en su inteligencia, dado que el jugador no sabe qué puede o no puede llegar a hacer este fantasma.

Este blueprint, a diferencia del resto que eran del tipo Actor, es del tipo *Pawn*, que se utiliza para la IA y personajes en general. En su *viewport* podemos encontrar los componentes que lo conforman.

De manera similar al blueprint del jugador, **FirstPersonCharacter**, cuenta con un *CharacterMovement*, que se encarga de permitir al personaje moverse por el escenario.

CapsuleComponent es la cápsula de colisión de dicho personaje y es la que realmente se mueve, pero dado que el resto de componentes son hijo de este *CapsuleComponent*, todo el contenido se desplaza a la vez.

Después podemos encontrar un elemento tipo *Mesh*, que contiene el modelo y el esqueleto del jugador, junto con sus animaciones, y dentro de este observamos un *SceneComponent* (componente vacío) llamado “Cara”, que nos servirá para obtener la posición de la cara del enemigo, la cual se utilizará en la animación de muerte del personaje, puesto que queremos que se centre en la cara de fantasma mientras se realiza la animación.

Destino es un *Billboard Component*, que actúa de forma similar a un *SceneComponent*, pero nos da más facilidades para manejarlo dentro del editor de niveles. Destino nos dará la posición a la que debe caminar el fantasma.

Por último, encontramos un elemento del tipo *Ambient Sound*. Este sonido posee atenuación y espacialización, y reproduce los llantos característicos del fantasma.

El enemigo se puede mover de distintas maneras: arrastrándose, andando o corriendo. Su movimiento es parcialmente libre, limitado por el *NavMesh* creado en el nivel.

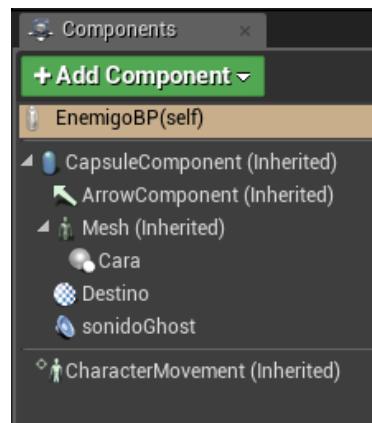


Figura 90. Componentes del blueprint *Enemigo* listados.

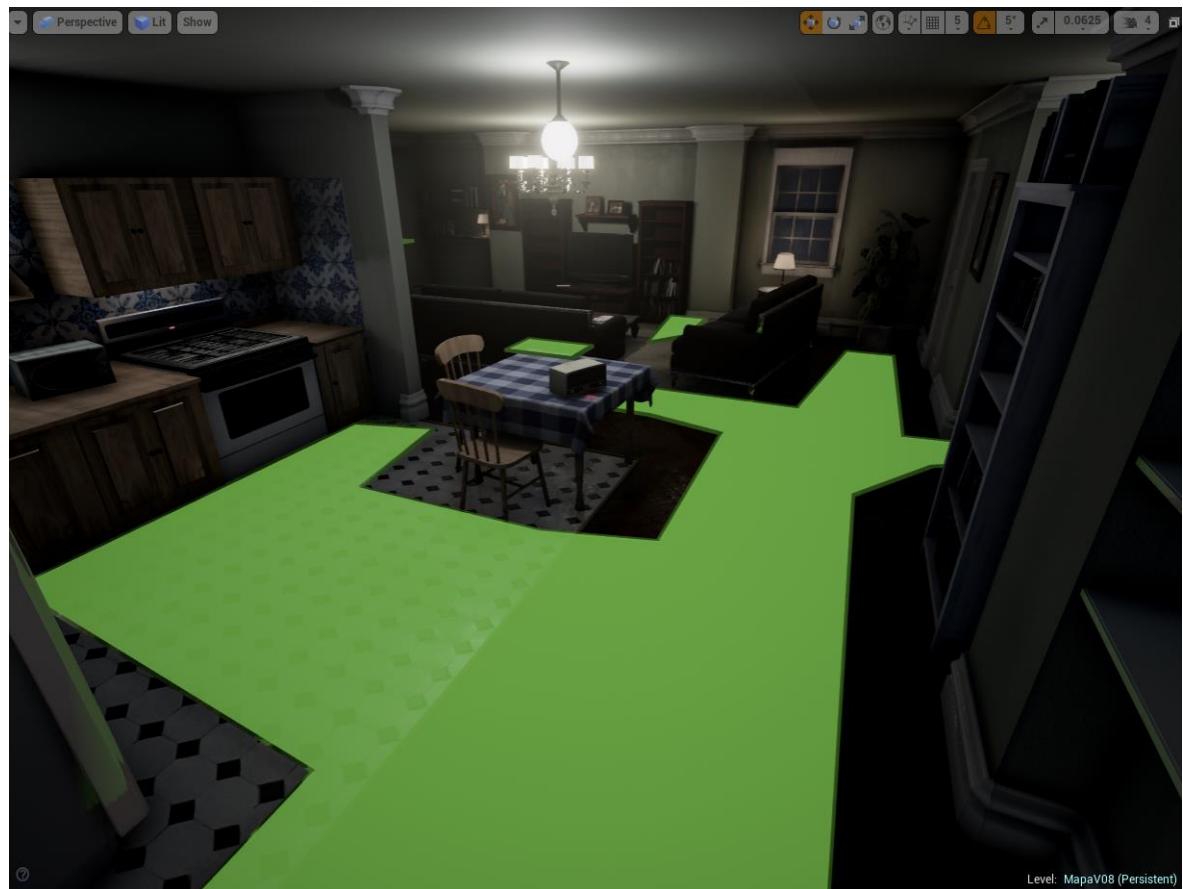


Figura 91. *NavMesh* del nivel de juego.

Enemigo Variables		
Nombre	Tipo	Descripción
BeginWalk	<i>Boolean</i>	Controla si el enemigo debe comenzar a caminar/correr.
DestinoOffset	<i>Vector</i>	Posición relativa del destino del enemigo.
DestinoLocation	<i>Vector</i>	Posición global del destino del enemigo.
Matar	<i>Boolean</i>	Indica si el enemigo debe matar al jugador al colisionar con él.
Player	<i>First Person Character</i>	Referencia al jugador para tener acceso a sus funciones y variables.
IsCrawling	<i>Boolean</i>	Establece si el enemigo debe moverse arrastrándose.
BeginCrawl	<i>Boolean</i>	Controla si el enemigo debe comenzar a arrastrarse.
WalkSpeed	<i>Float</i>	Velocidad de movimiento del enemigo.
IsPraying	<i>Float</i>	Indica si el fantasma está ejecutando la animación “Rezar”.

Figura 92. Tabla de variables del blueprint *Enemigo*.

Enemigo Functions	
Nombre	Descripción
ConstructionScript	Constructor de la clase. Establece el destino en coordenadas globales.
Walk	Manda al enemigo andar/correr hasta la posición destino.
StopEverything	Detiene el movimiento del enemigo.
Crawl	Manda al enemigo arrastrarse hasta la posición destino.
SetVolumenLlantos	Cambia el volumen del sonido de los llantos del fantasma.

Figura 93. Tabla de funciones del blueprint *Enemigo*.

5.3.4.4.1. Resumen del funcionamiento de Enemigo

La única IA que podemos encontrar en el videojuego es la de *Enemigo*. Se trata de una IA muy sencilla que tiene la capacidad de moverse por el nivel gracias a un *NavMesh* que podemos ver en la figura 88. La IA tiene la capacidad de matar al jugador y distintas formas de moverse: arrastrándose, andando o corriendo. Además de animaciones específicas como la de matar o una especial llamada “Rezar”.

Los eventos son los que controlan esta IA, son los que le dicen cuando aparecer y cuando desaparecer, además de dónde ir y si matar o no al jugador.

6. Conclusiones

Tras mucho tiempo y mucho trabajo, se ha conseguido tener un producto que podría ser considerado una demo del videojuego, pues a pesar de poder jugarse en una gran cantidad de ordenadores y de estar ampliamente testeado, tan sólo dura entre 10 y 15 minutos, por término medio.

Aun así, se trata de un trabajo de fin de grado, y no de un videojuego realizado por un equipo de desarrolladores especializados en cada sector. Es por ello que el resultado ha sido muy satisfactorio. Entre los objetivos que teníamos planteados se han cumplido todos.

Se ha realizado el GDD de un videojuego, donde se ha informado de todos los aspectos relacionados con el diseño del videojuego de una manera detallada. El videojuego puede ser jugado por cualquier persona, pues está testeado a prueba de bugs y se controla de una manera sencilla e intuitiva. También cabe notar que **The Loudest Sound** está sonorizado correctamente, atendiendo a los pequeños detalles y contando con sonidos dinámicos y de alta calidad. Además, se han creado una gran cantidad de modelos para el videojuego con programas de terceros que posteriormente se han introducido en él videojuego. De manera conjunta, se han texturizado y creado materiales de gran calidad para los objetos del videojuego, creando un aspecto realista y de muy buena calidad.

Cabe destacar que se ha construido un nivel realista y una ambientación extraordinaria, que puede ser disfrutada por todas las personas que les agrade el género de terror y se han implementado todas las mecánicas utilizando únicamente el sistema de Scripting Visual Blueprints. El videojuego también se ha compatibilizado con el dispositivo de realidad virtual Oculus Rift, a pesar de no haber sido optimizado para este tipo de uso.

Uno de los aspectos más importantes que me gustaría destacar después de haber realizado el proyecto es la aventura que ha supuesto crear el videojuego al completo con un presupuesto nulo. Realmente estoy muy agradecido a todos los desconocidos que, de forma gratuita, han compartido su arte, pues han hecho este videojuego posible.

En la realización de este proyecto he conocido a gente nueva y he contactado con algunas compañías independientes de videojuegos y desarrolladores que han resultado ser muy amables y me han enseñado una gran cantidad de técnicas nuevas.

Por último, como se ha comentado a lo largo del documento, el videojuego ha sido realizado de forma que sea fácilmente ampliable, tanto en el aspecto de niveles como en el aspecto de mecánicas. Este

proyecto se continuará en el futuro cercano con la finalidad de crear una experiencia completa, que con un poco de suerte se dé a conocer y pueda ser comercializado.

7. Anexo: Arte

En esta sección se muestran capturas del videojuego final. Además, se recomienda observar una demostración del videojuego en movimiento: <https://youtu.be/62F0hFU9XK0> .



Figura 94. Captura de *The Loudest Sound InGame* número 1.



Figura 95. Captura de *The Loudest Sound InGame* número 2.



Figura 96. Captura de *The Loudest Sound InGame* número 3.

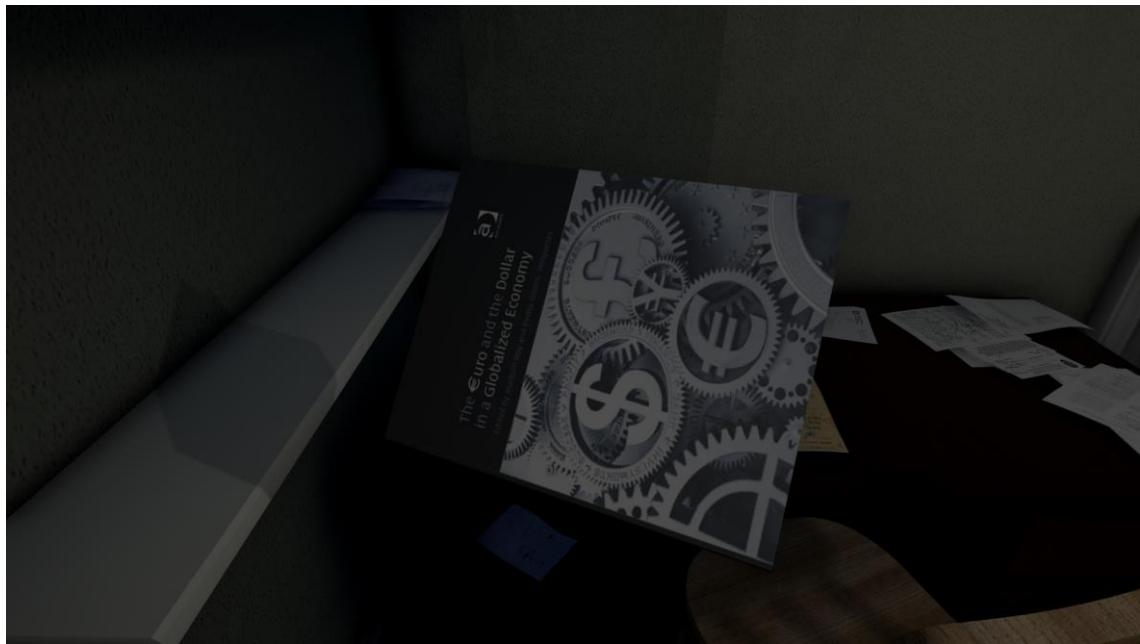


Figura 97. Captura de *The Loudest Sound* InGame número 4.



Figura 98. Captura de *The Loudest Sound* InGame número 5.



Figura 99. Captura de *The Loudest Sound* InGame número 6.



Figura 100. Captura de *The Loudest Sound* InGame número 7.



Figura 101. Captura de *The Loudest Sound* InGame número 8.



Figura 102. Captura de *The Loudest Sound* InGame número 9.



Figura 103. Captura de *The Loudest Sound* InGame número 10.



Figura 104. Captura de *The Loudest Sound* InGame número 11.



Figura 105. Captura de *The Loudest Sound InGame* número 12.



Figura 106. Captura de *The Loudest Sound InGame* número 13.



Figura 107. Captura de *The Loudest Sound* InGame número 14.



Figura 108. Captura de *The Loudest Sound* InGame número 15.



Figura 109. Captura de *The Loudest Sound InGame* número 16.



Figura 110. Captura de *The Loudest Sound InGame* número 17.



Figura 111. Captura de *The Loudest Sound InGame* número 18.



Figura 112. Captura de *The Loudest Sound InGame* número 19.



Figura 113. Captura de *The Loudest Sound* InGame número 20.



Figura 114. Captura de *The Loudest Sound* InGame número 21.



Figura 115. Captura de *The Loudest Sound InGame* número 22.



Figura 116. Captura de *The Loudest Sound InGame* número 23.



Figura 117. Captura de *The Loudest Sound InGame* número 24.



Figura 118. Captura de *The Loudest Sound InGame* número 25.



Figura 119. Captura de *The Loudest Sound InGame* número 26.



Figura 120. Captura de *The Loudest Sound InGame* número 27.



Figura 121. Captura de *The Loudest Sound InGame* número 28.



Figura 122. Captura de *The Loudest Sound InGame* número 29.



Figura 123. Captura de *The Loudest Sound InGame* número 30.



Figura 124. Captura de *The Loudest Sound InGame* número 31.



Figura 125. Captura de *The Loudest Sound InGame* número 32.

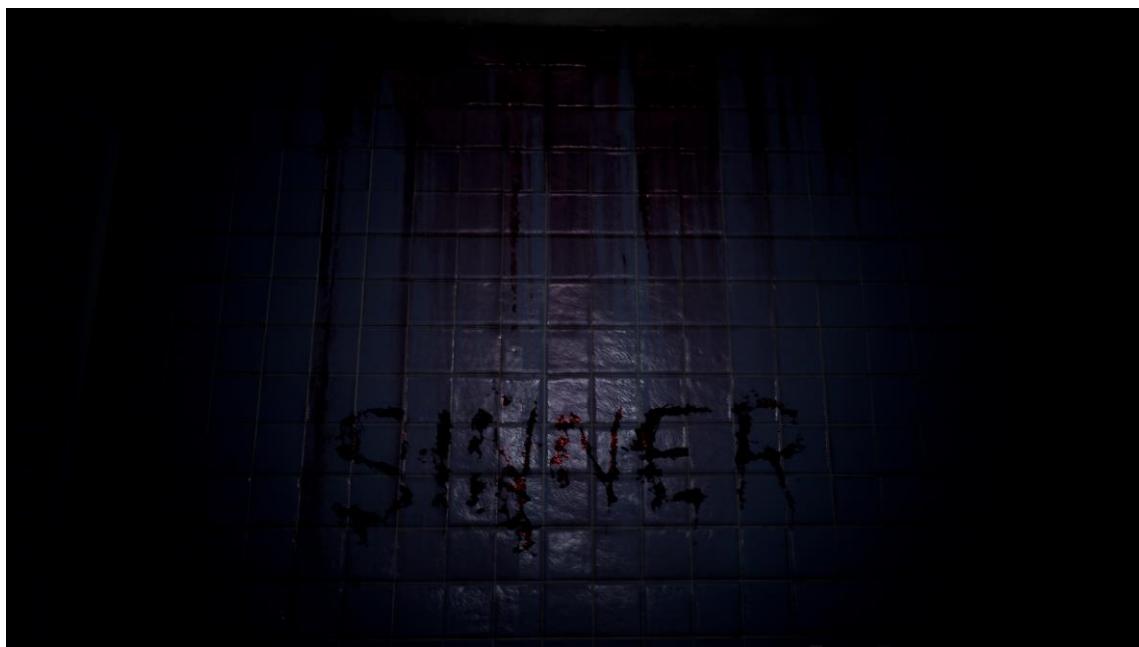


Figura 126. Captura de *The Loudest Sound InGame* número 33.

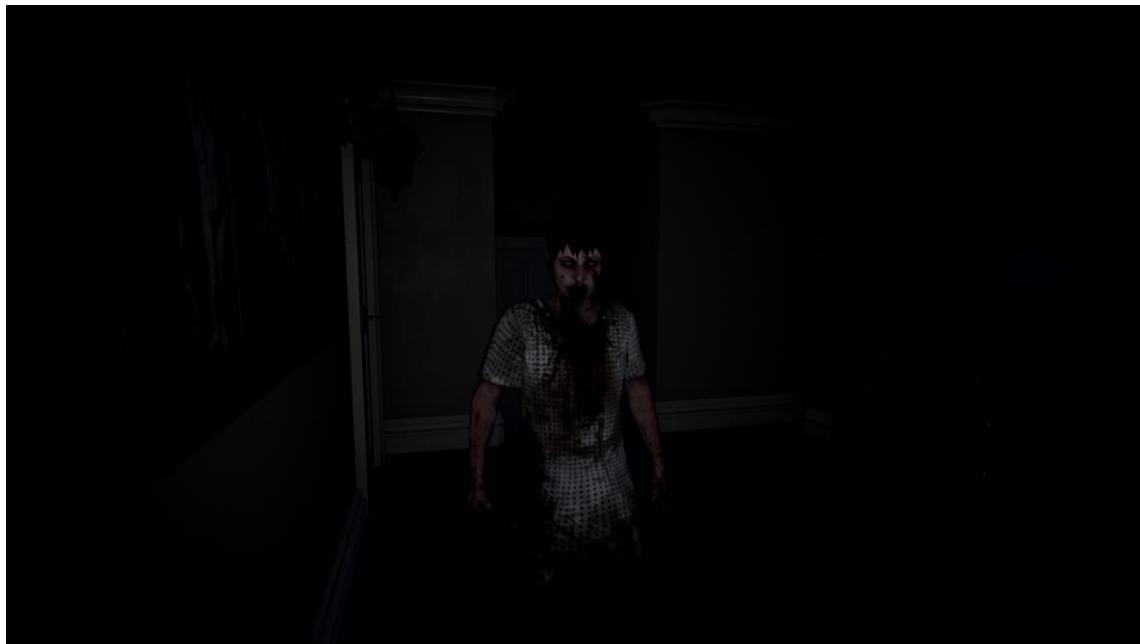


Figura 127. Captura de *The Loudest Sound InGame* número 34.

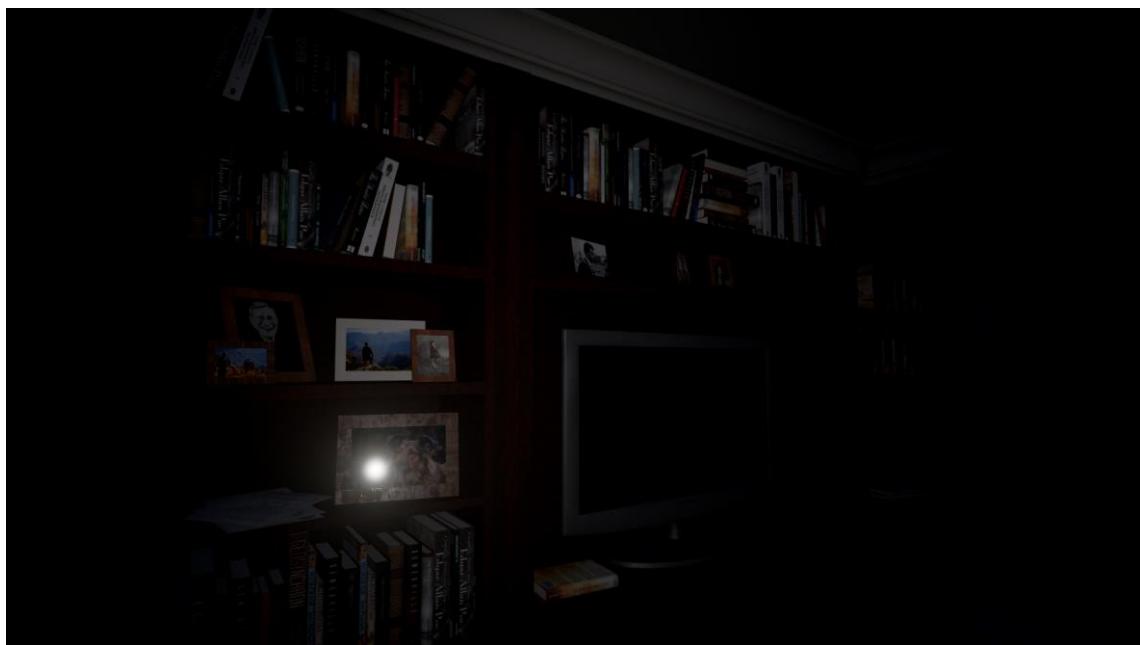


Figura 128. Captura de *The Loudest Sound InGame* número 35.



Figura 129. Captura de *The Loudest Sound InGame* número 36.

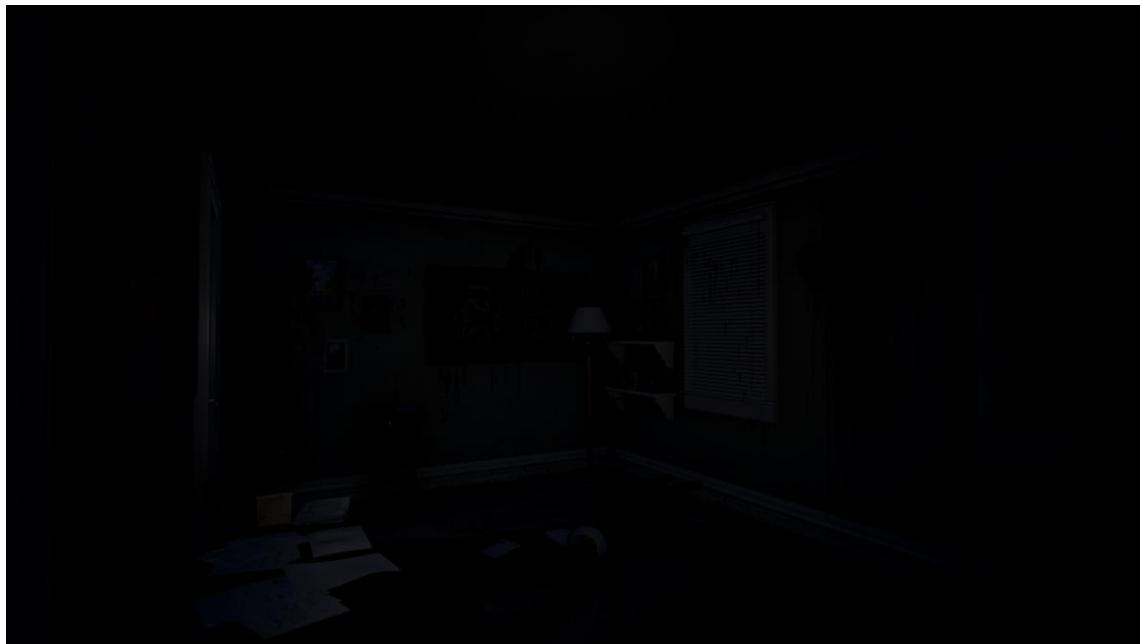


Figura 130. Captura de *The Loudest Sound InGame* número 37.



Figura 131. Captura de *The Loudest Sound InGame* número 38.

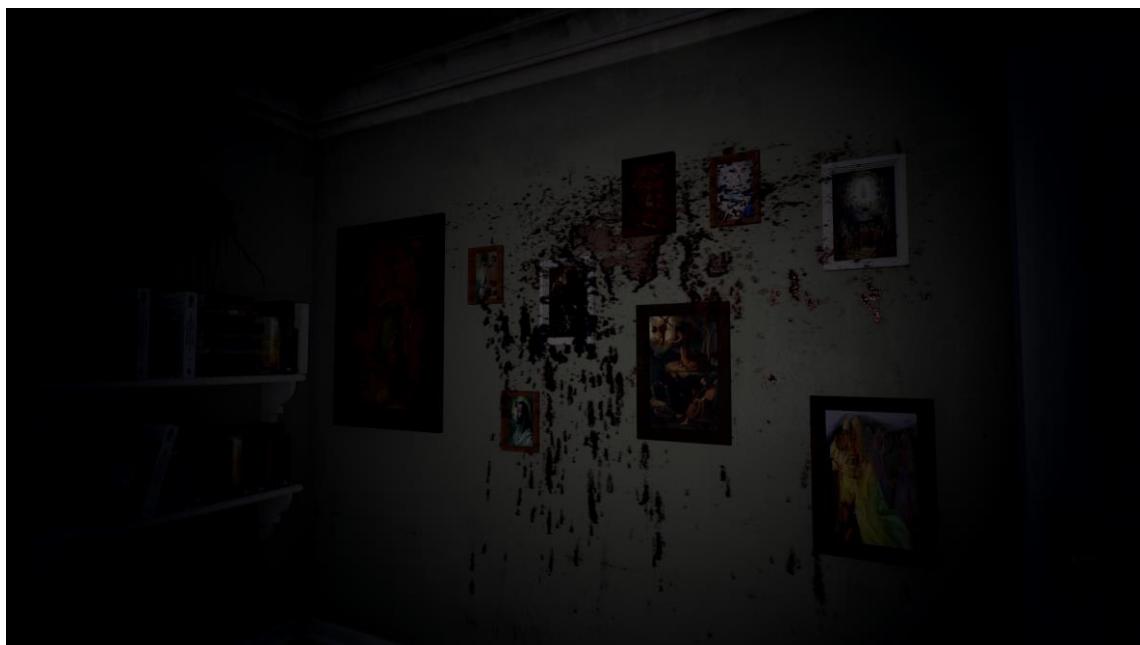


Figura 132. Captura de *The Loudest Sound InGame* número 39.

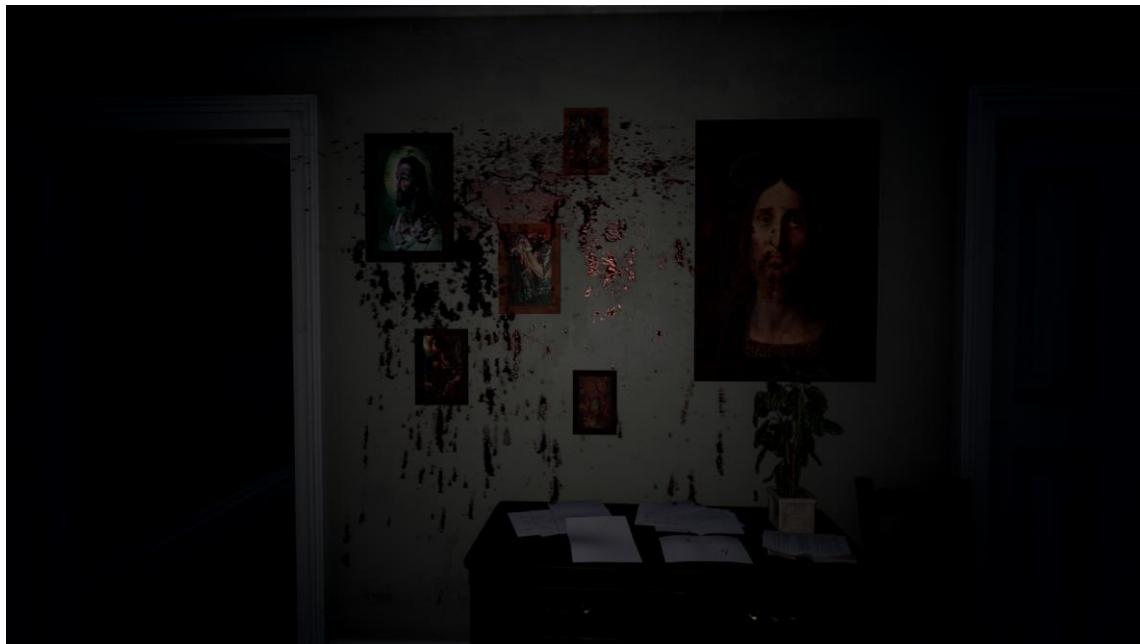


Figura 133. Captura de *The Loudest Sound InGame* número 40.

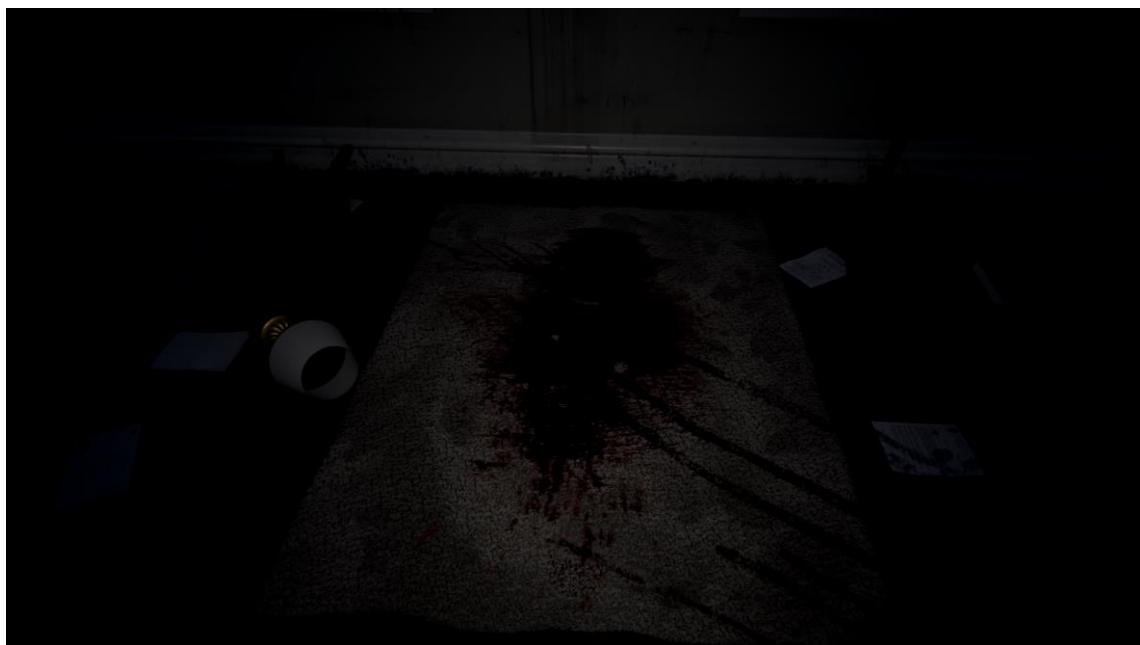


Figura 134. Captura de *The Loudest Sound InGame* número 41.

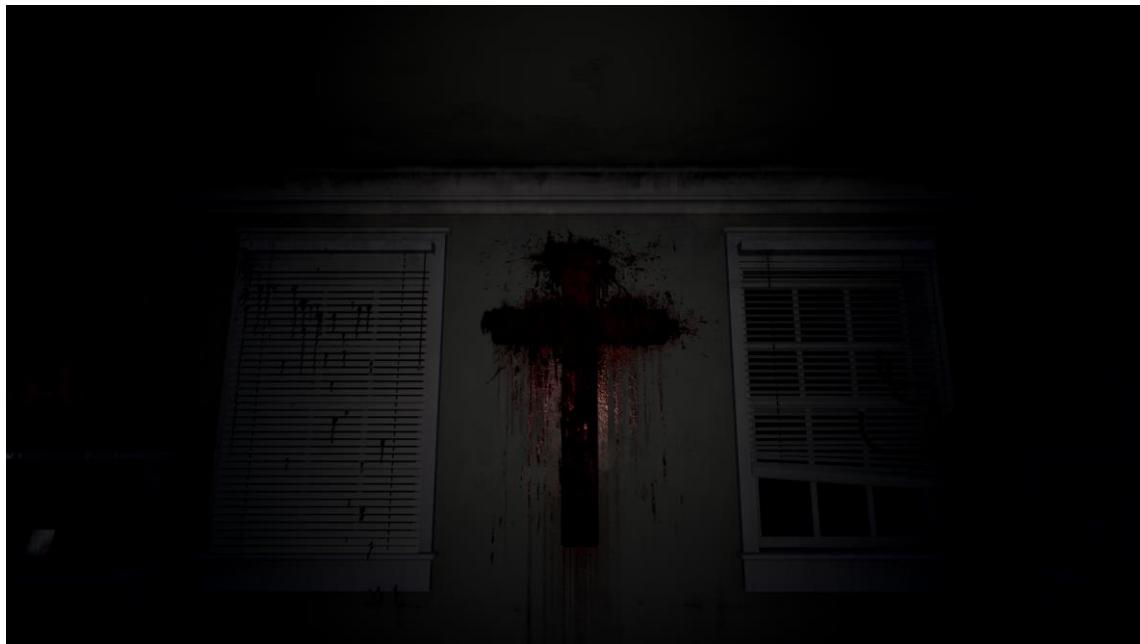


Figura 135. Captura de *The Loudest Sound InGame* número 42.

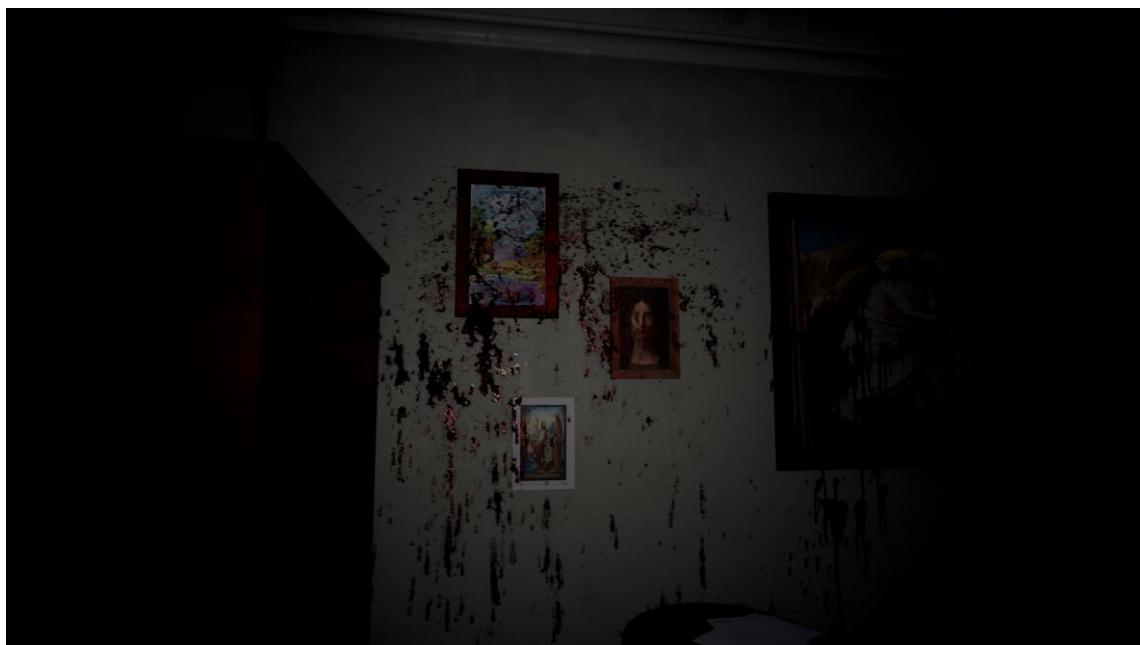


Figura 136. Captura de *The Loudest Sound InGame* número 43.

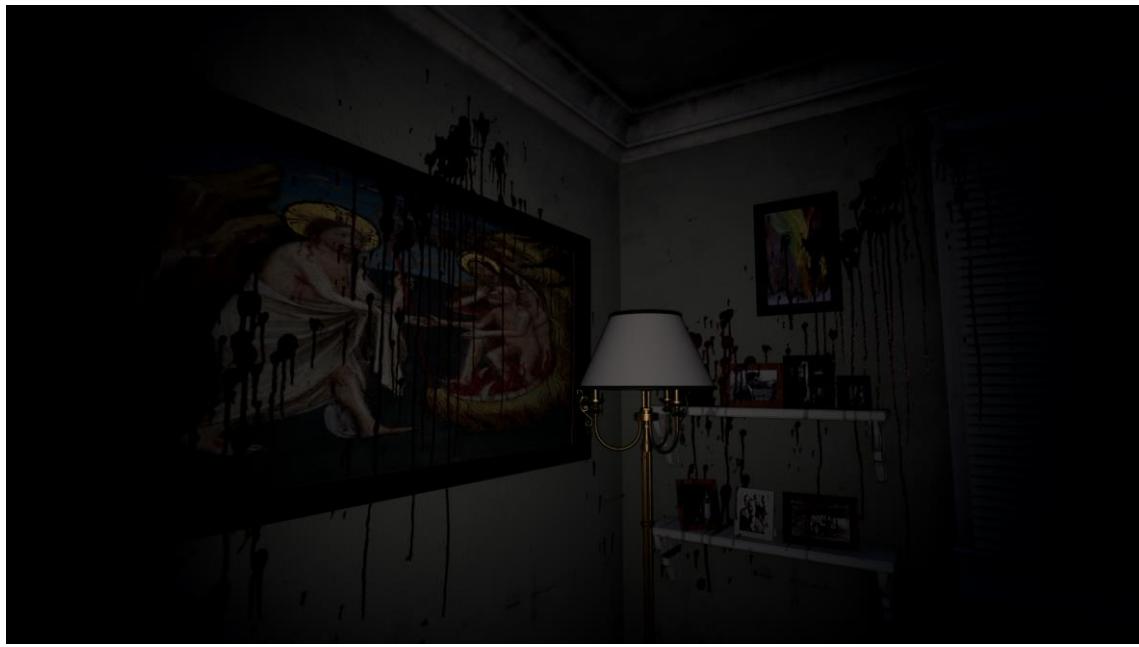


Figura 137. Captura de *The Loudest Sound InGame* número 44.

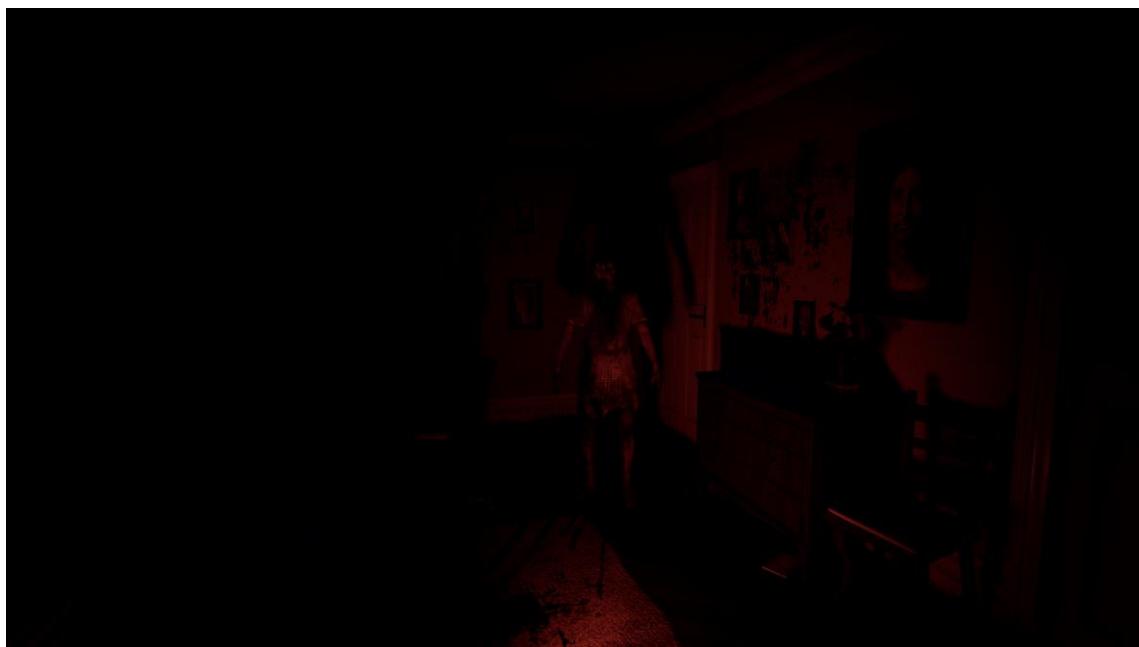


Figura 138. Captura de *The Loudest Sound InGame* número 45.

8. Bibliografía

Los géneros de los videojuegos. Recuperado el 2 de julio de 2016, del Sitio web de Wikipedia:

https://en.wikipedia.org/wiki/List_of_video_game_genres

El género Survival Horror. Recuperado el 2 de julio de 2016, del Sitio web de Wikipedia:

https://es.wikipedia.org/wiki/Survival_horror

La clasificación PEGI. Recuperado el 7 de agosto de 2016, del Sitio web de Pegi:

<http://www.pegi.info/es/index/id/955>

Documentación de Unreal Engine 4. Recuperado el 1 de septiembre de 2016, del Sitio web de Unreal

Engine: <https://docs.unrealengine.com/latest/INT/>

El método Kanban. Recuperado el 26 de julio de 2016, del Sitio web de Wikipedia:

<https://es.wikipedia.org/wiki/Kanban>

El concepto de videojuego. Recuperado el 15 de julio de 2016, del Sitio web de Wikia:

[http://es.nintendo.wikia.com/wiki/Videojuego_\(concepto\)](http://es.nintendo.wikia.com/wiki/Videojuego_(concepto))

Videojuego P.T. Recuperado el 20 de agosto de 2016, del Sitio web de Wikipedia:

[https://en.wikipedia.org/wiki/P.T._\(video_game\)](https://en.wikipedia.org/wiki/P.T._(video_game))

Videojuego Allison Road. Recuperado el 20 de agosto de 2016, del Sitio web de Wikipedia:

[https://en.wikipedia.org/wiki/Allison_Road_\(video_game\)](https://en.wikipedia.org/wiki/Allison_Road_(video_game))

Videojuego Visage. Recuperado el 20 de agosto de 2016, del Sitio web de Wikipedia:

[https://en.wikipedia.org/wiki/Visage_\(video_game\)](https://en.wikipedia.org/wiki/Visage_(video_game))

Videojuego Layers Of Fear. Recuperado el 20 de agosto de 2016, del Sitio web de Wikipedia:

https://en.wikipedia.org/wiki/Layers_of_Fear

Videojuego Amnesia: The Dark Descent. Recuperado el 20 de agosto de 2016, del Sitio web de

Wikipedia: https://en.wikipedia.org/wiki/Amnesia:_The_Dark_Descent

Videojuego SOMA. Recuperado el 20 de agosto de 2016, del Sitio web de Wikipedia:

[https://en.wikipedia.org/wiki/Soma_\(video_game\)](https://en.wikipedia.org/wiki/Soma_(video_game))

Videojuego Alien: Isolation. Recuperado el 20 de agosto de 2016, del Sitio web de Wikipedia:

https://en.wikipedia.org/wiki/Alien:_Isolation

Videojuego Outlast. Recuperado el 20 de agosto de 2016, del Sitio web de Wikipedia:

<https://en.wikipedia.org/wiki/Outlast>

